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

A R T I C L E   I N F O
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

A B S T R A C T
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 NFS 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.

Crown Copyright © 2013 Published by Elsevier B.V. All rights reserved.

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–CPSO algorithm. In the FLNFN–CPSO 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.
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 NFs 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 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 $i$th rule, (Rule $i$), which is instantiated as:

**Rule $i$:** IF $x_1$ is $A_{1i}$ and $x_2$ is $A_{2i}$ and \ldots $x_n$ is $A_{ni}$ THEN $y_i = w_{0i} + w_{1i}x_1 + \ldots + w_{ni}x_n$.

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 $w_{0i}, \ldots, w_{ni}$ 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 $O^{(j)}$ denotes the output of a node in the $j$th 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,

$$O^{(1)} = x_i,$$  \hspace{1cm} (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:

$$O^{(2)} = \mu_{ji} = \exp\left(-\frac{(x_i - m_{ji})^2}{\sigma_{ji}^2}\right),$$  \hspace{1cm} (3)

where $j = 1, 2, \ldots, M$, $M$ is the number of rules in the NFS, and $m_{ji}$ and $\sigma_{ji}$ 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
\[ O^{(3)} = R_j = \prod_{i=1}^{n} n_{ij}. \]  

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

2.1.4. Layer 4—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

\[ O^{(4)} = y = \frac{\sum_{j=1}^{m} R_j (\sum_{i=1}^{m} w_{ij} x_j + w_{ij})}{\sum_{j=1}^{m} R_j}. \]  

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 \( i \)th individual at the current generation, \( \text{gen} \), is shown as \( Z_{\text{gen}} = (z_{1, \text{gen}}, z_{2, \text{gen}}, z_{3, \text{gen}}, \ldots, z_{(D-1), \text{gen}}, z_{D, \text{gen}}) \). 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 \( Z_{\text{gen}} \), \( i = 1, 2, \ldots, NP \), where \( \text{gen} \) denotes the current generation and \( NP \) is population size, three other random individuals, \( Z_{1, \text{gen}}, Z_{2, \text{gen}}, \) and \( Z_{3, \text{gen}} \), are selected from the population such that \( r_1, r_2 \) and \( r_3 \in \{1, 2, \ldots, NP\} \) and \( i \neq r_1 \neq r_2 \neq r_3 \). In this way, a parent pool of four individuals is formed to produce an offspring. The different mutation strategies are frequently used as follows:

- \( \text{DE/rand/1} : V_{\text{gen}} = Z_{1, \text{gen}} + F(2 \text{rand} - 1)(Z_{2, \text{gen}} - Z_{3, \text{gen}}) \)
- \( \text{DE/best/1} : V_{\text{gen}} = Z_{\text{best,gen}} + F(Z_{\text{gen}} - Z_{\text{best,gen}}) \)
- \( \text{DE/target} - \text{to} - \text{best} : V_{\text{gen}} = Z_{1, \text{gen}} + F(Z_{\text{best,gen}} - Z_{1, \text{gen}}) + F(2 \text{rand} - 1)(Z_{2, \text{gen}} - Z_{3, \text{gen}}) \)

where \( F \) is the scaling factor \( \epsilon[0,1] \) and \( Z_{\text{best,gen}} \) is the best-so-far individual (i.e., \( Z_{\text{best,gen}} \) 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 \( V_{\text{gen}} \) is mated with \( Z_{\text{gen}} \) and generates the offspring \( U_{\text{gen}} \). The elements of an individual \( V_{\text{gen}} \) are inherited from \( Z_{\text{gen}} \) and \( V_{\text{gen}} \), which are determined by the parameter crossover probability (CR \( \in [0,1] \)), as follows:

\[ U_{\text{d,gen}} = \begin{cases} V_{\text{d,gen}}, & \text{if rand(d) \leq CR} \\ Z_{\text{d,gen}}, & \text{if rand(d) > CR} \end{cases} \]  

where \( d = 1, 2, \ldots, D \) denotes the \( d \)th element of individual vectors, \( D \) is total number of elements of the individual vector and \( \text{rand(d)} \in [0,1] \) is the \( d \)th 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 \( Z_{\text{gen}} \) and its offspring \( U_{\text{gen}} \) and the winner, selected deterministically based on objective function values, is then promoted to the next generation. The selection operation is described as follows:

\[ Z_{\text{gen+1}} = \begin{cases} Z_{\text{gen}}, & \text{if fitness}(Z_{\text{gen}}) < \text{fitness}(U_{\text{gen}}) \\ U_{\text{gen}}, & \text{otherwise} \end{cases} \]  

where \( f(Z) \) 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-delii design

This section describes the structure learning and parameter learning for the design of NFSSs. An AFCM method for structure learning determines the number of rules automatically. Following rule generation, all of the free parameters in the NFSSs 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 (\( \sigma_{ij} \)) of that cluster. Many previous studies have employed a clustering technique, such as fuzzy c-means [43], possibilistic C-means [44], and linear vector quantization [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 sets. 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 \( A_{ij} \) was assigned as:

\[ m_{ij} = \frac{1}{N} \sum_{k=1}^{N} x_{i(k)} \]  

\[ \sigma_{ij} = \frac{1}{N} \left( \sum_{k=1}^{N} (x_{i(k)} - m_{ij})^2 \right) \]  

where \( x_{i(k)} \) is the input value of the \( i \)th input variable, \( \text{dim}_{i} \) is the input dimension, \( N \) is the total of input patterns, and \( M \) is the total of fuzzy rules. If \( i \leq \text{dim}_{i} \), where \( \text{dim}_{i} \in [0, 1] \) 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:

\[ I = \min \max R_{ij}(\tilde{x}_{i}), \quad 1 < i < N, \quad 1 < j < M \]  

where \( R_{ij} \) is the firing strength of the \( j \)th fuzzy rule with an \( i \)th dimensional input, \( N \) is the total of input patterns, and \( M \) is the total of fuzzy rules. If \( I < \text{dim}_{i} \), where \( \text{dim}_{i} \in [0, 1] \) 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:
the range $[0, 1]$, we conclude that the threshold is defined as 

$$Rth_{\text{spec}}$$

for rule generation is repeated until the criterion satisfies the predefined threshold. In the AFCM algorithm, the threshold $Rth$ 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 conclude 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 $\sigma_{ij}$ of a Gaussian membership function with the weight of the consequent part $w_{ij}$, where $i$ and $j$ represent the $i$th input variable and the $j$th rule, respectively.

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:

$$NFS_q = [\text{rule}_1^q | \text{rule}_2^q | \ldots | \text{rule}_n^q]$$

$$= [m_{11}^q + \Delta m_{11}^q, \sigma_{11}^q + \Delta \sigma_{11}^q, w_{11}^q; \ldots; m_{n1}^q + \Delta m_{n1}^q, \sigma_{n1}^q + \Delta \sigma_{n1}^q, w_{n1}^q]$$

where $m_{ij}$ and $\sigma_{ij}$ are the results of AFCM algorithm for the mean and width of the Gaussian membership function of:

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

$$\sigma_{ij} = \sum_{k=jth \text{ cluster}} \frac{(x_i(k) - m_{ij})^2}{N_j} \quad \text{for} \quad i = 1, \ldots, N_j \quad \text{and} \quad j = 1, \ldots, M$$
the jth rule of the ith input variable, $\Delta m_{ij}^a$ and $\Delta m_{ij}^b$ are small random deviations that are uniformly generated from the interval $[0.1, 0.1]$, and $w_i$ 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:

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

where $y_k$ represents the model output of the kth pattern, $\hat{y}_k$ 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 $Z_{gen}^{i}$, where $i = 1, 2, ..., NP$, and $NP$ denotes the population size, four other random individuals, $Z_{1,gen}^{i}, Z_{2,gen}^{i}, Z_{3,gen}^{i}$, and $Z_{4,gen}^{i}$ are selected from the population such that $r_1, r_2, r_3,$ and $r_4 \in [1, 2, ..., NP]$ and $1 \neq r_1 \neq r_2 \neq r_3 \neq r_4$. 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 $V_{j,gen}^{i}$ according to the following equation:

$$V_{j,gen}^{i} = \alpha(V_{local}^{i}) + (1 - \alpha)(V_{global}^{i}) + \alpha(Z_{gen}^{i} + F(Z_{best,gen}^{i} - Z_{gen}^{i})) + (1 - F)(Z_{r_1,gen}^{i} - Z_{r_2,gen}^{i}) + (1 - F)(Z_{r_3,gen}^{i} - Z_{r_4,gen}^{i}),$$

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

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 $\alpha$ 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 $V_{global}^{i}$, the new mutation operation in the DELI algorithm considers both global information $V_{global}^{i}$ and local information $V_{local}^{i}$. 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 $Z_{gen}^{i}$ 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 $\alpha$. 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.
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 exploitation ability. If the success probability $P$ is less than $1/5$, the algorithm increases the exploration 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 exploration ability, while a smaller scaling factor $F$ means that the DELI algorithm favours the exploitation 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 $F$ is also adjusted by parameter tuning. The significance of the scaling factor $F$ is to favour global information, local information or both for offspring based on mutation operation. In the interesting case when the scaling factor $F = 0$, the mutation operation degraded to conventional DE mutation operation as follows:

$$V_{i,gen} = Z_{i,gen} + F(Z_{gbest,gen} - Z_{i,gen}) + (1 - F)(Z_{i,gen} - Z_{r,gen}).$$

This mutation operation favours only global information for evolving a better individual. A similar situation occurs with the case of $F = 1$, 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 $F$ 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 $\beta$, we follow suggestions from journal papers [46,47]. They recommend $0.85 \leq \beta < 1$ and $1 \leq G_p \leq D$, where $D$ is the dimensions of $Z_{gen}$. 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 NFs. 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:

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

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 $\{x(t-24), x(t-18), x(t-12), x(t-6)\}$. 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 [from $x(501)$ 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
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 DE, and MODE over 50 runs. The results show that the DELI algorithm achieves better performance than other evolutionary algorithms. Fig. 8 shows the prediction results of the DELI algorithm compared with those of GA, PSO, DE, and 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 case 2. 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], SPSSO [24], MOGUL-TSK [48], CPSSO [49] and 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 \( \mu \left(0.15;0.16\right) \) and \( x_2 \) is \( \mu \left(0.97;0.16\right) \) and \( x_3 \) is \( \mu \left(1.03;0.15\right) \) and \( x_4 \) is \( \mu \left(1.91;0.07\right) \) THEN \( y = -0.43x_1 + 4.99x_2 + 0.52x_3 + 0.38x_4 + 2.94 \).

Rule 2: IF \( x_1 \) is \( \mu \left(0.69;0.19\right) \) and \( x_2 \) is \( \mu \left(0.66;0.05\right) \) and \( x_3 \) is \( \mu \left(1.01;0.15\right) \) and \( x_4 \) is \( \mu \left(0.80;0.17\right) \) THEN \( y = 0.15x_1 + 0.55x_2 - 0.18x_3 + 0.75x_4 + 0.03 \).

Rule 3: IF \( x_1 \) is \( \mu \left(1.09;0.03\right) \) and \( x_2 \) is \( \mu \left(0.96;0.03\right) \) and \( x_3 \) is \( \mu \left(1.31;0.20\right) \) and \( x_4 \) is \( \mu \left(0.74;0.07\right) \) THEN \( y = -0.84x_1 + 0.30x_2 - 0.29x_3 - 0.40x_4 + 4.69 \).

Rule 4: IF \( x_1 \) is \( \mu \left(1.25;0.15\right) \) and \( x_2 \) is \( \mu \left(1.02;0.14\right) \) and \( x_3 \) is \( \mu \left(0.85;0.12\right) \) and \( x_4 \) is \( \mu \left(1.12;0.21\right) \) 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 \( \sigma \).

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

\[
y(k + 1) = \frac{y(k)}{y^2(k) + 1} + u^1(k),
\]

where \(-2 \leq y(k) \leq 2\), 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:

\[
y_d(k) = \sin \left( \frac{\pi k}{50} \right) \cos \left( \frac{\pi k}{30} \right), \quad 1 \leq k \leq 250.
\]
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 $l(0.70;0.31)$ and $x_2$ is $l(0.49;0.19)$ and $x_3$ is $l(0.47;0.19)$ and $x_4$ is $l(1.37;0.38)$ and $x_5$ is $l(0.29;0.21)$ THEN $y = -7.18x_1 - 4.12x_2 - 7.24x_3 - 6.17x_4 + 2.81x_5 + 28.49$.

Rule 2: IF $x_1$ is $l(0.22;0.17)$ and $x_2$ is $l(0.45;0.08)$ and $x_3$ is $l(0.18;0.14)$ and $x_4$ is $l(0.19;0.24)$ and $x_5$ is $l(1.49;0.014)$ and $x_6$ is $l(1.48;0.014)$ and $x_7$ is $l(0.95;0.29)$ and $x_8$ is $l(-1.16;1.71)$ THEN $y = 29.84x_1 - 31.02x_2 - 35.30x_3 - 48.79x_4 + 45.61x_5 - 49.98$.

Rule 3: IF $x_1$ is $l(1.44;0.13)$ and $x_2$ is $l(1.49;0.014)$ and $x_3$ is $l(1.48;0.014)$ and $x_4$ is $l(0.95;0.29)$ and $x_5$ is $l(-1.16;1.71)$ 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 $l(0.63;0.52)$ and $x_2$ is $l(0.42;0.14)$ and $x_3$ is $l(1.01;0.38)$ and $x_4$ is $l(1.34;0.36)$ and $x_5$ is $l(1.46;1.32)$ 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. 7. Training RMSEs at each performance evaluation for the DELI algorithm and other algorithms in case 1.](image1)

![Fig. 8. Symbol “+” represents the desired results and “O” represents the prediction results of the DELI algorithm for NFSs optimisation in case 1.](image2)

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

<table>
<thead>
<tr>
<th>Method</th>
<th>Testing RMSE</th>
</tr>
</thead>
<tbody>
<tr>
<td>NFS-DELI</td>
<td>0.023</td>
</tr>
<tr>
<td>HGAPSO [28]</td>
<td>0.047</td>
</tr>
<tr>
<td>SPSO [24]</td>
<td>0.054</td>
</tr>
<tr>
<td>MOGUL-TSK [48]</td>
<td>0.045</td>
</tr>
<tr>
<td>CPSO [49]</td>
<td>0.032</td>
</tr>
<tr>
<td>HPSO-TVAC [25]</td>
<td>0.047</td>
</tr>
</tbody>
</table>
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.
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 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.

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

<table>
<thead>
<tr>
<th>Method</th>
<th>GA</th>
<th>PSO</th>
<th>DE</th>
<th>MODE</th>
<th>DELI</th>
</tr>
</thead>
<tbody>
<tr>
<td>Rule number</td>
<td>4 ± 0.8</td>
<td>4 ± 0.8</td>
<td>4 ± 0.8</td>
<td>4 ± 0.8</td>
<td>4 ± 0.8</td>
</tr>
<tr>
<td>Training RMSE Mean</td>
<td>3.89</td>
<td>3.35</td>
<td>3.27</td>
<td>2.51</td>
<td>2.36</td>
</tr>
<tr>
<td>STD</td>
<td>0.51</td>
<td>0.56</td>
<td>0.56</td>
<td>0.22</td>
<td>0.15</td>
</tr>
<tr>
<td>Testing RMSE Mean</td>
<td>4.02</td>
<td>3.66</td>
<td>3.61</td>
<td>2.89</td>
<td>2.58</td>
</tr>
<tr>
<td>STD</td>
<td>0.77</td>
<td>0.68</td>
<td>0.72</td>
<td>0.34</td>
<td>0.21</td>
</tr>
<tr>
<td>p-Values</td>
<td>8.53 $\times 10^{-19}$</td>
<td>5.36 $\times 10^{-19}$</td>
<td>5.06 $\times 10^{-09}$</td>
<td>2.97 $\times 10^{-05}$</td>
<td>–</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Method</th>
<th>Testing RMSE</th>
</tr>
</thead>
<tbody>
<tr>
<td>NFS-DELI</td>
<td>2.58</td>
</tr>
<tr>
<td>HGAPSO [28]</td>
<td>2.97</td>
</tr>
<tr>
<td>MOGUL-TSK [48]</td>
<td>5.16</td>
</tr>
<tr>
<td>CPSO [49]</td>
<td>2.66</td>
</tr>
<tr>
<td>HPSO-TVAC [25]</td>
<td>2.72</td>
</tr>
</tbody>
</table>

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 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.
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


