A multi-swarm optimizer based fuzzy modeling approach for dynamic systems processing

Ben Nia,b,*, Yunlong Zhua, Xiaoxian Hea,b, Hai Shenia,b

aShenyang Institute of Automation, Chinese Academy of Sciences, Shenyang 110016, China
bGraduate School of the Chinese Academy of Sciences, Beijing 100049, China

Received 20 September 2006; received in revised form 26 April 2007; accepted 19 May 2007

Communicated by D.-S. Huang

Abstract

Inspired by the phenomenon of symbiosis in natural ecosystems a multi-swarm cooperative particle swarm optimizer (MCPSO) is proposed as a new fuzzy modeling strategy for identification and control of non-linear dynamical systems. In MCPSO, the population consists of one master swarm and several slave swarms. The slave swarms execute particle swarm optimization (PSO) or its variants independently to maintain the diversity of particles, while the particles in the master swarm enhance themselves based on their own knowledge and also the knowledge of the particles in the slave swarms. With four benchmark functions, MCPSO is proved to have better performance than PSO and its variants. MCPSO is then used to automatically design the fuzzy identifier and fuzzy controller for non-linear dynamical systems. The proposed algorithm (MCPSO) is shown to outperform PSO and some other methods in identifying and controlling dynamical systems.

© 2007 Elsevier B.V. All rights reserved.

Keywords: Dynamical systems; Fuzzy controller; Fuzzy identifier; Particle swarm optimization; Multi-swarm

1. Introduction

The identification and control of non-linear dynamical systems has been a challenging problem in the control area for a long time. Since for a dynamic system, the output is a non-linear function of past output or past input or both, and the exact order of the dynamical systems is often unavailable, the identification and control of this system is much more difficult than that has been done in a static system. To deal with this problem, many soft computing methods such as neural networks [27,28,33,14,13], fuzzy neural networks [24,18,19] and fuzzy inference systems [43] have been developed to process dynamic systems.

Recently, interest in using recurrent networks has become a popular approach for the identification and control of temporal problems. Many types of recurrent networks have been proposed, among which two widely used categories are recurrent neural networks (RNN) [33,5,11] and recurrent fuzzy networks (RFNN) [24,17].

On the other hand, fuzzy inference systems have been developed to provide successful results in identifying and controlling non-linear dynamical systems [43,45].

Two most common fuzzy inference systems are Mamdani fuzzy model [25] and T–S fuzzy model [41]. In Mamdani fuzzy model, the antecedent and consequent parts of the fuzzy rule are fuzzy propositions, whereas in the T–S fuzzy model, the consequent part is a crisp linear function of the antecedent variables. It was pointed out by [39] that the T–S fuzzy models are capable of approximating highly complex systems by means of a small number of rules when compared against the Mamdani-type fuzzy models.

The bottleneck of the construction of a T–S model is the identification of the antecedent membership functions, which is a non-linear optimization problem. Typically, both the premise parameters and the consequent parameters of T–S fuzzy model are adjusted by using gradient descent optimization techniques [41,42]. Those methods are
sensitive to the choice of the initial parameters, easily got stuck in local minima, and have poor generalization properties. This hampers the a posteriori interpretation of the optimized T–S model.

The advent of evolutionary algorithm (EA) has attracted considerable interest in the construction of fuzzy systems [20,46,15], the most well known of which is genetic algorithms (GAs). In [20,46], GAs have been applied to learn both the antecedent and consequent part of fuzzy rules, and models with both fixed and varying number of rules have been considered. In [40] GAs have been used with fuzzy modeling for identification of the structure of a fuzzy model and selection of input variables. As compared to traditional gradient-based computation system, GAs provide a more robust and efficient approach for the construction of fuzzy systems.

Recently, a new evolutionary computation technique, the particle swarm optimization (PSO) algorithm, was introduced by Kennedy and Eberhart [8,21], and has already come to be widely used in many areas [49,16,26]. Compared to GA, PSO has some attractive advantages: (1) it has a fast convergence rate (2) it is easy to implement (3) it has few parameters to adjust.

However, the studies by Angeline [2] showed that the original PSO (or SPSO) had difficulties in controlling the balance between exploration (global investigation of the search place) and exploitation (the fine search around a local optimum). The SPSO, while quickly converging towards an optimum in the first period of iterations, may have problems when it comes to reach a near optimal solution.

Various attempts have been made to improve the performance of basic PSO, which can be classified here as follows:

i. tuning the parameters in the velocity and position update equations of PSO [50,4],
ii. designing different population topologies [23,22,38],
iii. combining PSO with other search techniques [37,48,47],
iv. incorporating bio-inspired mechanisms into the basic PSO [12,29–31].

We present here a new approach to balance the exploration and exploitation in PSO belonging to the third aforementioned family. Our proposed method is referred to as multi-swarm cooperative particle swarm optimizer (MCPSO) inspired by the symbiotic phenomenon in real natural ecosystem. MCPSO is based on a master–slaver model, in which a population consists of one master swarm and several slave swarms. The slave swarms can supply new promising particles (the position giving the best fitness value) to the master swarm as the evolution progresses. In this way, both the consequent parameters and premise parameters are determined simultaneously.

To demonstrate the performance of the proposed algorithm, MCPSO is applied into the identification and control of several typical non-linear dynamical systems, and the performance is also compared with some existing methods.

The rest of the paper is organized as follows. Section 2 gives a review of PSO and a description of the proposed algorithm MCPSO. In Section 3, it will be shown that MCPSO outperforms PSO and its variants on some benchmark test functions. Section 4 describes the T–S model and a detailed design algorithm of fuzzy model by MCPSO. In Section 5, simulation results of one non-linear plant identification problem and two non-linear dynamical system control problems using fuzzy inference systems based on MCPSO are presented. Finally, conclusions are drawn in Section 6.

2. PSO and MCPSO

2.1. Particle swarm optimization

Particle swarm optimization (PSO) is inspired by natural concepts such as fish schooling, bird flocking and human social relations. The basic PSO is a population based optimization tool, where the system is initialized with a population of random solutions and searches for optima by updating generations. In PSO, the potential solutions, called particles, fly in a D-dimension search space with a velocity which is dynamically adjusted according to its own experience and that of its neighbors.

The location and velocity for the ith particle is represented as \( x_i = (x_{i1}, x_{i2}, \ldots, x_{ID}) \) and \( v_i = (v_{i1}, v_{i2}, \ldots, v_{ID}) \), respectively. The best previous position of the ith particle is recorded and represented as \( P_i = (P_{i1}, P_{i2}, \ldots, P_{ID}) \), which is also called pbest. The index of the best particle among all the particles in the population is represented by the symbol \( g \), and \( p_g \) is called gbest. At each
time step $t$, the particles are manipulated according to the following equations:

$$v_i(t + 1) = v_i(t) + R_1 c_1 (p_i - x_i(t)) + R_2 c_2 (p_g - x_i(t)),$$

(1)

$$x_i(t + 1) = x_i(t) + v_i(t),$$

(2)

where $R_1$ and $R_2$ are random values within the interval $[0,1]$, $c_1$ and $c_2$ are acceleration constants. For Eq. (1), the portion of the adjustment to the velocity influenced by the individual’s own $p_{best}$ position is considered as the cognition component, and the portion influenced by $g_{best}$ is the social component.

A drawback of the aforementioned version of PSO is associated with the lack of a mechanism responsible for the control of the magnitude of the velocities, which fosters the danger of swarm explosion and divergence. To solve this problem, [34] later introduced an inertia term $w$ by modifying (1) to become

$$v_i(t + 1) = w \times v_i(t) + R_1 c_1 (p_i - x_i(t)) + R_2 c_2 (p_g - x_i(t)).$$

(3)

They proposed that suitable selection of $w$ will provide a balance between global and local explorations, thus requiring less iteration on average to find a sufficiently optimal solution. As originally developed, $w$ often decreases linearly from about 0.9 to 0.4 during a run. In general, the inertia weight $w$ is set according to the following equation:

$$w = w_{max} - \frac{w_{max} - w_{min}}{iter_{max}} \times iter,$$

(4)

where $iter_{max}$ is the maximum number of iterations, and $iter$ is the current number of iterations. Hereafter, in this paper, this version of PSO is referred to as standard PSO (SPSO).

Aside from LPSO, Eberhart and Shi [10] have also proposed a random inertia weight factor for tracking dynamic systems. In this development, the inertia weight factor is set to change randomly according to the following equation:

$$w = 0.5 + \frac{rand()}{2},$$

(5)

where $rand()$ is a uniformly distributed random number within range $[0,1]$. It is recommended that the acceleration coefficients keep constant at 1.494. In the remainder of this paper, this method is referred to as random weight method (RPSO).

Another important modification of SPSO is the constriction factor approach PSO (CPSO), which was proposed by Clerc and Kenedy [6]. A detailed discussion of the constriction factor is beyond the scope of this paper, but a simplified method of incorporating it is described in Eq. (6), where $K$ is a function of $c_1$ and $c_2$ as reflected in Eq. (7)

$$v_i(t + 1) = K(v_i(t) + R_1 c_1 (p_i - x_i(t)) + R_2 c_2 (p_g - x_i(t))),$$

(6)

where $K$ is called constriction factor, given by

$$K = \frac{2}{-\varphi + \sqrt{\varphi^2 - 4\varphi}},$$

(7)

where $\varphi = c_1 + c_2$, $\varphi > 4$.

2.2. Multi-swarm cooperative particle swarm optimization

The initial inspiration for the PSO was the coordinated movement of swarms of animals in nature, e.g., schools of fish or flocks of birds. It reflects the cooperative relationship among the individuals within a swarm. However, in natural ecosystems, many species have developed cooperative interactions with other species to improve their survival. Such cooperative co-evolution is called symbiosis, firstly coined by German mycologist, Anton de Bary in 1879 [1]. The phenomenon of symbiosis can be found in all forms of life, from simple cells (e.g., eukaryotic organisms resulted probably from the mutualistic interaction between prokaryotes and some cells they infected) through to birds and mammals (e.g., African tick birds obtain a steady food supply by cleaning parasites from the skin of giraffes) [32].

According to the different symbiotic interrelationships, symbiosis can be classified into three main categories: mutualism (both species benefit by the relationship), commensalism (one species benefits while the other species is not affected), and parasitism (one species benefits and the other is harmed) [7]. We found that the commensalism model is suitable to be incorporated in the SPSO. Inspired by this research, a master–slave mode is incorporated into the SPSO, and the multi-swarm (species) cooperative optimizer (MCPSO) is thus developed.

In our approach, a population consists of one master swarm and several slave swarms. The symbiotic relationship between the master swarm and slave swarms can keep a tight balance of exploration and exploitation, which is essential for the success of a given optimization task. The master–slave communication model, as shown in Fig. 1, is used to assign fitness evaluations and maintain algorithm synchronization.

In Fig. 1 each slave swarm executes a single PSO or its variants, including the update of position and velocity, and the creation of a new local population. When all the slave swarms are ready with the new generations, each slave swarm then sends the best local individual to the master swarm. The master swarm selects the best of all received
Table 1: List of variables used in MCPSO

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>(M)</td>
<td>Master swarm</td>
</tr>
<tr>
<td>(S)</td>
<td>Slave swarm</td>
</tr>
<tr>
<td>(R_3)</td>
<td>Random number between 0 and 1</td>
</tr>
<tr>
<td>(c_3)</td>
<td>Acceleration constant</td>
</tr>
<tr>
<td>(p^M)</td>
<td>The best previous particle in master swarm</td>
</tr>
<tr>
<td>(p^S)</td>
<td>The best previous particle in slave swarms</td>
</tr>
<tr>
<td>(G_{best}^M)</td>
<td>Fitness values determined by (p^M)</td>
</tr>
<tr>
<td>(G_{best}^S)</td>
<td>Fitness values determined by (p^S)</td>
</tr>
</tbody>
</table>

Table 2: Pseudocode for the MCPSO algorithm

Algorithm MCPSO

\(\text{Begin}\)
- Initialize all the populations
- Evaluate the fitness value of each particle

\(\text{Repeat}\)
- Do in parallel
  - Node \(i, 1 \leq i \leq K/K\) is the number of slaver swarms
    - End Do in parallel
- Barrier synchronization \(\text{//wait for all processes to finish}\)
  - Select the fittest local individual \(p^S_i\) from the slave swarms
  - Evolve the master swarm
    \(\text{//update the velocity and position using (8) and (9), respectively}\)
    - Evaluate the fitness value of each particle
- Until a terminate-condition is met

\(\text{End}\)

Table 3: Parameters of the benchmark functions

<table>
<thead>
<tr>
<th>Function</th>
<th>(n)</th>
<th>Minimum value</th>
<th>Range of search</th>
<th>Range of initialization</th>
</tr>
</thead>
<tbody>
<tr>
<td>(f_1)</td>
<td>30</td>
<td>0</td>
<td>([-100, 100])</td>
<td>([50, 100])</td>
</tr>
<tr>
<td>(f_2)</td>
<td>30</td>
<td>0</td>
<td>([-100, 100])</td>
<td>([15, 30])</td>
</tr>
<tr>
<td>(f_3)</td>
<td>30</td>
<td>([-5.12, 5.12])</td>
<td>([2.56, 5.12])</td>
<td></td>
</tr>
<tr>
<td>(f_4)</td>
<td>30</td>
<td>([-600, 600])</td>
<td>([300, 600])</td>
<td></td>
</tr>
</tbody>
</table>
starting at 0.9 and ending at 0.4 following Shi and Eberhart [35] was used for the SPSO and MCPSO. For CPSO, the constriction factor \( K = 0.729 \) was adopted.

The number of slave swarms \( S \) was set as three and three versions of PSO were selected as evolution strategies for the slave swarms: standard PSO (SPSO), random inertial weight version of PSO (RPSO), constriction factor version of PSO (CPSO). All the parameters used in the slave swarms were the same as those defined above. For fair comparison, in all cases, the population size of SPSO and MCPSO was set at 80 (all the swarms of MCPSO include the same particles) and a fixed number of maximum generations 1000 were applied to all algorithms. A total of 50 runs for each experimental setting were conducted.

The results of the averaged fitness values over 50 runs based on MCPSO and some PSO variants are shown in Figs. 2–5 and tabulated in Table 4. Generally, it can be seen that the performance of the proposed MCPSO is better than that of other three versions of PSO.

We believe that the better performance observed in those functions for MCPSO is the result of diversity maintained in the multi-swarm evolutionary process. But this indicator is not enough—we need to assess also the robustness of the algorithm, which may be evaluated by the stability of the results. Figs. 6–9 show the multi-swarm co-evolutionary process for all of the functions by MCPSO. It should be mentioned that we present those graphs only from a single test run.

In each case, we found that the slave swarms performed well in initial iterations but failed to make further progress in later iterations. The experiments with the Rosenbrock function illustrate this point clearly. There was a quick decrease in fitness, in initial iterations, for the master swarm and the slave swarms probably when the particles where descending the walls of the function valley. When the plateau was reached, the slave swarms started to produce improvements very slowly as usually happens with
this function while the master swarm kept finding better solutions long after the stagnation of the slave swarms. By looking at the shapes of the curves in the graphs, it is easy to see that each particle in the master swarm can keep track of the previous best position found by the slave swarms, as well as find a better position based on its own knowledge. In fact, since the competition relationship of the slave swarms, the master swarm would not be influenced much when a certain slave swarm got stuck at a local optima. It may be concluded that the results generated by MCPSO is robust.

Therefore, the MCPSO method can be identified as a highly consistent strategy in finding the optimum solution compared with other methods.

4. Fuzzy model based on MCPSO

4.1. T–S fuzzy model systems

In the last decade, the fuzzy inference system suggested by Takagi and Sugeno [41] (T–S fuzzy model) has

<table>
<thead>
<tr>
<th>Table 4</th>
<th>Results for all algorithms on benchmark functions</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>MCPSO</td>
</tr>
<tr>
<td>( f_1 )</td>
<td></td>
</tr>
<tr>
<td>Best</td>
<td>5.9114e−035</td>
</tr>
<tr>
<td>Worst</td>
<td>3.2327e−030</td>
</tr>
<tr>
<td>Mean</td>
<td>1.9008e−031</td>
</tr>
<tr>
<td>Std</td>
<td>7.6249e−031</td>
</tr>
<tr>
<td>( f_2 )</td>
<td></td>
</tr>
<tr>
<td>Best</td>
<td>5.1000e−003</td>
</tr>
<tr>
<td>Worst</td>
<td>8.9301e+000</td>
</tr>
<tr>
<td>Mean</td>
<td>2.5875e+000</td>
</tr>
<tr>
<td>Std</td>
<td>2.0618e+000</td>
</tr>
<tr>
<td>( f_3 )</td>
<td></td>
</tr>
<tr>
<td>Best</td>
<td>2.4705e−009</td>
</tr>
<tr>
<td>Worst</td>
<td>3.9798e+000</td>
</tr>
<tr>
<td>Mean</td>
<td>1.5561e+000</td>
</tr>
<tr>
<td>Std</td>
<td>1.0855e+000</td>
</tr>
<tr>
<td>( f_4 )</td>
<td></td>
</tr>
<tr>
<td>Best</td>
<td>2.2100e−002</td>
</tr>
<tr>
<td>Worst</td>
<td>8.8500e−002</td>
</tr>
<tr>
<td>Mean</td>
<td>5.6500e−002</td>
</tr>
<tr>
<td>Std</td>
<td>1.9500e−002</td>
</tr>
</tbody>
</table>
become a most significant topic in several applications of fuzzy modeling and control. T–S fuzzy model describes a system by a set of local linear input–output relations. Due to its non-linearity and simple structure (T–S) type fuzzy controllers have gained much attention to approach highly non-linear dynamical systems. In this paper, the T–S fuzzy model is employed to represent a non-linear system. A T–S fuzzy system is described by a set of fuzzy IF-THEN rules that represent local linear input–output relations of non-linear systems. The overall system is then an aggregation of all such local linear models. More precisely, a T–S fuzzy system is formulated in the following form:

\[ R^l : \text{if } x_1 \text{ is } A^l_1 \text{ and } \ldots x_n \text{ is } A^l_n, \text{ then} \]

\[ y^l = z_0^l + z_1^l x_1 + \ldots + z_n^l x_n, \quad (14) \]

where \( y^l (1 \leq l \leq r) \) is the output due to rule \( R^l \) and \( z_i^l (1 \leq i \leq n), \) called the consequent parameters, are the coefficients of the linear relation in the \( l \text{th} \) rule and will be identified. \( A^l_i (x_i) \) are the fuzzy variables defined as the following Gaussian membership function:

\[ A^l_i (x_i) = \exp \left[ -\frac{1}{2} \left( \frac{x_i - m^l_i}{\sigma^l_i} \right)^2 \right], \quad (15) \]

where \( 1 \leq l \leq r, \ldots, 1 \leq i \leq n, x_i \in R, \) \( m^l_i \) and \( \sigma^l_i \) represent the center (or mean) and the width (or standard deviation) of the Gaussian membership function, respectively. \( m^l_i \) and \( \sigma^l_i \) are adjustable parameters called the premise parameters, which will be identified.

Given an input \( (x_1^0 (k), \ldots, x_n^0 (k)) \), the final output \( y (k) \) of the fuzzy system is inferred as follows:

\[ y (k) = \sum_{i=1}^{n} \sum_{l=1}^{r} \tilde{A}^l_i (x_i^0 (k)) \prod_{i=1}^{n} A^l_i (x_i^0 (k)) = \sum_{i=1}^{n} \sum_{l=1}^{r} \tilde{A}^l_i (x_i^0 (k)), \quad (16) \]

where the weight strength \( w^l (k) \) of the \( l \text{th} \) rule, is calculated by

\[ w^l (k) = \prod_{i=1}^{n} A^l_i (x_i^0 (k)). \quad (17) \]

### 4.2. Fuzzy model strategy based on MCPSO

The detailed design algorithm of fuzzy model by MCPSO is introduced in this section. The overall learning process can be described as follows:

(1) Parameter representation

In our work, the parameter matrix, which consists of the premise parameters and the consequent parameters described in Section 3.1, is defined as a two-dimensional matrix, i.e.,

\[
\begin{bmatrix}
    m_1^1 & \sigma_1^1 & \cdots & m_n^1 & \sigma_n^1 & x_0^1 & x_1^1 & \cdots & x_n^1 \\
    m_1^2 & \sigma_1^2 & \cdots & m_n^2 & \sigma_n^2 & x_0^2 & x_1^2 & \cdots & x_n^2 \\
    \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\
    m_1^r & \sigma_1^r & \cdots & m_n^r & \sigma_n^r & x_0^r & x_1^r & \cdots & x_n^r
\end{bmatrix}.
\]

The size of the matrix can be represented by \( D = r \times (3n + 1) \).

(2) Parameter learning

(a) In MCPSO, the master swarm and the slave swarm both works with the same parameter settings except for the velocity update equation. Initially, \( N \times n (N \geq 2, n \geq 2) \) individuals forming the population should be randomly generated and the individuals can be divided into \( N \) swarms (one master swarm and \( N-1 \) slave swarms). Each swarm contains \( n \) individuals with random positions and velocities on \( D \) dimensions. These individuals may be regarded as particles in terms of PSO. In T–S fuzzy model system, the number of rules, \( r \), should be assigned in advance. In addition, the maximum iterations \( w_{\text{max}} \), minimum inertia weight \( w_{\text{min}} \) and the learning parameters \( c_1, c_2 \), the migration coefficient \( c_3 \) should be assigned in advance. After initialization, new individuals on the next generation are created by the following step.

(b) For each particle, evaluate the desired optimization fitness function in \( D \) variables. The fitness function is defined as the reciprocal of \( \text{RMSE} \) (root mean square error), which is used to evaluate various individuals within a population of potential solutions. Considering the single output case for clarity, our goal is to minimize the error function:

\[ \text{RMSE} = \sqrt{\frac{1}{K} \sum_{k=1}^{K} (y_r (k + 1) - y_s (k + 1))^2}, \quad (18) \]

where \( K \) is the total time steps, \( y_r (k + 1) \) is the inferred output and \( y_s (k + 1) \) is the desired reference output.

(c) Evaluate the fitness for each particle.

(d) Compare the evaluated fitness value of each particle with its \( p_{\text{best}} \). If current value is better than \( p_{\text{best}} \), then set the current location as the \( p_{\text{best}} \) location in \( D \)-dimension space. Furthermore, if current value is better than \( g_{\text{best}} \), then reset \( g_{\text{best}} \) to the current index in particle array. This step will be executed in parallel for both the master swarm and the slave swarms.

(e) In each generation, after step (d) is executed, the best-performing particle \( p_{\text{gbest}} \) among the slave swarms should be marked.

(f) Update the velocity and position of all the particles in \( N-1 \) slave swarms according to Eqs. (3) and (2), respectively (Suppose that \( N-1 \) populations of
SPSPO with the same parameter setting are involved in MCPSO as the slave swarms).

(g) Update the velocity and position of all the particles in the master swarm according to Eqs. (5) and (6), respectively.

(3) Termination condition

The computations are repeated until the premise parameters and consequent parameters are converged. It should be noted that after the operation in master swarm and slaver swarm the values of the individual may exceed its reasonable range. Assume that the domain of the ith input variable has been found to be [min(xi), max(xi)] from training data, then the domains of ml and r1 are defined as [min(xi) – δi, max(xi) + δi] and [di – δi, di + δi], respectively, where δi is a small positive value defined as δi = (max(xi) – min(xi))/10, and di is the predefined width of the Gaussian membership function, the value is set as (max(xi) – min(xi))/r, the variable r is the number of fuzzy rules.

5. Illustrative examples

In this section, three non-linear dynamic applications, including an example of identification of a dynamic system and two examples of control of a dynamic system are conducted to validate the capability of the fuzzy inference systems based on MCPSO to handle the temporal relationship. The main reason for using these dynamic systems is that they are known to be stable in the bounded input bounded output (BIBO) sense.

5.1. Dynamic system identification

The systems to be identified are dynamic systems whose outputs are functions of past inputs and past outputs as well. For this dynamic system identification, a serial-parallel model is adopted as identification configuration shown in Fig. 10.

Example 1. The plant to be identified in this example is guided by the difference equation [28,24]:

\[ y_p(k + 1) = f[y_p(k), y_p(k - 1), y_p(k - 2), u(k), u(k - 1)], \]

where

\[ f[x_1, x_2, x_3, x_4, x_5] = \frac{x_1 x_2 x_3 x_4 (x_3 - 1) + x_4}{1 + x_1^2 + x_2^2}. \]  \hspace{1cm} (20)

Here, the current output of the plant depends on three previous outputs and two previous inputs. Unlike the authors in [27] who applied a feedforward neural network with five input nodes for feeding appropriate past values of \(y_p(k)\) and \(u(k)\), we only use the current input \(u(k)\) and the output \(y_p(k)\) as the inputs to identify the output of the plant \(y_p(k + 1)\). In training the fuzzy model using MCPSO for the non-linear plant, we use only 10 epochs and there are 900 time steps in each epoch. Similar to the inputs used in [28,33]. The input is an independent and identically distributed \((iid)\) uniform sequence over \([-2,2]\) for about half of the 900 time steps and a single sinusoid given by \(1.05 \ast \sin(\pi k/45)\) for the remaining time steps. In applying MCPSO to this plant, the number of swarms, the population size of each swarm and evolution strategies for the slave swarms are same as that defined in Section 3. The number of the fuzzy rules is set to be 4. For master swarm, inertial weights \(w_{\text{max}}, w_{\text{min}}, w_{\text{max}}, w_{\text{min}}, c_1, c_2\) and the migration coefficient \(c_3\), are set to 0.35, 0.1, 1.5, 1.5 and 0.8, respectively. In slave swarms the inertial weights and the acceleration constants are the same as those used in master swarm. To show the superiority of MCPSO, the fuzzy identifiers designed by PSO are also applied to the same identification problem. In PSO, the population size is set as 80 and initial individuals are the same as those used in MCPSO. For fair comparison, the other parameters, \(w_{\text{max}}, w_{\text{min}}, c_1, c_2\) are the same as those defined in MCPSO. To see the identified result, the following input as used in [33,24] is adopted for test:

\[ u(k) = \begin{cases} \sin(\pi k/25), & k < 250 \\ 1.0, & 250 \leq k < 500 \\ -1.0, & 500 \leq k < 750 \\ 0.3 \sin(\pi k/25) + 0.1 \sin(\pi k/32) + 0.6 \sin(\pi k/10), & 750 \leq k < 1000. \end{cases} \]  \hspace{1cm} (21)

Fig. 11 shows the desired output (denoted as a solid curve) and the inferred output obtained by using MCPSO.
(denoted as a dotted curve) for the testing input signal. Table 5 gives the detailed identification results using different methods, where the results of the methods RSONFIN, RFNN and TRFN-S come from literature [18]. From the comparisons, we see that the fuzzy controller designed by MCPSO is superior to the method using RSONFIN, and is slightly inferior to the methods using RFNN and TRFN-S. However, among the three types of methods, it achieves the highest identification accuracy in the test part, which demonstrates its better generalized ability. The results of MCPSO identifier also demonstrate the improved performance compared to the results of the identifiers obtained by PSO. The abnormal phenomenon that the test RMSE is smaller than the train RMSE using fuzzy identifier based on PSO and MCPSO may attribute to the well-regulated input data in test part (during time steps [250]; and [500]; the input data is equal to a constant).

5.1. Dynamic system control

As compare to linear systems, for which there now exists considerable theory regarding adaptive control, very little is known concerning adaptive control of plants governed by non-linear equations. It is in the control of such systems that we are primarily interested in this section. Based on MCPSO, the fuzzy controller is designed for the control of dynamical systems. The control configuration and input–output variables of MCPSO fuzzy controller are shown in Fig. 12, and are applied to one MISO (multi-input–single-output) plant control problem and one MIMO (multi-input–multi-output) in the following examples. The comparisons with other control methods are also presented.

Example 2. The controlled plant is the same as that used in [28] and [18] and is given by

\[
y_p(k+1) = \frac{y_p(k)y_p(k-1)(y_p(k) + 2.5)}{1 + y_p^2(k) + y_p^2(k-1)} + u(k).
\]  

(22)

In designing the fuzzy controller using MCPSO, the desired output \( y_r \) is specified by the following 250 pieces of data:

\[
y_r(k + 1) = 0.6y_r(k) + 0.2y_r(k - 1) + r(k), 1 \leq k \leq 250,
\]

\[
r(k) = 0.5 \sin(2\pi k/45) + 0.2 \sin(2\pi k/15)
\]

\[+ 0.2 \sin(2\pi k/90).\]

The inputs to MCPSO fuzzy controller are \( y_p(k) \) and \( y_r(k) \) and the output is \( u(k) \). There are five fuzzy rules in MCPSO fuzzy controller, i.e., \( r = 5 \), resulting in a total of 35 free parameters. Other parameters in applying MCPSO are the same as those used in Example 1. The fuzzy controller designed by PSO is also applied to the MISO control problem and the parameters in using PSO are also the same as those defined in Example 1. The evolution is processed for 100 generations and is repeated for 50 runs. The averaged best-so-far RMSE value over 50 runs for each generation is shown in Fig. 13. From the figure, we
can see that MCPSO converges with a higher speed compared to PSO and obtains a better result.

The best and averaged RMSE error for the 50 runs after 100 generations of training for each run are listed in Table 6, where the results of the methods GA and HGAPSO are from [24]. It should be noted that the TRFN controller designed by HGPSO (or GA) is evolved for 100 generations and repeated for 100 runs in literature [24]. To test the performance of the designed fuzzy controller, another reference input \( r(k) \) is given by

\[
\begin{align*}
  r(k) &= 0.3 \sin(2\pi k/50) + 0.2 \sin(2\pi k/25) \\
  &+ 0.4 \sin(2\pi k/60), \quad 251 \leq k \leq 500.
\end{align*}
\]

The best and averaged control performance for the test signal over 50 runs is also listed in Table 6. From the comparison results, we can see that the fuzzy controller based on MCPSO outperforms those based on GA and PSO greatly especially in the test results, and reaches the same control level with the TRFN controller based on HGPSO.

To demonstrate control performance using the MCPSO fuzzy controller for the MISO control problem, one control performance of MCPSO is shown in Fig. 14 for both training and test control reference output.

### Example 3 (MIMO control)

The MIMO plant to be controlled is the same as described in [27,19]:

\[
\begin{align*}
  y_{p1}(k+1) &= 0.5 \left( y_{p1}(k) + \frac{y_{p1}(k)}{1 + y_{p2}^2(k)} \right) \\
  y_{p2}(k+1) &= 0.5 \left( y_{p2}(k) + \frac{y_{p1}(k)}{1 + y_{p2}^2(k)} \right) \\
  y_{c1}(k) &= 0.5 \left( u_1(k-1) \right) \\
  y_{c2}(k) &= 0.5 \left( u_2(k-1) \right).
\end{align*}
\]

The controlled outputs should follow desired outputs \( y_{r1} \) and \( y_{r2} \) as specified by the following 250 pieces of data:

\[
\begin{align*}
  y_{r1}(k) &= \sin(k\pi/45), \\
  y_{r2}(k) &= \cos(k\pi/45), \quad 1 \leq k \leq 250.
\end{align*}
\]

The inputs to EPPSO controller are \( y_{r1}(k), y_{r2}(k), y_{p1}(k), \) and \( y_{p2}(k) \), and the outputs are \( u_1(k) \) and \( u_2(k) \). There are four fuzzy rules in MCPSO fuzzy controller for MIMO control, resulting in 72 free parameters totally. To show the superiority of MCPSO, the fuzzy controllers designed by PSO is also applied to the MIMO control problem.
The RMSE defined for MIMO control performance index is
\[
RMSE = \left( \frac{1}{K} \sum_{k=1}^{K} \left( (y_{r1}(k+1) - y_{p1}(k+1))^2 + (y_{r2}(k+1) - y_{p2}(k+1))^2 \right) / K \right)^{0.5}
\] (24)
where \( K \) is the total time steps, \( y_{r1}(k+1) \) and \( y_{r2}(k+1) \) are the desired output and \( y_{p1}(k+1), y_{p2}(k+1) \) are the inferred output.

The evolutions are processed for 100 generations and repeated for 25 runs. The averaged best-so-far RMSE value over 25 runs for each generation is shown in Fig. 15. To demonstrate the control result, one control performance is shown in Fig. 16.

Table 7 gives the control performance for the MIMO control problem based on different methods. From the table, we can see that the fuzzy controller based on MCPSO outperforms all other methods including TRFN controller using HGAPSO [24], which is evolved for 100 generations and repeated for 100 runs. The shortcoming of using HGAPSO in designing TRFN is that there are too many free parameters (156) to be evolved, and the binary representation in using GA has brought much complexity to the evolution process which greatly influences the control performance.

6. Conclusions

Inspired by the phenomenon of symbiosis in natural ecosystems, a multi-swarm cooperative particle swarm optimizer has been proposed to improve the performance of SPSO. MCPSO is a master–slave model that consists of one master swarm and several slave swarms. The evolution of slave swarms is likely to amplify the diversity of individuals of the population and consequently to generate more promising particles for the master swarm. The master swarm updates the particle states based on both its own experience and that of the most successful particles in the slave swarms.

In contrast to ordinary PSO, where there is no information sharing among individuals except that global best broadcasts the information to the other individuals, MCPSO is able to transferred the information among the individuals from the slave swarms to master swarms. The interactions between the master swarm and the slave swarms influence the balance between exploration and exploitation and maintain some diversity in the whole population, even when it is approaching convergence, thus reducing the risk of convergence to local sub-optima.

MCPSO is proved to be a powerful global optimization algorithm by comparing to three versions of PSO on four benchmark functions. MCPSO is then employed to the design of an identifier and controller for processing non-linear dynamic systems. The new learning approach for the determination of the consequent parameters and premise
parameters of a T–S type fuzzy model is formulated and explained.

In the simulation part, we apply the suggested method to, respectively, design a fuzzy identifier for a non-linear dynamic plant identification problem and a fuzzy controller for a non-linear dynamic plant control problem. To demonstrate the effectiveness of the proposed algorithm MCPSO, its performance is compared to several typical methods in dynamical systems.

Experimental results indicate that our proposed method has much higher convergence speed and stability than that of some existing methods, and makes the T–S fuzzy model design process easier and more automatic.

Future work will focus on further improving the MCPSO by the proper selection of parameters and the design of different topologies. In addition, MCPSO applications in the structure and parameters learning of systems of higher complexity will be investigated.

Acknowledgements

This work is supported by the National Natural Science Foundation of China (No.70431003) and the National Basic Research Program of China (No.2002CB312200). The first author would like to thank Ms. Jingru Zhang and Dr. Jun Zhang for reviewing the manuscript and giving many valuable comments. We also acknowledge the detailed and thoughtful reviews by two anonymous reviewers.

References


Ben Niu received his B.Sc. degree in Mechanical Engineering from the Hefei Union University, Hefei, China, in 2001. In the same year, he enrolled at the Anhui Agriculture University in Hefei where he earned his M.Sc. degree in Enterprise Information Management in 2004. He is currently pursuing his Ph.D. in the Shenyang Institute of Automation of the Chinese Academy of Sciences. His current research interests include Swarm Intelligence, Bioinformatics and Computational Biology, Fuzzy Systems and Neural Networks, with an emphasis on evolutionary and other stochastic optimization methods.

Yunlong Zhu is the director of the Key Lab. of Advanced Manufacturing Technology, Shenyang Institute of Automation of the Chinese Academy of Sciences. He received his Ph.D. in 2005 from the Chinese Academy of Sciences, China. He has research interests in various aspects of Enterprise Information Management but he has ongoing interests in Artificial Intelligence, Machine Learning, and related areas. Prof Zhu’s research has led to a dozen professional publications in these areas.

Xiaoxian He obtained his B.Sc. degree in Computer Science from the Department of Computer Science and Technology, Hunan Normal University, China, in July 2002. Currently he is a Ph.D. candidate in the Shenyang Institute of Automation, Chinese Academy of Sciences. His research interests include Swarm Intelligence, Complex Adaptive System, Bio-inspired computing and Decision Making.

Hai Shen earned her B.Sc. and her M.Sc. in computer application technology in 1998 and 2005, respectively, both from the Shenyang University of Technology in Shenyang, Liaoning, China. From 1998–2002, she worked as a teacher in the Shenyang Normal University. She is currently pursuing her Ph.D. in the Shenyang Institute of Automation, Chinese Academy of Sciences. Her research interests include Swarm Intelligence, Evolutionary Computation, Bio-computing and Multi-objective Optimization.