Knowledge-Based Systems 44 (2013) 78–89

Contents lists available at SciVerse ScienceDirect

Knowledge-Based Systems

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

Differential evolution with local information for neuro-fuzzy systems optimisation

Ming-Feng Han *, Chin-Teng Lin, Jyh-Yeong Chang

Institute of Electrical Control Engineering, National Chiao Tung University, 1001 University Road, Hsinchu 300, Taiwan, ROC

ARTICLE INFO

Article history:
Received 3 October 2012
Received in revised form 21 December 2012
Accepted 23 January 2013
Available online 4 February 2013

Keywords:
Neuro-fuzzy systems (NFSs)
Differential evolution (DE)
Neuro-fuzzy systems optimisation
Evolutionary algorithm (EA)
Optimisation

ABSTRACT

This paper proposes a differential evolution with local information (DELI) algorithm for Takagi–Sugeno–Kang-type (TSK-type) neuro-fuzzy systems (NFSs) optimisation. The DELI algorithm uses a modified mutation operation that considers a neighbourhood relationship for each individual to maintain the diversity of the population and to increase the search capability. This paper also proposes an adaptive fuzzy c-means method for determining the number of rules and for identifying suitable initial parameters for the rules. Initially, there are no rules in the NFS model; the rules are automatically generated by the fuzzy measure and the fuzzy c-means method. Until the firing strengths of all of the training patterns satisfy a pre-specified threshold, the process of rule generation is terminated. Subsequently, the DELI algorithm optimises all of the free parameters for NFSs design. To enhance the performance of the DELI algorithm, an adaptive parameter tuning based on the 1/5th rule is used for the tuning scale factor *F*. The 1/5th rule dynamically adjusts the tuning scale factor in each period to enhance the search capability of the DELI algorithm. Finally, the proposed NFS with DELI model (NFS-DELI) is applied to nonlinear control and prediction problems. The results of this paper demonstrate the effectiveness of the proposed NFS-DELI model.

1. Introduction

The popular research topic of neuro-fuzzy systems (NFSs) has been successfully applied to many areas [1–10]. Such systems bring both low-level learning and computational power of neural networks into fuzzy systems and the high-level, human-like thinking and reasoning of fuzzy systems to neural networks. To train the parameters for designing an NFS, many papers had employed the backpropagation (BP) algorithm [1,8–10]. The BP algorithm is a powerful training technique that quickly minimises the error function of the NFS. However, the BP algorithm may become trapped at a local optimal solution and never find the global optimal solution. To overcome this disadvantage, many researchers have proposed NFS design that uses evolutionary algorithms (EA) [11–23].

The genetic algorithm (GA) is one of the most well-known EAs. Many researchers have developed GAs to implement fuzzy systems and neuro-fuzzy systems to automate the determination of parameters and structures [14–22].The genetic fuzzy system [8–17] was characterised using a fuzzy system as an individual in a genetic operator. Russo [18] applied a GA to the design of fuzzy controller membership functions, in which each fuzzy rule was treated as an individual. Ng and Li [19] applied chromosomes in a GA to optimise the sophisticated membership functions for a nonlinear water-level control system. Seng [20] proposed a neuro-fuzzy network that was based on the radial basis function neural network in which all of the parameters are tuned simultaneously using a GA. Juang [22] successfully applied a GA to TSK-type recurrent neuro-fuzzy system design for control problems.

Another category of EAs for NFSs design called particle swarm optimisation (PSO) appears to provide efficient and powerful search capability in the search space; this evolutionary computation technique was developed by Kennedy and Eberhart in 1995 [42]. The underlying motivation for the development of the PSO algorithm is the social behaviour of animals, such as bird flocking, fish schooling and swarming. The PSO algorithm has been successfully applied to many optimisation problems, such as NFSs design [23–26] for control problems, and has shown improved performance over GAs.

Hybrid EAs have been investigated in many studies [27–30]. Such hybrids are frequently combinations of local searches and EA s, also known memetic algorithms [29,30]. A hybrid of a cultural method and a cooperative PSO (CPSO) was applied in the design of a functional link-based neural fuzzy network (FLNFN) [27]; this method was referred to as the FLNFN–CCPSO algorithm. In the FLNFN–CCPSO algorithm, a swarm optimises only one parameter of an FLNFN. Another hybrid EA was the combination of a GA and a PSO, which is called the HGAPSO algorithm [28]. In the HGAPSO algorithm, new individuals are created by a combination of a PSO and the crossover and mutation operations of a GA.

* Corresponding author. Tel.: +886 3 5712121.
E-mail addresses: ming0901@gmail.com (M.-F. Han), ctlin@mail.nctu.edu.tw (C.-T. Lin), jychang@mail.nctu.edu.tw (J.-Y. Chang).

0950-7051/$ - see front matter Crown Copyright © 2013 Published by Elsevier B.V. All rights reserved.
http://dx.doi.org/10.1016/j.knosys.2013.01.023

Moreover, the differential evolution (DE) algorithm, proposed by Storn and Price [31,32], which is an efficient global optimiser in the continuous search domain. The DE algorithm employs the difference of two randomly selected individuals as the source of random variations for third individual and applies to difficult optimisation problems [31–39,52–54]. Comparison with GA, the DE algorithm is faster and easier EA. Further research [40] has proposed a modified DE (MODE) algorithm for an adaptive neural fuzzy network (ANFN-MODE) design. This MODE algorithm provides a cluster-based mutation scheme for preventing the algorithm from becoming trapped in the local optima of the search space. In addition, the MODE algorithm has been applied to locally recurrent NFSs design [41]. However, the MODE algorithm lacks a self-adaptive parameter tuning strategy for obtaining more reliable convergence performance [34,39].

This paper proposes a DELI algorithm for designing a TSK-type NFS. Initially, there are no rules in the NFS model; the fuzzy rules are automatically generated by an adaptive fuzzy c-means method. Subsequently, all free parameters are learned by the proposed DELI algorithm. In the DELI algorithm, a self-adaptive parameter tuning strategy based on the 1/5th rule is used to adjust the tuning scale factor F effectively. In the simulation, the proposed NFS with DELI model (NFS-DELI) is applied to nonlinear control and prediction problems. The contributions of this paper are summarized as follows:

(1) An adaptive fuzzy c-means method is proposed to determine the number of rules for the NFS and identify suitable initial parameters for an individual in the DELI algorithm.
(2) The proposed DELI algorithm adopts a modified mutation operation in evaluating the global best individual, the local best individual, and randomly chosen individuals for maintaining the diversity of population and increasing the search capability.
(3) We also propose a self-adaptive parameter tuning strategy based on the 1/5th rule to effectively adjust the tuning scale factor F in the DELI algorithm and balance the exploitation ability and the exploration ability.
(4) Three simulations are conducted to evaluate the performance of the NFS-DELI algorithm. Comparisons with other EAs demonstrate the superiority of the performance of the NFS-DELI model.
(5) In the simulation, we use non-parametric Wilcoxon signed-rank test to verify the statistical significance. Results indicate that the DELI algorithm and the other algorithms show a statistically significant difference.

This paper is organised as follows: Section 2 describes the basic operation of the NFS and DE. Section 3 introduces the rule generation and parameter optimisation algorithms used with the DELI algorithm. Section 4 presents the NFS-DELI design and compares its performance with other EAs. Finally, Section 5 draws conclusions.

2. Neuro-fuzzy systems and DE algorithm

This section describes the architecture of the TSK-type NFSs to be designed in this paper. The basic evolution process of the DE algorithm, including mutation operation, crossover operation and selection operation, is also described in this section.

2.1. Neuro-fuzzy systems

The NFSs realises a first-order TSK-type fuzzy IF–THEN rule, i.e., the ith rule, (Rule i), which is instantiated as:

$$\begin{aligned} &Rulej: \text{IF } x_1 \text{ is } A_{1j} \text{ and } x_2 \text{ is } A_{2j} \text{ and } \ldots x_n \text{ is } A_{nj} \\ &\text{THEN } y_j = w_{0j} + w_{1j}x_1 + \ldots + w_{nj}x_n, \end{aligned} \tag{1}$$

where $x_1, \ldots, x_n$ are the input variables, y_i is the system output variable, $A_{1i}, \ldots, A_{ni}$ are the linguistic terms of the pre-condition part with a Gaussian membership function, n is the number of input variables, and $w_{0j}, \ldots, w_{nj}$ are the TSK weights.

The architecture of four-layered NFS is shown in Fig. 1, which is comprised of the input, membership function, rule, and output layers. The operation functions of the nodes in each layer of the NFS are described as follows, where $O^{(l)}$ denotes the output of a node in the lth layer:

2.1.1. Layer 1—input layer

No computation is performed in this layer. Each node in this layer corresponds to one input variable and only transmits input values directly to the next layer. That is,

$$O^{(1)} = x_i, \tag{2}$$

where $i = 1, 2, \ldots, n$ are the input variables of the NFS.

2.1.2. Layer 2—membership function layer

Each node in this layer is a membership function that corresponds to the linguistic label of one of the input variables in layer 1. In other words, the membership value specifies the degree to which an input value belongs to a fuzzy set and is calculated in layer 2 according to the expression:

$$O^{(2)} = \mu_{ij} = \exp\left(-(x_i - m_{ij})^2/\sigma_{ij}^2\right), \tag{3}$$

where $j = 1, 2, \ldots, M$, M is the number of rules in the NFS, and m_{ij} and σ_{ij} are the centre (mean) and the width (variance) of the Gaussian membership function of the input variable, respectively.

2.1.3. Layer 3—Rule layer

This layer receives one-dimensional membership degrees of the associated rule from a set of nodes in layer 2. Here, the product operator is adopted to perform the precondition part of the fuzzy rules. As a result, the output function of each inference node is

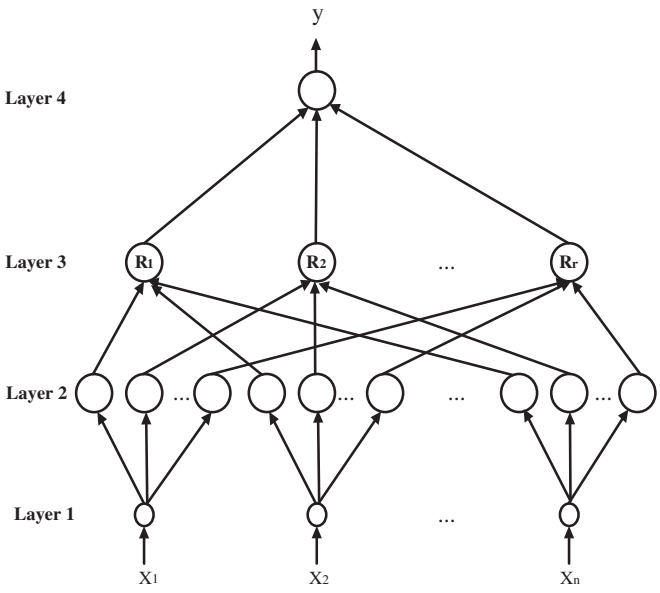

Fig. 1. The architecture of the NFS.

$$O^{(3)} = R_j = \prod_{i=1}^{n} \mu_{ij}. \tag{4}$$

The output of a layer 3 node represents the firing strength of the corresponding fuzzy rule.

2.1.4. Layer4—output layer

Each node in this layer corresponds to one output variable. The node integrates all of the actions recommended by layer 3 and the consequent part and acts as a defuzzifier with

$$O^{(4)} = y = \frac{\sum_{j=1}^{M} R_j \left(\sum_{i=1}^{l} w_{ij} x_i + w_{0j} \right)}{\sum_{j=1}^{M} R_j}. \tag{5}$$

In this paper, the consequent part uses a TSK-type rule, which is a linear combination of the TSK parameters w_{ij} and the input variables.

2.2. DE algorithm

Initially, a population of NP D-dimensional parameter vectors, which represent the candidate solutions (individuals), is generated by a uniformly random process. A simple representation of the ith individual at the current generation, gen, is shown as $\boldsymbol{Z}_{i,gen} = (z_{i,1,gen}, z_{i,2,gen}, z_{i,3,gen}, \ldots, z_{i,D-1,gen}, z_{i,D,gen})$. After the first NP individuals are produced, the fitness evaluation process measures the quality of the individuals to calculate the individual performance. The succeeding steps, including mutation operation, crossover operation and selection operation, are described in the following paragraphs.

In the mutation operation, each individual in the current generation is allowed to breed through mating with other randomly selected individuals from the population. The process randomly selects a parent pool of three individuals to produce an offspring. For each individual $\boldsymbol{Z}_{i,gen}$, $i = 1, 2, \ldots, NP$, where gen denotes the current generation and NP is population size, three other random individuals, $\boldsymbol{Z}_{r1,gen}$, $\boldsymbol{Z}_{r2,gen}$ and $\boldsymbol{Z}_{r3,gen}$ are selected from the population such that $r1$, $r2$ and $r3 \in \{1, 2, \ldots, NP\}$ and $i \neq r1 \neq r2 \neq r3$. In this way, a parent pool of four individuals is formed to produce an offspring. The different mutation strategies are frequently used as follows:

$$\mathrm{DE/rand/1} : \mathbf{V}_{i,gen} = \boldsymbol{Z}_{r1,gen} + F(\boldsymbol{Z}_{r2,gen} - \boldsymbol{Z}_{r3,gen}) \tag{6}$$

$$\mathrm{DE/best/1} : \mathbf{V}_{i,gen} = \boldsymbol{Z}_{gbest,gen} + F(\boldsymbol{Z}_{r1,gen} - \boldsymbol{Z}_{r2,gen}) \tag{7}$$

$$\mathrm{DE/target} - to - best : \mathbf{V}_{i,gen} = \boldsymbol{Z}_{r1,gen} + F(\boldsymbol{Z}_{gbest,gen} - \boldsymbol{Z}_{r1,gen}) + F(\boldsymbol{Z}_{r2,gen} - \boldsymbol{Z}_{r3,gen}) \tag{8}$$

where F is the scaling factor $\epsilon[0, 1]$ and $\boldsymbol{Z}_{gbest,gen}$ is the best-so-far individual (i.e., $\boldsymbol{Z}_{gbest,gen}$ stores the best fitness value in the population up to the current time).

After the mutation operation, the DE algorithm uses a crossover operation, often referred to as discrete recombination, in which the mutated individual $\mathbf{V}_{i,gen}$ is mated with $\boldsymbol{Z}_{i,gen}$ and generates the offspring $\mathbf{U}_{i,gen}$. The elements of an individual $\mathbf{U}_{i,gen}$ are inherited from $\boldsymbol{Z}_{i,gen}$ and $\mathbf{V}_{i,gen}$, which are determined by the parameter crossover probability ($\mathrm{CR} \in [0, 1]$), as follows:

$$U_{d,i,gen} = \begin{cases} \boldsymbol{V}_{d,i,gen}, \text{if } \mathrm{rand}(d) \leqslant \mathrm{CR} \\ \boldsymbol{Z}_{d,i,gen}, \text{if } \mathrm{rand}(d) > \mathrm{CR} \end{cases} \tag{9}$$

where $d = 1, 2, \ldots, D$ denotes the dth element of individual vectors, D is total number of elements of the individual vector and $\mathrm{rand}(d) \in [0, 1]$ is the dth evaluation of a random number generator.

In the selection operation, the DE algorithm applies the selection operation to determine whether the individual survives to the next generation. A knockout competition occurs between each individual $\boldsymbol{Z}_{i,gen}$ and its offspring $\mathbf{U}_{i,gen}$, and the winner, selected deterministically based on objective function values, is then promoted to the next generation. The selection operation is described as follows:

$$\boldsymbol{Z}_{i,gen+1} = \begin{cases} \boldsymbol{Z}_{i,gen}, \text{if fitness}(\boldsymbol{Z}_{i,gen}) < \text{fitness}(\boldsymbol{U}_{i,gen}) \\ \boldsymbol{U}_{i,gen}, \text{otherwise} \end{cases} \tag{10}$$

where $f(\boldsymbol{Z})$ is the fitness value of individual z. After the selection operation, the population gets better or retains the same fitness value, but never deteriorates. All steps are repeated until the evolution process reaches the terminal condition.

3. NFS-deli design

This section describes the structure learning and parameter learning for the design of NFSs. An AFCM method for structure learning determines the number of rules automatically. Following rule generation, all of the free parameters in the NFSs are learned by the DELI algorithm for parameter learning. Finally, a self-adaptive parameter tuning strategy adjusts the tuning factor F for the DELI algorithm during parameter learning.

3.1. Rule generation using the adaptive fuzzy c-means algorithm

The first step is to determine the number of clusters in the universal of discourse of each input variable. One cluster in the input space corresponds to one potential fuzzy logic rule according to the mean (m_{ij}) and width (σ_{ij}) of that cluster. Many previous studies have employed a clustering technique, such as fuzzy c-means [43], possibilistic C-means [44], and linear vector quantisation [45] for rule generation. However, these clustering techniques require prior knowledge, such as the number of clusters present in a pattern set. To solve this problem, an AFCM algorithm was proposed for rule generation. This algorithm used the rule firing strength as a criterion for determining whether to add a new rule. Based on this concept, previous studies [6,12,12–14,41] have used the same idea to generate a new fuzzy rule by assigning preliminary or random values to the centre and width of Gaussian fuzzy set. However, this assignment strategy likely has no benefit for optimal solution search. Therefore, the AFCM algorithm employed the FCM methods and statistical theory to locate effective starting values for the centre and the width of the fuzzy sets. A detailed flowchart of rule generation based on the AFCM algorithm is shown in Fig. 2. The NFS-DELI algorithm initially has no rules. This algorithm performs FCM with one cluster to obtain the first fuzzy rule. The centre and width of this Gaussian fuzzy set $\boldsymbol{A}_{ij}$ was assigned as:

$$m_{i1} = \sum_{k=1}^{N} \frac{x_i(k)}{N}, \quad \text{for } i = 1, \ldots, n \tag{11}$$

$$\sigma_{i1} = \sum_{k=1}^{N} \frac{(x_i(k) - m_{i1})^2}{N}, \quad \text{for } i = 1, \ldots, n \tag{12}$$

Next, the criterion for rule generation is defined by

$$I = \min_i \max_j R_{ij}(\vec{x}_i), \qquad 1 < i < N, \quad 1 < j < M \tag{13}$$

where R_{ij} is the firing strength of the jth fuzzy rule with an ith dimensional input, N is the total of input patterns, and M is the total of fuzzy rules. If $I \leqslant R_{th}$, where $R_{th} \in [0, 1]$ is a pre-specified threshold, then the FCM method with $M + 1$ clusters is performed to generate a new fuzzy rule. The centre and width of Gaussian fuzzy sets are defined as follows:

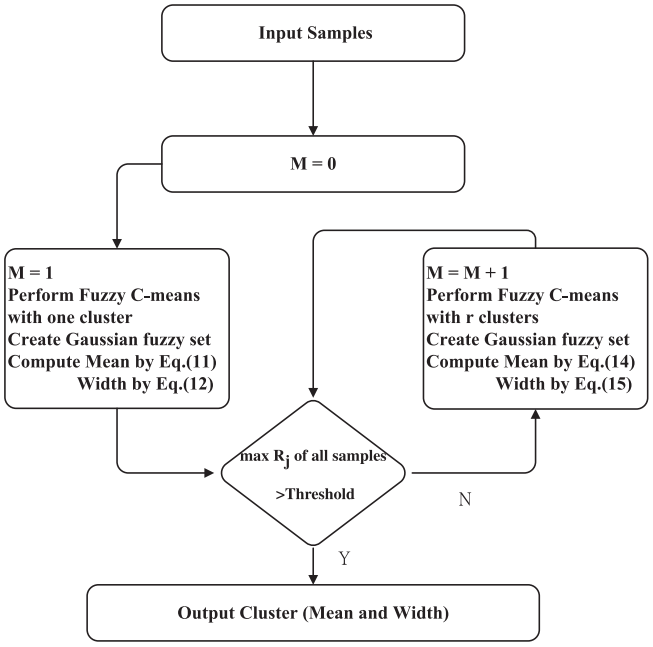

Fig. 2. A flow chart representing the AFCM method for rule generation.

$$m_{ij} = \sum_{k \in j\text{th cluster}} \frac{x_i(k)}{N_j}, \quad \text{for } i = 1, \ldots, N_j \text{ and } j = 1, \ldots, M \tag{14}$$

$$\sigma_{ij} = \sum_{k \in j\text{th cluster}} \frac{(x_i(k) - m_{ij})^2}{N_j}, \quad \text{for } i = 1, \ldots, N_j \text{ and } j = 1, \ldots, M \tag{15}$$

where N_j is the total of patterns in the jth cluster. The above process for rule generation is repeated until the criterion satisfies the prespecified threshold. In the AFCM algorithm, the threshold R_{th} is an important parameter. The threshold is set to between zero and one. A low threshold leads to fewer rules in the NFS, whereas a high threshold leads to more rules. As a result of our extensive experiments and by carefully examining the threshold value, which uses the range $[0, 1]$, we concludes that the threshold is defined as 0.01–0.001 times of the number of input variables.

3.2. The DELI algorithm for parameter learning

To effectively learn parameters in EAs, a trade off between exploration and exploitation is necessary. However, the traditional DE algorithm favours exploitation when determining the global best solution. All of the individuals in an optimal solution search based on the best position may have contributed to convergence towards the same solution [34,39]. To deal with this drawback, we consider local information as neighbourhood relationship of each individual in the DELI algorithm. This idea employs the local best solution in the local region of each individual to increase the diversity of the population and prevent the population from getting trapped in a local minimum solution.

The DELI learning algorithm for NFSs optimisation consists of six major steps: the coding step, the population step, the evaluation step, the mutation step, the crossover step, and the selection step. Fig. 3 shows a flow chart of DELI, and the entire learning process is described as follows:

(1) *The coding step*: The foremost step in the DELI algorithm is the coding of the NFS into an individual. Fig. 4 shows an example of the coding of parameters of the NFS into an individual, where NP is the population size, r is the number of rules, and n is the total number of input variable. In this paper, a Gaussian membership function is used with variables that represent the mean and width of the membership function. Fig. 4 represents the NFS that consists of the mean m_{ij} and width σ_{ij} of a Gaussian membership function with the weight of the consequent part w_{kj}, where i and j represent the ith input variable and the jth rule, respectively.

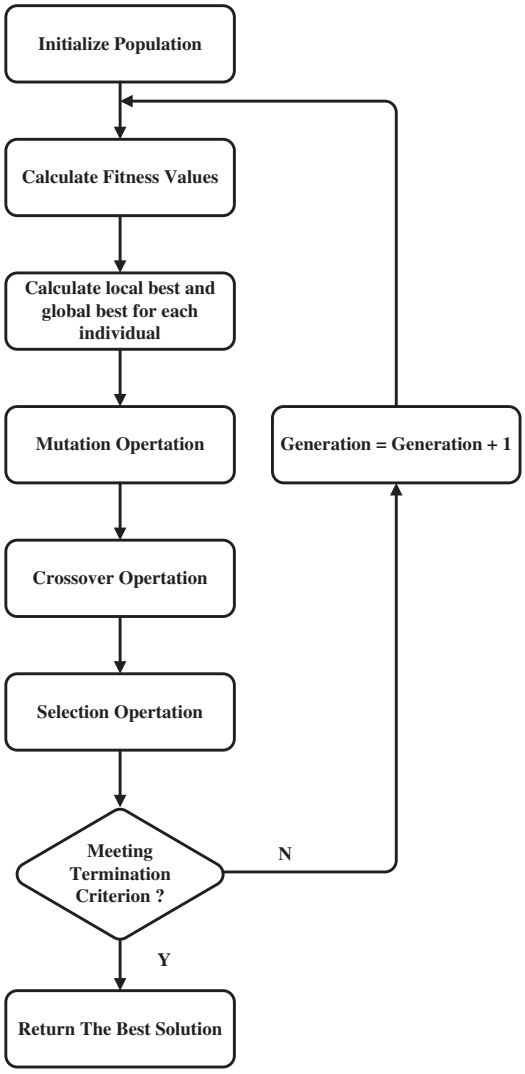

Fig. 3. A flow chart representing the DELI algorithm.

(2) *The population step*: Before the DELI method can be designed, the individuals that constitute the initial population must be created. A niching operation [13,14,41] creates a good initial population in the input space for NFSs optimisation. The initial population is created according to the range of the mean and variance of the membership function, which are computed by the AFCM algorithm in section III-A. The following formulations represent the generation of the initial population:

$$\begin{aligned} NFS_q &= [rule_1^q | rule_2^q | \ldots | rule_r^q] \\ &= [m_{i1}^* + \Delta m_{i1}^q, \sigma_{i1}^* + \Delta\sigma_{i1}^q, w_{k1}^q | \cdots | m_{ij}^* + \Delta m_{ij}^q, \sigma_{ij}^* \\ &\quad + \Delta\sigma_{ij}^q, w_{kj}^q | \cdots | m_{ir}^* + \Delta m_{ir}^q, \sigma_{ir}^* + \Delta\sigma_{ir}^q, w_{kr}^q] \end{aligned} \tag{16}$$

where m_{ij}^* and σ_{ij}^* are the results of AFCM algorithm for the mean and width of the Gaussian membership function of

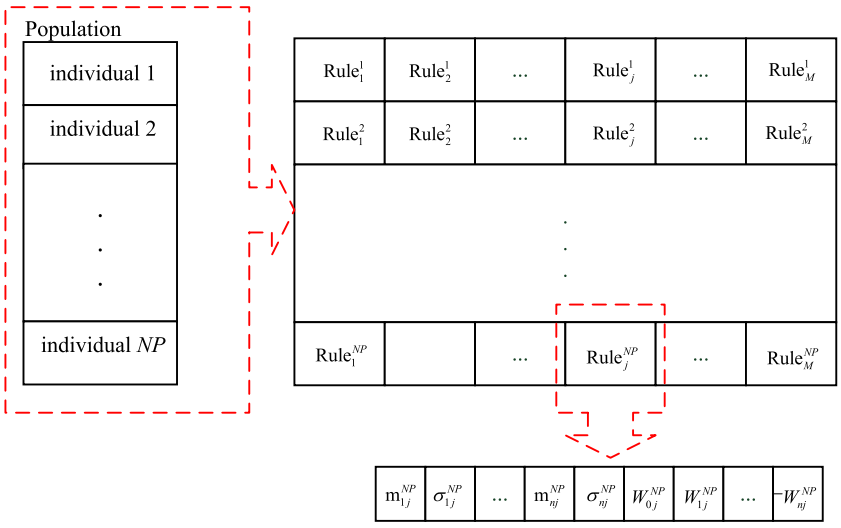

Fig. 4. A schematic showing encoding of the NFSs into individuals and the population for the DELI algorithm.

the jth rule of the ith input variable, Δm_{ij}^q and $\Delta\sigma_{ij}^q$ are small random deviations that are uniformly generated from the interval $[-0.1, 0.1]$, and w_{kj} are randomly and uniformly generated from an interval with a range identical to the NFS output y range. The size of the population depends on the complexity of the problem. Many experiments have shown that a population size of 50 is the most effective [27,40,41]. In addition to the number of fuzzy systems to be formed and evaluated in each generation, the crossover rate, and the mutation rate and these parameters must be set and also depend on the complexity of the problem.

(3) *The evaluation step*: In this paper, we adopted a fitness function (i.e., objective function) to evaluate the performance of each individual. The fitness function used in this paper is the root mean-squared error (RMSE) between the desired output and the actual output. In NFSs control, for example, the performance of a NFS is the RMSE between its controlled plant-output trajectory and the desired trajectory. The fitness function is defined as follows:

$$fitness = \sqrt{\frac{\sum_{k=1}^{N}(y_k - \bar{y}_k)^2}{N}}, \tag{17}$$

where y_k represents the model output of the kth pattern, $\bar{y}_k$ represents the desired output of the kth pattern, and N represents the number of the training pattern.

(4) *The mutation step*: Each individual in the current generation is allowed to breed through mating with other randomly selected individuals from the population. Specifically, for each individual $\boldsymbol{Z}_{i,gen}$, $i = 1, 2, ..., NP$, where gen denotes the current generation, and NP denotes the population size, four other random individuals, $\boldsymbol{Z}_{r1,gen}$, $\boldsymbol{Z}_{r2,gen}$, $\boldsymbol{Z}_{r3,gen}$, and $\boldsymbol{Z}_{r4,gen}$ are selected from the population such that $r1$, $r2$, $r3$,and $r4 \in \{1, 2, \ldots, NP\}$and $i \neq r1 \neq r2 \neq r3 \neq r4$. In this way, a parent pool of four individuals is formed to produce an offspring. A mutation operation with local information applies a differential operation to generate a mutated individual $\boldsymbol{V}_{i,gen}$ according to the following equation:

$$\begin{aligned}\boldsymbol{V}_{i,gen} &= \alpha(V_{i,gen}^{local}) + (1-\alpha)(\boldsymbol{V}_{i,gen}^{global}) \\ &= \alpha(\boldsymbol{Z}_{i,gen} + F(\boldsymbol{Z}_{lbest,i,gen} - \boldsymbol{Z}_{i,gen}) + (1-F)(\boldsymbol{Z}_{r1,gen} \\ &\quad - \boldsymbol{Z}_{r2,gen})) + (1-\alpha)(\boldsymbol{Z}_{i,gen} + F(\boldsymbol{Z}_{gbest,gen} - \boldsymbol{Z}_{i,gen}) \\ &\quad + (1-F)(\boldsymbol{Z}_{r3,gen} - \boldsymbol{Z}_{r4,gen})),\end{aligned} \tag{18}$$

where α and F are the scaling factors $\epsilon[0, 1]$, $\boldsymbol{Z}_{gbest,gen}$ is the best-so-far individual (i.e., $\boldsymbol{Z}_{gbest,gen}$ maintains the best fitness value of the current individual in the population), and $\boldsymbol{Z}_{lbest,i,gen}$ is local best individual in the neighbourhood of individual $\boldsymbol{Z}_{i,gen}$.

A convex relationship is set as F and $(1 - F)$ in Eq. (18). This idea is recommended by [40,41]. Their papers use convex relationship to achieve a better performance for NFSs optimisation in the evolutionary algorithm. The same idea is also applied in the scaling factor α and $(1 - \alpha)$. In addition, this paper employs a new and flexible mutation operation in DELI for NFSs optimisation. Unlike the mutation operation conventionally used with the DE algorithm, which only utilises global information $\boldsymbol{V}_{i,gen}^{global}$, the new mutation operation in the DELI algorithm considers both global information $\boldsymbol{V}_{i,gen}^{global}$ and local information $\boldsymbol{V}_{i,gen}^{local}$. It is worth noting that conventional DE is a special case of the DELI algorithm, when the scaling factor $\alpha = 0$. Thus, the new mutation operation in the DELI algorithm has more variety in offspring production than the conventional mutation operation.

The concept of local information is schematically illustrated in Fig. 5. To determine the neighbourhood relationship of an individual, we use a neighbourhood topology based on the Euclidean distance over the individual space. To set a suitable neighbourhood size, we performed additional experiments using neighbourhood sizes of 5%, 10%, 15%, 20%, and 25% of NP individuals near individual $\boldsymbol{Z}_{i,gen}$ to evaluate the local information for finding local best individual. Finally, we chose 10% of NP individuals as the neighbourhood size because it obtained the best performance in this paper.

(5) *The crossover step and the selection step*: After the mutation operation, the DELI algorithm uses a crossover operation and selection operation to generate the offspring. In this paper, we adopt the traditional crossover and selection operations from the DE algorithm. The operations are described in Section 2.2.

3.3. Self-adaptive parameter tuning strategy based on 1/5th rule

An important task within an EA is controlling parameters that can directly influence the convergence speed and search capability, such as the scaling factors F and α. However, a conventional DE algorithm uses a trial-and-error method for choosing a suitable scaling factor F and requires multiple optimisation runs. In this section, we propose a self-adaptive approach based on the 1/5th rule

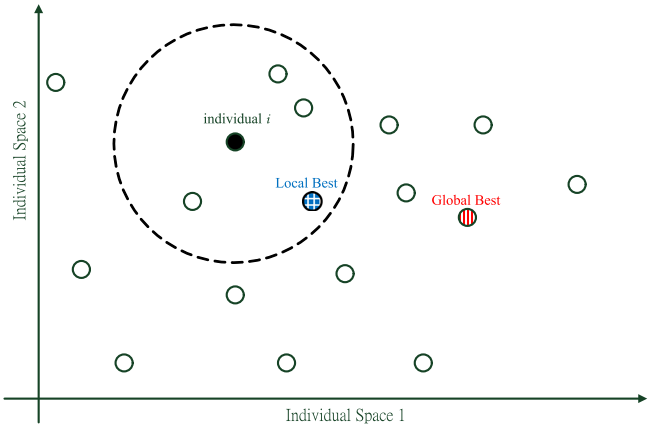

Fig. 5. Local information in the DELI algorithm.

Initialization: assign initial scaling factor F, adjustment factor $\beta \in [0,1]$ and period G_p as number of generation.

Algorithm:

1) Perform DELI algorithm.
2) Every G_p generation,
   - Calculate the number N_s of updating global best solution.
   - Determine an estimate of success probability P as

$$P = \frac{N_s}{G_p} \tag{20}$$

   - Change scaling factor F according to

$$F = \begin{cases} F/\beta & \text{if } P > 1/5 \\ F \cdot \beta & \text{if } P < 1/5 \\ F & \text{if } P = 1/5 \end{cases} \tag{21}$$

Fig. 6. The 1/5th rule method for parameter tuning in the DELI algorithm.

for automatic parameter tuning. The 1/5th rule [46,47] is a well-known method for parameter tuning and is typically used in evolution strategies for controlling the mutation strength. The idea of the 1/5th rule is to trade off the exploitation ability and the exploration ability based on a success probability P. If the success probability P is greater than 1/5, the algorithm increases the exploration ability. If the success probability P is less than 1/5, the algorithm increases the exploitation ability. In the case that P equals 1/5, the same ability is retained by the algorithm. Based on this concept, the scaling factor F is adapted in a similar way by the 1/5th rule in the DELI algorithm. A larger scaling factor F means that the DELI algorithm favours the exploitation ability, while a smaller scaling factor F means that the DELI algorithm favours the exploration ability. Thus, the 1/5th rule is a proper parameter tuning strategy for the DELI algorithm. A complete self-adaptive parameter tuning strategy based on the 1/5th rule is shown in Fig. 6.

The scaling factor α is also adjusted by parameter tuning. The significance of the scaling factor α is to favour global information, local information or both for offspring based on mutation operation. In the interesting case when the scaling factor $\alpha = 0$, the mutation operation degraded to conventional DE mutation operation as follows:

$$\boldsymbol{V}_{i,gen} = \boldsymbol{Z}_{i,gen} + F(\boldsymbol{Z}_{gbest,gen} - \boldsymbol{Z}_{i,gen}) + (1-F)(\boldsymbol{Z}_{r3,gen} - \boldsymbol{Z}_{r4,gen}). \tag{19}$$

This mutation operation favours only global information for evolving a better individual. A similar situation occurs with the case of $\alpha = 1$, when the DELI algorithm performs the new mutation operation that favours only local information. Other cases use both global information and local information in the mutation operation. To increase stochastic search capability and to diversify the population for the DELI algorithm, the scaling factor α is set to vary as a uniformly distributed random number $[0,1]$ in every period G_p.

For setting the period G_p and adjustment factor β, we follow suggestions from journal papers [46,47]. They recommend $0.85 \leqslant \beta < 1$ and $1 \leqslant G_p \leqslant D$, where D is the dimensions of $\boldsymbol{Z}_{i,gen}$. In this paper, we set reasonable values as $\beta = 0.9$ and $G_p = 20$ for our simulations.

4. Simulation

This section discusses three simulations to evaluate the NFS model with DELI method. The first case involves chaotic time series prediction, the second case involves nonlinear plant-tracking control, and the third case involves auto-MPG6 prediction [51]. Table 1 presents the initial parameters prior to training used in each of the three simulations.

For comparison, the evolutionary algorithms, such as GA, PSO, DE and MODE, are applied to the same problems for NFSs optimisation. We use the same population size and number of generations in each of these evolutionary algorithms. The AFCM method is also used for rule generation.

In the following simulations, the major computation time is evaluating the performance of the NFSs. All evolutionary algorithms are compared using the same population size and number of generations in a single run. Thus, the overall computation time is almost the same for different evolutionary algorithms. In our simulation, the number of evaluations is $2000 \times 50 = 100{,}000$ for each run.

4.1. Case 1: chaotic series prediction

The time-series prediction problem used in this case is the chaotic Mackey–Glass time series, which is generated from the following differential equation:

$$\frac{dx(t)}{dt} = \frac{0.2x(t-\tau)}{1+x^{10}(t-\tau)} - 0.1x(t) \tag{22}$$

where $\tau > 17$. As in previous studies [6,12], the parameter $\tau = 30$, and $x(0) = 1.2$ in this simulation. Four past values are used to predict $x(t)$, and the input–output pattern format is given by $[x(t-24), x(t-18), x(t-12), x(t-6)|x(t)]$.

A total of 1000 patterns are generated from $t = 124$ to 1123, where the first 500 patterns [from $x(1)$ to $x(500)$] are used to train, and the last 500 patterns [form $x(501)$ to $x(1000)$] are used to test. A total of 50 runs are performed for statistical analysis. In the DELI algorithm, the threshold R_{th} is set to 0.002.

Table 2 shows the average and standard deviation (STD) of the number of rules over 50 runs, which is denoted by the average ± STD. Fig. 7 shows the learning curves of the GA, PSO, DE, MODE and DELI algorithms in case 1. The learning curve of the GA algorithm presented a tardy convergence result and ultimately

Table 1
Initial parameters before training.

Parameter	Value
Population size	50
Neighbourhood size	5
Crossover rate	0.8
Adjustment factor β	0.9
Initial F	0.5
Period G_p	20
Generation	2000
Coding type	Real number

remained at a training RMSE = 0.081. The PSO and DE presented a rapid convergence result over the first 150 generations that became trapped at local minimum solutions at training RMSE = 0.066 and 0.069, respectively. The MODE and DELI algorithms continually kept convergence results during evolution. It is clear from these data that the proposed DELI method shows better learning curves than the other methods. Table 2 shows that the average performance of the DELI algorithm compared with those of GA, PSO, DE, and MODE over 50 runs. The results show that the DELI algorithm for NFSs optimisation offers a smaller testing RMSE than the other methods. Table 3 shows that the testing RMSE of HGAPSO [28], SPSO [24], MOGUL-TSK [48], CPSO [49] and HPSO-TVAC [25] from other journal papers. In this table, the proposed DELI algorithm achieves better performance than other evolutionary algorithms. Fig. 8 shows the prediction results of the DELI algorithm for the desired output and the actual output. The designed NFS with four fuzzy IF–THEN rules is presented as follows:

Rule 1: IF x_1 is μ (0.15; 0.16) and x_2 is μ (0.97; 0.16) and x_3 is μ (1.03; 0.15) and x_4 is μ (1.91; 0.07) THEN $y = -0.43x_1 + 4.99x_2 + 0.52x_3 + 0.38x_4 + 2.94$.
Rule 2: IF x_1 is μ (0.69; 0.19) and x_2 is μ (0.66; 0.05) and x_3 is μ (1.01; 0.15) and x_4 is μ (0.80; 0.17) THEN $y = 0.15x_1 + 0.55x_2 - 0.18x_3 + 0.75x_4 + 0.03$.
Rule 3: IF x_1 is μ (1.09; 0.03) and x_2 is μ (0.96; 0.03) and x_3 is μ (1.31; 0.20) and x_4 is μ (0.74; 0.07) THEN $y = -0.84x_1 + 0.30x_2 - 0.29x_3 - 4.05x_4 + 4.69$.
Rule 4: IF x_4 is μ (1.25; 0.15) and x_2 is μ (1.02; 0.14) and x_3 is μ (0.85; 0.12) and x_4 is μ (1.12; 0.21) THEN $y = -0.36x_1 - 1.07x_2 + 0.06x_3 + 0.52x_4 + 1.82$.

Where the $\mu(m; \sigma)$is a Gaussian membership function with centre m and the width σ.

To verify the statistical significance of the difference between performance of the DELI algorithm and that of the other algorithms, the non-parametric Wilcoxon signed-rank test [50] is employed. Table 2 shows the result of Wilcoxon signed-rank test. The null hypothesis is rejected at p-values < 0.05, which indicates a statistically significant difference between the DELI algorithm and the other algorithms.

4.2. Case 2: nonlinear plant-tracking control

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

$$y(k+1) = \frac{y(k)}{y^2(k)+1} + u^3(k), \tag{23}$$

where $-2 \leqslant y(k) \leqslant 2$, with $y(0) = 0$, $u(k)$ is the control input, and $u(k) \in [-1, 1]$. As in previous studies [13], the objective is to control the output $y(k)$ to track the following desired trajectory by a NFS:

$$y_d(k) = \sin\left(\frac{\pi k}{50}\right)\cos\left(\frac{\pi k}{30}\right), \quad 1 \leqslant k \leqslant 250. \tag{24}$$

The designed NFS inputs are $y_d(k+1)$ and $y(k)$, and the output is $u(k)$. The error function for performance evaluation is defined to be the RMSE, i.e.,

$$RMSE = \sqrt{\frac{\sum_{k=0}^{249}(y_d(k+1) - y(k+1))^2}{250}}. \tag{25}$$

In the DELI algorithm, the threshold R_{th} is set to 0.005. For statistical analysis, the learning process is repeated for 50 runs for all algorithms. Fig. 9 shows the learning curves of the GA, PSO, DE, MODE and DELI algorithms in case 2. Table 4 shows the performances of the GA, PSO, DE, MODE and DELI algorithms using the same number of rules for NFSs optimisation over 50 runs. The performance of the GA was poorer than other EAs in case2. The PSO and DE algorithms presented similar learning curves and obtained similar performance results (training RMSE = 0.033 and 0.032, respectively). The result of the MODE algorithm remained at a training RMSE = 0.026 after 500 generations, and this result was better than those of the GA, PSO and DE algorithms. The performance of the DELI algorithm obtained a training RMSE = 0.022, which was better than the other algorithms in case 2. In addition, the DELI algorithm is also compared with other evolutionary algorithms in Table 5. These compared algorithms include HGAPSO [28], SPSO [24], CPSO [49], CB-CPSO [24], HPSO-TVAC [25], TPSIA [13] and RACO [13]. The proposed DELI algorithm obtains better performance than other evolutionary algorithms. Fig. 10 shows the control result of the DELI algorithm for the desired output (symbol "+") and the actual output (symbol "O"). The designed NFS are presented by seven TSK-type fuzzy IF–THEN rules as follows:

Rule 1: IF x_1 is μ (0.32; 0.15) and x_2 is μ (0.49; 0.13) THEN $y = 0.003x_1 + 0.69x_2 + 0.17$.
Rule 2: IF x_1 is μ (1.47; 0.36) and x_2 is μ (0.63; 0.29) THEN $y = 0.23x_1 + 0.50x_2 + 0.15$.
Rule 3: IF x_1 is μ (−0.16; 0.07) and x_2 is μ (−0.11; 0.22) THEN $y = -0.71x_1 + 1.05x_2 + -0.05$.
Rule 4: IF x_1 is μ (0.22; 0.12) and x_2 is μ (0.40; 0.12) THEN $y = -0.57x_1 + 0.26x_2 + 0.28$.
Rule 5: IF x_1 is μ (−0.70; 0.19) and x_2 is μ (−0.46; 0.25) THEN $y = 0.22x_1 + 0.53x_2 - 0.14$.
Rule 6: IF x_1 is μ (−0.50; 0.20) and x_2 is μ (−0.73; 0.30) THEN $y = 0.08x_1 + 0.89x_2 + 0.02$.
Rule 7: IF x_1 is μ (−0.09; 0.14) and x_2 is μ (0.07; 0.17) THEN $y = 0.22x_1 + 0.87x_2 - 0.02$.

In Table 4, the p-values are smaller than 0.05 for which the null hypothesis is rejected for statistic comparison using the Wilcoxon signed-rank test. The DELI algorithm performs significantly better than other evolutionary algorithms.

4.3. Case 3: auto-MPG6 prediction

This is a real-world problem that concerns the prediction of automobile city-cycle fuel consumption, in miles per gallon

Table 2
Performance of the DELI algorithm and the other algorithms in case 1.

		GA	PSO	DE	MODE	DELI
Rule number		4 ± 0	4 ± 0	4 ± 0	4 ± 0	4 ± 0
Training RMSE	Mean	0.081	0.066	0.069	0.044	0.016
	STD	0.032	0.018	0.021	0.015	0.005
Testing RMSE	Mean	0.090	0.077	0.072	0.050	0.023
	STD	0.037	0.022	0.020	0.016	0.010
p-Values		1.75×10^{-09}	7.55×10^{-10}	1.02×10^{-09}	2.47×10^{-08}	–

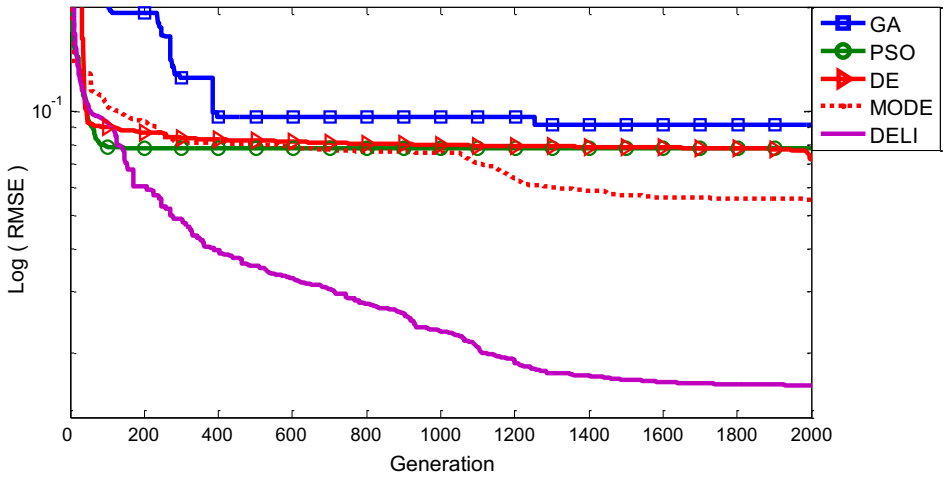

Fig. 7. Training RMSEs at each performance evaluation for the DELI algorithm and other algorithms in case 1.

Table 3
Comparison of the DELI algorithm and other papers for case 1.

Method	Testing RMSE
NFS-DELI	0.023
HGAPSO [28]	0.047
SPSO [24]	0.054
MOGUL-TSK [48]	0.045
CPSO [49]	0.032
HPSO-TVAC [25]	0.047

(MPG). There are five inputs and one output in the prediction model. The real dataset contains 398 examples and can be downloaded from KEEL (http://www.keel.es/) [51]. Evaluation of this model used the five-fold cross-validation datasets in KEEL. The inputs are scaled to the range [0,1]. For each cross-validation dataset, a learning algorithm is repeated for ten runs. For the DELI algorithm, the threshold R_{th} is set to 0.0005. Fig. 11 shows the learning curves of the GA, PSO, DE, MODE and DELI algorithms in case 3. Table 6 shows the performances of the GA, PSO, DE, MODE and DELI algorithms using the same number of rules for NFSs optimisation. When comparing the performances of basic evolutionary algorithm, such as GA, PSO and DE, the result of the DE algorithms was better than that of the GA and PSO algorithms in cases 1, 2 and 3. We compared the performance of our method with advance evolutionary algorithms, and the comparison results were tabulated in Table 6. According to these results, the proposed DELI algorithm outperforms the MODE algorithm. Table 7 shows a comparative result for the DELI algorithm and other papers. The result presents that the proposed DELI algorithm achieves better performance than HGAPSO [28], MOGUL-TSK [48], CPSO [49] and HPSO-TVAC [25].

Fig. 12 shows the training output of the DELI algorithm for the desired output (symbol "+") and the actual output (symbol "O"). Fig. 13 shows the testing result of the DELI algorithm. The TSK-type fuzzy IF–THEN rules of the designed NFS are presented by

Rule 1: IF x_1 is $\mu(0.70;0.31)$ and x_2 is $\mu(0.49;0.19)$ and x_3 is $\mu(0.47;0.19)$ and x_4 is $\mu(1.37;0.38)$ and x_5 is $\mu(0.29;0.21)$ THEN $y = -7.18x_1 - 4.1x_2 - 7.24x_3 - 6.17x_4 + 2.81x_5 + 28.49$.
Rule 2: IF x_1 is $\mu(-0.22;0.17)$ and x_2 is $\mu(0.45;0.08)$ and x_3 is $\mu(0.18;0.14)$ and x_4 is $\mu(-0.19;0.24)$ and x_5 is $\mu(0.58;0.19)$ THEN $y = 14.49x_1 - 2.89x_2 + 49.68x_3 - 13.98x_4 + 14.63x_5 + 4.49$.
Rule 3: IF x_1 is $\mu(1.44;0.13)$ and x_2 is $\mu(-1.49;0.001)$ and x_3 is $\mu(1.48;0.14)$ and x_4 is $\mu(-0.95;0.29)$ and x_5 is $\mu(-1.16;1.71)$ THEN $y = 29.84x_1 - 31.02x_2 - 35.30x_3 - 48.79x_4 + 45.61x_5 - 49.98$.
Rule 4: IF x_1 is $\mu(-0.63;0.52)$ and x_2 is $\mu(0.42;0.14)$ and x_3 is $\mu(1.01;0.38)$ and x_4 is $\mu(1.34;0.36)$ and x_5 is $\mu(1.46;1.32)$ THEN $y = 12.27x_1 - 22.39x_2 - 35.38x_3 + 5.33x_4 + 4.33x_5 + 35.71$.

The p-values in Table 6 indicate that the null hypothesis is rejected, with p-values < 0.05. The DELI algorithm and the other

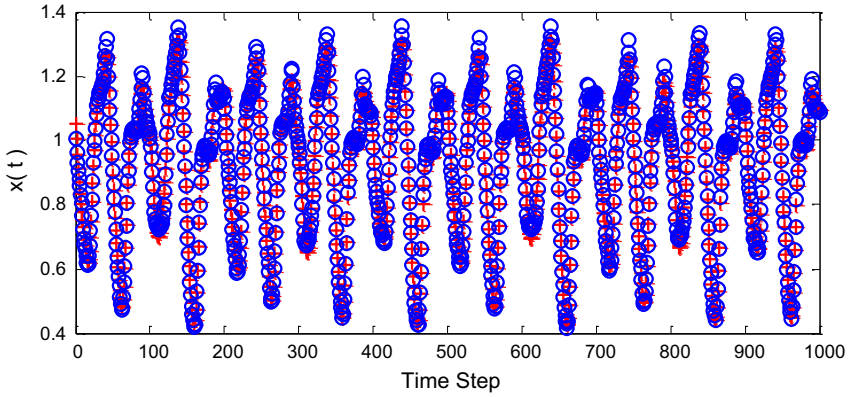

Fig. 8. Symbol "+" represents the desired results and "O" represents the prediction results of the DELI algorithm for NFSs optimisation in case 1.

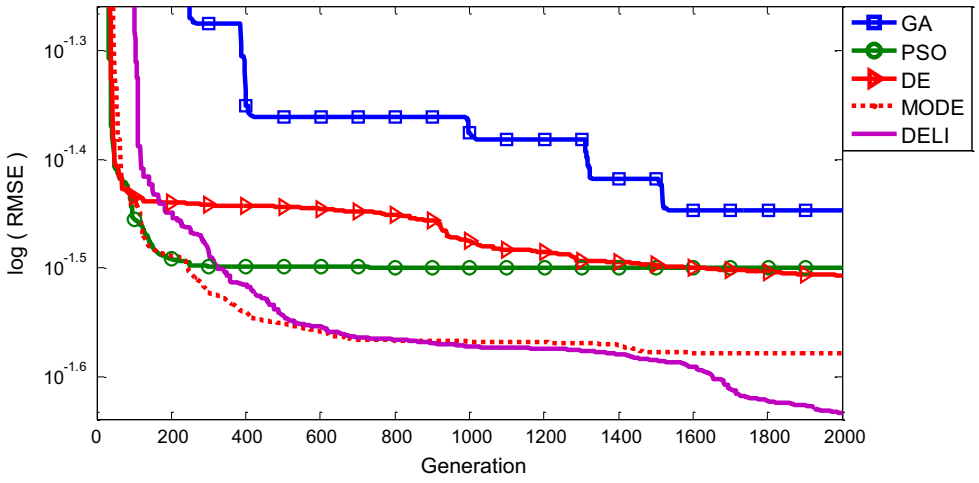

Fig. 9. The training RMSEs for each performance evaluation of the DELI algorithm and the other algorithms in case 2.

Table 4
Performance of the DELI algorithm and the other algorithms in case 2.

		GA	PSO	DE	MODE	DELI
Rule number		7 ± 0	7 ± 0	7 ± 0	7 ± 0	7 ± 0
Training RMSE	Mean	0.039	0.033	0.032	0.026	0.022
	STD	0.009	0.010	0.007	0.009	0.005
p-Values		1.65×10^{-09}	1.38×10^{-09}	2.34×10^{-08}	4.00×10^{-02}	–

Table 5
Comparison of the DELI algorithm and other papers for case 2.

Method	Training RMSE
NFS-DELI	0.022
HGAPSO [28]	0.028
SPSO [24]	0.026
CPSO [49]	0.032
CB-CPSO [24]	0.030
HPSO-TVAC [25]	0.025
TPSIA [13]	0.033
RACO [13]	0.026

algorithms show a statistically significant difference in performance based on Wilcoxon signed-rank test.

5. Conclusion

This study has proposed a DELI algorithm for TSK-type NFSs optimisation. The AFCM method for rule generation in the DELI algorithm helps to determine the number of rules and to locate good initial parameters. All free parameters are learned by the DELI algorithm. The DELI algorithm considers local best information and global best information using a mutation operation for increasing the search capability. In addition, an adaptive parameter tuning strategy based on the 1/5th rule is used to adjust the scale factor

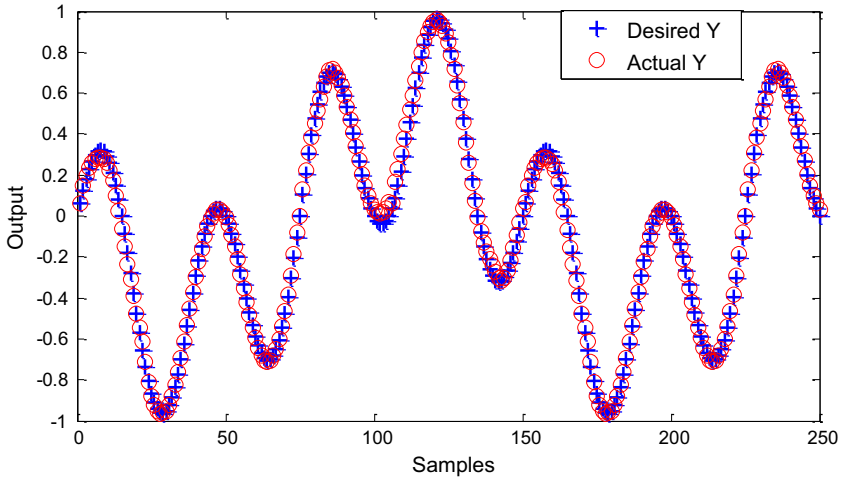

Fig. 10. The control output of the DELI algorithm for NFSs optimisation in case 2.

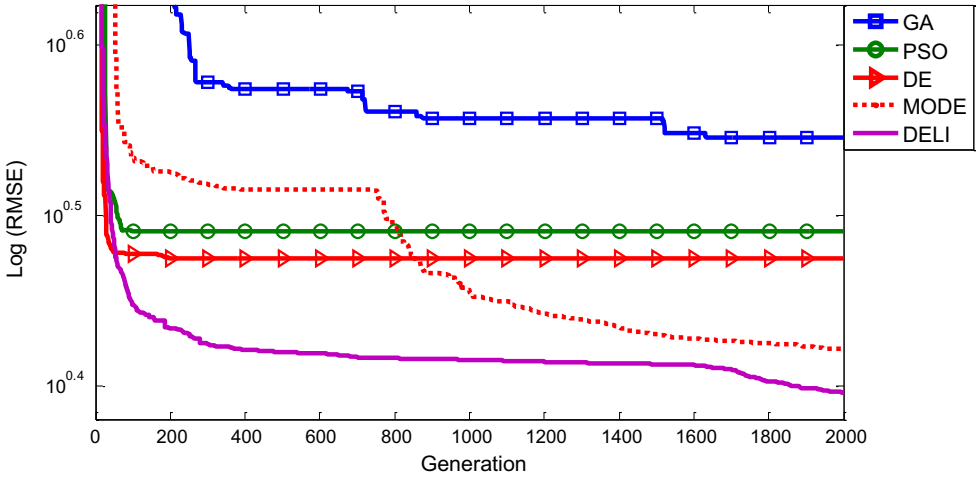

Fig. 11. The training RMSEs for each performance evaluation of the DELI algorithm and the other algorithms in case 3.

Table 6
The performance of the DELI algorithm and other algorithms in case 3.

		GA	PSO	DE	MODE	DELI
Rule number		4 ± 0.8	4 ± 0.8	4 ± 0.8	4 ± 0.8	4 ± 0.8
Training RMSE	Mean	3.89	3.35	3.27	2.51	2.36
	STD	0.51	0.56	0.56	0.22	0.15
Testing RMSE	Mean	4.02	3.66	3.61	2.89	2.58
	STD	0.77	0.68	0.72	0.34	0.21
p-Values		8.53×10^{-19}	5.36×10^{-09}	5.06×10^{-09}	2.97×10^{-05}	–

Table 7
Comparison of the DELI algorithm and other papers for case 3.

Method	Testing RMSE
NFS-DELI	2.58
HGAPSO [28]	2.97
MOGUL-TSK [48]	5.16
CPSO [49]	2.66
HPSO-TVAC [25]	2.72

F in DELI. The experimental results demonstrate that the NFS-DELI method can obtain a smaller RMSE than other evolutionary algorithms for solving prediction and control problems.

Two advanced topics should be addressed in future research on the proposed DELI algorithm. First, the DELI algorithm may adopt other optimisation algorithms to improve the performance. For example, Lin and Chen [27] applied the notion of symbiotic evolution to the PSO algorithm for NFSs design. The basic idea of symbiotic evolution is that an individual is used to represent a single fuzzy rule. A fuzzy system is formed when several individuals, which are randomly selected from a population, are combined. This method increases the number of combination possibilities in

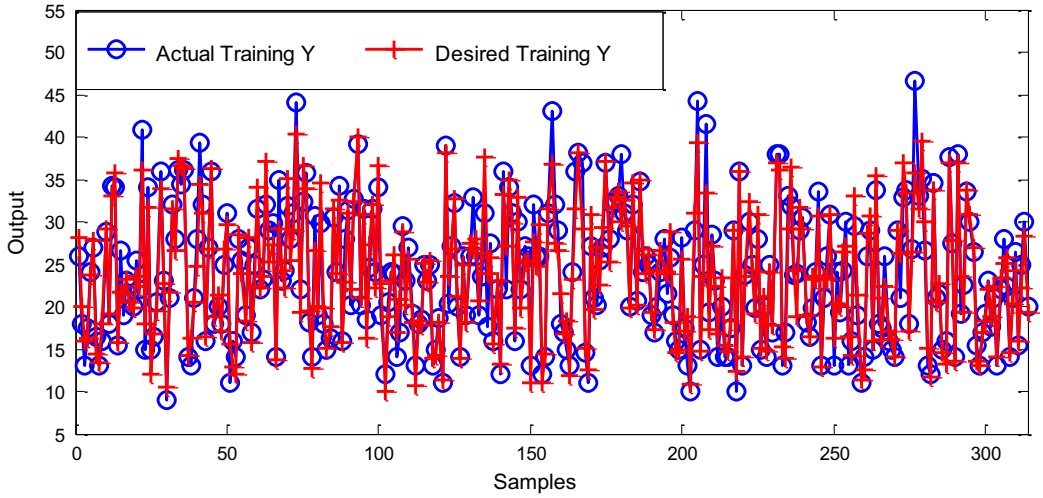

Fig. 12. The training output of the DELI algorithm for NFSs optimisation in case 3.

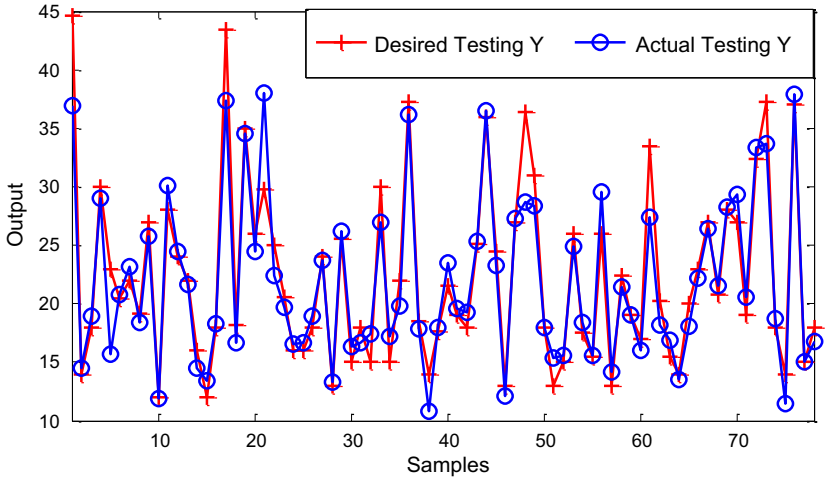

Fig. 13. The testing output of the DELI algorithm for NFSs optimisation in case 3.

evolution process. Second, the advanced crossover and selection operations [11,33,47,54] will be considered in the DELI algorithm. The performance of proposed algorithm can be enhanced by these operations.

Acknowledgement

This work was supported by the UST-UCSD International Center of Excellence in Advanced Bioengineering sponsored by the Taiwan National Science Council I-RiCE Program under Grant Number NSC-101-2911-I-009-101 and by the Aiming for the Top University Plan of National Chiao Tung University, the Ministry of Education, Taiwan, under Contract 101W963.

References

[1] C.-T. Lin, C.S.G. Lee, Neural Fuzzy Systems: A Neuro-Fuzzy Synergism to Intelligent System, Prentice-Hall, Englewood Cliffs, NJ, 1996.
[2] H. Ohtake, K. Tanaka, H.O. Wang, Fuzzy model-based servo and model following control for nonlinear systems, IEEE Trans. Syst., Man, Cybern. B, Cybern. 39 (6) (2009) 1634–1639.
[3] D. Coyle, G. Prasad, T.M. McGinnity, Faster self-organizing fuzzy neural network training and a hyperparameter analysis for a brain–computer interface, IEEE Trans. Syst., Man, Cybern. B, Cybern. 39 (6) (2009) 458–1471.
[4] S.-S. Kim, K.-C. Kwak, Development of quantum-based adaptive neuro-fuzzy networks, IEEE Trans. Syst., Man, Cybern. B, Cybern. 40 (1) (2010) 91–100.
[5] P.R. Vundavilli, M.B. Parappagoudar, S.P. Kodali, S. Benguluri, Fuzzy logic-based expert system for prediction of depth of cut in abrasive water jet machining process, Knowledge-Based Syst. 27 (2012) 456–464.
[6] C.-F. Juang, C. Lo, Zero-order TSK-type fuzzy system learning using a two-phase swarm intelligence, Fuzzy Sets Syst. 159 (21) (2008) 2910–2926.
[7] E. Hadavandi, H. Shavandi, A. Ghanbari, Integration of genetic fuzzy systems and artificial neural networks for stock price forecasting, Knowledge-Based Syst. 23 (8) (2010) 800–808.
[8] J. Liu, W. Wang, F. Golnaraghi, E. Kubica, A neural fuzzy framework for system mapping applications, Knowledge-Based Syst. 23 (6) (2010) 572–579.
[9] Y.-H. Chien, W.-Y. Wang, Y.-G. Leu, T.-T. Lee, Robust adaptive controller design for a class of uncertain nonlinear systems using online T–S fuzzy-neural modeling approach, IEEE Trans. Syst., Man, Cybern. B, Cybern. 41 (2) (2011) 542–552.
[10] S.-B. Roh, S.-K. Oh, W. Pedrycz, A fuzzy ensemble of parallel polynomial neural networks with information granules formed by fuzzy clustering, Knowledge-Based Syst. 23 (3) (2010) 202–219.
[11] S.N. Qasem, S.M. Shamsuddin, A.M. Zain, Multi-objective hybrid evolutionary algorithms for radial basis function neural network design, Knowledge-Based Syst. 27 (2012) 475–495.
[12] M.-F. Han, C.-T. Lin, J.-Y. Chang, A compensatory neurofuzzy system with online constructing and parameter learning, in: Proceedings of 2010 IEEE International Conference on Systems, Man and, Cybernetics, pp. 552–556, October 2010.
[13] C.-F. Juang, P.-H. Chang, Designing fuzzy rule-based systems using continuous ant colony optimization, IEEE Trans. Fuzzy Syst. 18 (1) (2010) 38–149.
[14] C.-F. Juang, Combination of on-line clustering and Q-value based GA for reinforcement fuzzy system design, IEEE Trans. Fuzzy Syst. 13 (3) (2005) 289–302.
[15] F. Hoffmann, D. Schauten, S. Holemann, Incremental evolutionary design of TSK fuzzy controllers, IEEE Trans. Fuzzy Syst. 15 (4) (2007) 563–577.
[16] E. Sanchez, T. Shibata, L.A. Zadeh, Genetic Algorithms and Fuzzy Logic Systems: Soft Computing Perspectives, World Scientific, Singapore, 1997.
[17] O. Cordoon, F. Herrera, F. Hoffmann, L. Magdalena, Genetic fuzzy systems: evolutionary tuning and learning of fuzzy knowledge bases, Advances in Fuzzy Systems—Applications and Theory, World Scientific, Singapore, 2001.
[18] M. Russo, Genetic fuzzy learning, IEEE Trans. Evol. Comput. 4 (3) (2000) 259–273.
[19] K.C. Ng, T. Li, Design of sophisticated fuzzy logic controllers using genetic algorithms, in: Proceedings of 3rd IEEE International Conference on Fuzzy Systems, 1994, pp. 1708–1711.
[20] T.L. Seng, M.B. Khalid, R. Yusof, Tuning of a neuro-fuzzy controller by genetic algorithm, IEEE Trans. Syst., Man, Cybern. B 29 (1999) 226–236.
[21] C.-H. Chou, Genetic algorithm-based optimal fuzzy controller design in the linguistic space, IEEE Trans. Fuzzy Syst. 14 (3) (2006) 372–385.
[22] C.-F. Juang, A TSK-type recurrent fuzzy network for dynamic systems processing by neural network and genetic algorithms, IEEE Trans. Fuzzy Syst. 10 (2) (2002) 155–170.
[23] K.D. Sharma, A. Chatterjee, A. Rakshit, A hybrid approach for design of stable adaptive fuzzy controllers employing Lyapunov theory and particle swarm optimization, IEEE Trans. Fuzzy Syst. 17 (2) (2009) 329–342.
[24] C.-F. Juang, C.M. Hsiao, C.H. Hsu, Hierarchical cluster-based multispecies particle-swarm optimization for fuzzy-system optimization, IEEE Trans. Fuzzy Syst. 18 (1) (2010) 14–26.
[25] A. Ratnaweera, S.K. Halgamuge, H.C. Watson, Self-organizing hierarchical particle swarm optimizer with time-varying acceleration coefficients, IEEE Trans. Evol. Comput. 8 (3) (2004) 240–255.
[26] R.-E. Precup, R.-C. David, E.M. Petriu, M.-B. Rădac, S. Preitl, J. Fodor, Evolutionary optimization-based tuning of low-cost fuzzy controllers for servo systems, Knowledge-Based Syst. 38 (2013) 74–84.
[27] C.-J. Lin, C.-H. Chen, C.-T. Lin, A hybrid of cooperative particle swarm optimization and cultural algorithm for neural fuzzy network and its prediction applications, IEEE Trans. Syst., Man, Cybern. C, Appl. Rev. 39 (1) (2009) 55–68.
[28] C.-F. Juang, A hybrid of genetic algorithm and particle swarm optimization for recurrent network design, IEEE Trans. Syst., Man, Cybern. B, Cybern. 34 (2) (2004) 997–1006.
[29] N. Krasnogor, J. Smith, A memetic algorithm with self-adaptive local search: TSP as a case study, in Proceedings of Genetic and Evolutionary Computation Conference, Las Vegas, NV, July 2000, pp. 987–994.
[30] H. Ishibuchi, T. Yoshida, T. Murata, Balance between genetic algorithm and local search in memetic algorithms for multiobjective permutation flowshop scheduling, IEEE Trans. Evol. Comput. 7 (2003) 204–223.
[31] R. Storn, K. Price, Differential evolution: a simple and efficient heuristic for global optimization over continuous spaces, J. Global Optimiz. (1997).
[32] R. Storn, System design by constraint adaptation and differential evolution, IEEE Trans. Evol. Comput. 3 (1) (1999) 22–34.
[33] R.M. Alguliev, R.M. Aliguliyev, N.R. Isazade, DESAMC + DocSum: Differential evolution with self-adaptive mutation and crossover parameters for multi-document summarization, Knowledge-Based Syst. 36 (2012) 21–38.
[34] S. Das, P.N. Suganthan, Differential evolution: a survey of the state-of-the-art, IEEE Trans. Evol. Comput. 14 (1) (2011) 4–31.
[35] S. Das, A. Abraham, U.K. Chakraborty, A. Konar, Differential evolution using a neighborhood-based mutation operator, IEEE Trans. Evol. Comput. 13 (3) (2009) 526–553.

[36] J. Brest, S. Greiner, B. Boskovic, M. Mernik, V. Zumer, Self-adapting control parameters in differential evolution: a comparative study on numerical benchmark problems, IEEE Trans. Evol. Comput. 10 (6) (2006) 646–659.
[37] F. Al-Obeidat, N. Belacel, J.A. Carretero, P. Mahanti, Differential Evolution for learning the classification method PROAFTN, Knowledge-Based Syst. 23 (5) (2010) 418–426.
[38] L. Wang, C.-X. Dun, W.-J. Bi, Y.-R. Zeng, An effective and efficient differential evolution algorithm for the integrated stochastic joint replenishment and delivery model, Knowledge-Based Syst. 36 (2012) 104–114.
[39] M.-F. Han, C.-T. Lin, J.-Y. Chang, D.-L. Li, Group-based differential evolution for numerical optimization problems, Int. J. Innov. Comput., Inform. Control 9 (3) (2013) 1357–1372.
[40] C. -H Chen, C.-J. Lin, C.-T. Lin, Nonlinear system control using adaptive neural fuzzy networks based on a modified differential evolution, IEEE Trans. Syst. Man Cybern. Part C, Appl. Rev. 39 (4) (2009) 459–473.
[41] C.-T. Lin, M.-F. Han, Differential evolution based optimization of locally recurrent neuro-fuzzy system for dynamic system identification, in: The 17th National Conference on Fuzzy Theory and its Applications, 2010, pp. 702–707.
[42] J. Kennedy, R. Eberhart, Particle swarm optimization, in Proceedings of IEEE International Conference on Neural Network, December, 1995, pp. 1942–1948.
[43] M.-C. Hung, D.-L. Yang, The efficient fuzzy c-means clustering technique, in: Proceedings of IEEE International Conference on Data Mining, December 2001, pp. 225–232.
[44] R. Krishnapuram, J.M. Keller, The possibilistic C-means algorithm: insights and recommendations, IEEE Trans. Fuzzy Syst. 4 (3) (1996).
[45] J. He, L. Liu, G. Palm, Speaker identification using hybrid LVQ-SLP networks, in: Proceedings of IEEE International Conference on Neural Networks, vol. 4, 1995, pp. 2052–2055.
[46] T. Thomas Bäck, H.-P. Schwefel, Evolution strategies I: variants and their computational implementation, Gen. Algor. Eng. Comput. Sci. (1995) 111–126.
[47] H.G. Beyer, H.P. Schwefel, Evolution strategies: a comprehensive introduction, Natural Comput. (2002) 3–52.
[48] R. Alcaíla, J. Alcaíla-Fdez, J. Casillas, O. Cordón, F. Herrera, Local identification of prototypes for genetic learning of accurate TSK fuzzy rule-based systems, Int. J. Intell. Syst. 22 (2007) 909–941.
[49] F. Van Den Bergh, A.P. Engelbrecht, A cooperative approach to particle swarm optimization, IEEE Trans. Evol. Comput. 8 (3) (2004) 225–239.
[50] J. Demsar, Statistical comparisons of classifiers over multiple data sets, J. Mach. Learn. Res. (2006) 1–30.
[51] J. Alcalá-Fdez, A. Fernandez, J. Luengo, J. Derrac, S. García, L. Sánchez, F. Herrera, KEEL data-mining software tool: data set repository, integration of algorithms and experimental analysis framework, J. Mult.-Val. Logic Soft Comput. 17 (2011) 255–287.
[52] M.-F. Han, S.-H. Liao, J.-Y. Chang, C.-T. Lin, Dynamic group-based differential evolution using a self-adaptive strategy for global optimization problems, Appl. Intell., (in press) doi:http://dx.doi.org/10.1007/s10489-012-0393-5.
[53] Y. Wang, Z. Cai, Q. Zhang, Differential evolution with composite trial vector generation strategies and control parameters, IEEE Trans. Evolution. Comput. 15 (1) (2011) 55–66.
[54] Y. Wang, Z. Cai, Q. Zhang, Enhancing the search ability of differential evolution through orthogonal crossover, Inform. Sci. 185 (1) (2012) 153–177.