IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received July 5, 2017, accepted September 4, 2017, date of publication September 15, 2017,
date of current version October 12, 2017.

Digital Object Identifier 10.1109/ACCESS.2017.2752901

An Adaptive Memetic Algorithm With Rank-Based Mutation for Artificial Neural Network Architecture Optimization

WEIGUO SHENG[1], PENGXIAO SHAN[2], JIAFA MAO[2], YUJUN ZHENG[1], (Member, IEEE), SHENGYONG CHEN[2], (Senior Member, IEEE), AND ZIDONG WANG[3], (Fellow, IEEE)
[1]Hangzhou Normal University, Hangzhou 311121, China
[2]Zhejiang University of Technology , Hangzhou 311121, China
[3]Department of Computer Science, Brunel University London, Uxbridge UB8 3PH, U.K.

Corresponding author: Weiguo Sheng (w.sheng@ieee.org)

ABSTRACT Designing a well-generalized architecture for artificial neural networks (ANNs) is an important task. This paper presents an adaptive memetic algorithm with a rank-based mutation, denoted as AMARM, to design ANN architectures. The proposed algorithm introduces an adaptive multi-local search mechanism to simultaneously fine-tune the number of hidden neurons and connection weights. The adaptation of the multi-local search mechanism is achieved by identifying effective local searches based on their search characteristics. Such an algorithm is distinguishable from previous evolutionary algorithm-based methods that incorporate one single local search for evolving ANN architectures. Furthermore, a rank-based mutation strategy is devised for avoiding premature convergence during evolution. The performance of the proposed algorithm has been evaluated on a number of benchmark problems and compared with related work. The results show that the AMARM can be used to design compact ANN architectures with good generalization capability, outperforming related work.

INDEX TERMS Artificial neural networks (ANNs), evolutionary algorithm, rank based mutation, adaptation strategy, local searches.

I. INTRODUCTION

Artificial neural networks (ANNs) have been widely applied in scientific problems such as pattern recognition [1], classification [2], regression [3] and dynamic system control [4]–[6]. The performance of ANNs is greatly dependent on their architectures. An excessively large architecture may overfit the training set, due to its excess information processing capability. On the other hand, an excessively small architecture can underfit the training set, due to its limit information processing capability. Both overfitting and underfitting will lead to poor generalizations. Thus, designing appropriate ANN architectures is essential to effectively solve various problems.

The Backpropagation (BP) training algorithm [7] and its variants have been traditionally used to train a fixed ANN architecture. These algorithms, however, suffer from the local optima issue [8], and may not deliver ANN architectures with a satisfactory performance within a given training period. Constructive methods [9]–[11] and pruning methods [12]–[14] based on hill–climbing search have also been developed for designing ANN architectures. Such methods typically explore a small architectural space and could also be trapped into local optimal solutions [15].

To address the local optima issue, stochastic algorithms have been proposed, a prominent approach of which is the evolutionary algorithms (EAs) [16]. Due to their good global search capabilities, EAs have been popularly used to evolve ANN architectures [17]–[21]. Existing methods can be generally classified into two major types [18]: the ''invasive'' approach, which relies solely on EAs to evolve ANNs' architectures and the ''noninvasive'' approach, in which EAs are incorporated with a certain local search to evolve ANNs' architectures. The major issue of the ''invasive'' approach is that it has little capability of fine-tuning the solutions and thus usually takes a relatively long time to locate the optima in a region of convergence. On the other hand, the ''noninvasive'' approach, which incorporates local search, can greatly improve the time efficiency of evolutionary process.

However, employing one single local search, as usually adopted in existing methods, may not well suit to solve the complex optimization problem of designing ANN architectures.

Recently, a few studies [22]–[24] have shown that EAs incorporating multiple local searches are promising for solving complex optimization problems. In these methods, multiple local searches are employed to cooperatively and competitively search the solution space. EAs hybridized with local searches are often referred to as Memetic Algorithms (MAs) [25]. Since we are concerned here with EAs in which local searches play a critical role, this term will be used in the paper.

Although MAs can generally deliver solutions more efficiently than traditional EAs, they also suffer from the premature convergence [26]. To deal with this issue, controlling the algorithm's parameters, especially the mutation probability is a promising approach. This is confirmed by studies such as in [27] and [28], which show that appropriately controlling the EA's mutation probabilities can help maintain the balance between exploration and exploitation of the search space, thus alleviating the premature convergence. However, how to adapt the mutation probabilities of individuals for improving the EA's search capability is a vital yet open issue.

In this paper, we propose an algorithm, called adaptive memetic algorithm with rank based mutation (AMARM), for designing ANN architectures. In the proposed algorithm, we introduce an adaptive multi-local search mechanism to fine-tune the ANN architectures. Three local searches are employed in the mechanism. The first two aim to refine the number of hidden neurons in ANN architectures, while the third one is used to fine-tune connection weights. The adaptation is achieved based on a strategy, which is devised to choose an appropriate local search from three local searches based on their search characteristics. Furthermore, a rank based mutation strategy, which assigns different mutation probabilities to different individuals based on their fitness ranks, has been designed to avoid premature convergence during evolution. The experimental results show that the proposed multi-local search mechanism plays an important role of effectively exploiting the decision space, while the rank based mutation helps avoid the premature convergence. As a result, our proposed algorithm is able to deliver ANN architectures with satisfactory performance.

The key contributions of this work are as follows:

- An adaptive multi-local search mechanism, in which two purposely-devised local searches along with the BP algorithm are dynamically employed to fine-tune the ANN's structure as well as connection weights simultaneously, is proposed and incorporated into an EA for efficiently exploiting the decision space of ANN architectures;
- A rank based mutation strategy is introduced to alleviate the premature convergence of EA search;
- A "sliding-windows" based cross validation is introduced as the termination condition of the evolution to reduce the ANNs' generalization loss.

This work is an extension of its shorter conference version [29], in which an adaptive multi-local search mechanism based memetic algorithm for ANN architecture optimization is briefly described along with some preliminary results. In this work, we extend our previous paper by giving a comprehensive description of the proposed mechanism and providing extensive experiments as well as comparison studies to analyze its behavior and effectiveness. Further, the algorithm proposed in [29] has been extended by introducing and incorporating a rank based mutation strategy, which is used to maintain the diversity of population. Consequently, more accurate results can be achieved comparing to our previous work and also other related methods in literature on the same datasets.

The rest of this paper is organized as follows. Section II discusses related work. Section III presents our proposed algorithm. This is followed by a detailed description of adaptive multi-local search mechanism in Section IV. Section V evaluates the proposed algorithm and compares it with related methods. Finally, Section VI concludes the paper with a brief summary and several further directions.

II. RELATED WORK

A number of heuristic methods have been proposed to design ANN architectures. Hill-climbing based constructive and pruning methods are popular ones. The constructive methods start with a minimal network and then add new layers, neurons as well as connections if necessary during training, while the pruning methods do the opposite by deleting unnecessary layers, neurons and connections from an oversized network. For example, Islam *et al.* [10] proposed a constructive algorithm (NCA) to determine the complete topological information of a feedforward ANN architecture. This algorithm emphasizes on both architectural and functional adaptation. NCA trains hidden neurons by using different training sets, which is based on the performance of existing ANN architectures, and gradually adds hidden neurons or layers to the ANN's architecture. Lauret *et al.* [14] proposed a pruning algorithm to obtain the ANN's architecture. The relevance of hidden neurons is determined by the global sensitivity analysis of model output. According to the obtained information, the most unfavorable neuron will be eliminated. A good review can be found in [30] and [31] for constructive algorithms and in [32] for pruning algorithms. Recently, hybrid methods [33], [34], which combine the constructive and pruning algorithms have also been proposed in literature. For instance, Islam *et al.* [33] developed an adaptive merging and growing algorithm to design the ANN architecture. During the adaptation process, this algorithm merges and adds hidden neurons repeatedly or alternatively, aiming to produce compact ANN architectures. Such an algorithm can quickly locate a near optimal ANN architecture. However, by employing the hill-climbing search, it is susceptible to local optima.

Recently, EAs have been widely employed to design ANN architectures including connection weights, structures and learning rules [35], [36]. For example, Yao and Liu [17] proposed a method called EPNet to design ANN architectures. The EPNet is based on evolutionary programming (EP) with five mutation operators, which is used to reduce the detrimental effect of permutation problem. In addition, in the EPNet, BP algorithm is adopted as partial training to improve the computational efficiency. Such a method is known to suffer from the noisy fitness evaluation problem due to its continued reliance on BP algorithm, which is highly sensitive to initial weights. To alleviate this issue, Palmes *et al.* [18] developed a mutation-based genetic neural network, in which instead of the BP algorithm, a scheduled mutation strategy is employed. While in [21], Ong and Isa devised a hybrid evolutionary artificial neural network (HEANN) for simultaneously evolving the ANN's topology and weights. The HEANN combines a global mutation probability with a local mutation probability to search for the near optimal or optimal ANN architecture. The above methods are all based on EP and emphasis on the preservation of behavioral links between parent and child solutions. In these methods, the crossover operators, which recombine one part of an ANN with another part of an ANN, may destroy both ANNs [35]. Further, although these methods can deliver better solutions as compared to hill-climbing based methods, they generally require a large amount of time to converge.

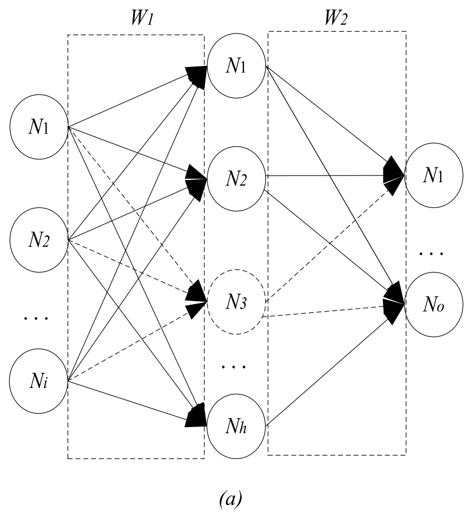
In order to improve the computational efficiency, hybrid methods [37], [38], [58] that incorporate local searches into EAs have gained popularity for designing the ANN architectures. For instance, Lu *et al.* [39] presented a hybrid learning method by incorporating the BP into GA to design ANN architectures. Similar method has also been proposed by Nikolaev and Iba [40], in which the network structure is first determined by a genetic programming algorithm and then improved by the BP algorithm. In [41], Martínez-Estudillo *et al.* performed a cluster analysis to group the ANN individuals, followed by employing the Levenberg-Marguardt (LM) based local search to fine-tune the best individual in each cluster. Tsai *et al.* [42] proposed to incorporate the Taguchi method into a GA for designing ANNs. Leung *et al.* [43] developed an improved genetic algorithm, which has the ability to tune the structure and parameters of neural networks. In [59], the BP algorithm was combined with a differential evolution (DE) algorithm for optimizing the weights of ANN. In this method, the DE is first used to identify promising regions of the search space. Then, the BP method is applied to move solutions towards the optima. In [60], Zhang *et al.* devised a hybrid algorithm, called ODE-LM, in which an orthogonal differential evolution algorithm is combined with the LM method, to optimize the weights and biases of ANN. In this method, the orthogonal differential evolution is employed to optimize network weights for a certain generations. This is followed by the LM method until a user-specified maximum number of iterations being reached. In the above methods, one single local search is considered and applied to refine-tune the connection weights of ANN. This, however, may not well suit to fine-tune the entire ANN architecture and could significantly limit their generalization ability for solving a given problem.

Apart from EAs, swarm intelligence algorithms have also been used for designing the ANN. For example, in [64], three variants of particle swarm optimization (PSO) algorithms (i.e., basic PSO, second generation of PSO and a new model PSO) are employed to evolve the weights, connections and transfer functions of ANN. Yaghini *et al.* [61] proposed to combine a PSO with the BP algorithm to train ANN. Nandy *et al.* [62] presented a hybridization of bee colony algorithm with BP for training ANN. While in [63], Socha and Blum devised a hybrid ant colony optimization method, in which short runs of classical gradient techniques are employed as the local search, to optimize ANN. Same as hybrid EAs for ANN design, in these methods, one single local search is considered and employed to refine-tune the connection weights of ANN.

Rather than hybridizing global and local search, an approach, which combines two global optimization algorithms, has also been developed to design ANN. For example, Almeida and Ludermir [44] combined evolution strategy with PSO to form a hybrid intelligent system for ANN design. In [45], on the other hand, a GA was hybridized with PSO, resulting a method called HGAPSO, for recurrent neural/fuzzy network design. In this method, individuals in a new generation are created not only by crossover and mutation operation as in GA but also by PSO.

More recently, multiple local searches have been incorporated into EAs to solve optimization problems other than designing the ANN architectures. In [46], Kononova *et al.* designed an evolutionary framework, in which three local searches are used for solving the inverse problem. These three local searches are adaptively governed by a fitness diversity based measure. Similarly, Caponio *et al.* [47] devised two local searches with different search characteristics, pivot rule and neighborhood, for online and offline control design of permanent-magnet synchronous motor. In [65], Dinneen and Wei presented a theoretical study of an adaptive MA with two local searches (i.e., random permutation and random complete) along with several other variants of EAs for maximum clique problem. In [48], a multi-local search mechanism based MA is investigated for automatic data clustering. In this local search mechanism, three purposely-designed local searches, each with different search features, are employed to adaptively exploit the decision space of data clustering. These studies have shown that MAs with multiple local searches can lead to a better performance, compared with one single local search based MAs. In this paper, we introduce an adaptive multi-local search mechanism to simultaneously fine-tune the number of hidden neurons and connection weights of ANN architectures. To the best of our knowledge, no such work has yet been reported in literature.

```
Algorithm 1 An Adaptive Memetic Algorithm With Rank Based Mutation for Designing ANN Architectures
Step 1: Generate an initial population of ANNs with three layers. Each ANN is encoded using the real-value representation.
        The number of hidden neurons and connections are generated randomly, and the initial connection weights are also
        randomly chosen in the range between -1 and 1.
Step 2: Calculate the fitness value according to the fitness function for each ANN individual in the initial population.
Step 3: Repeat the following steps (a) to (f) until the stopping criterion is met.
        (a) Select solution pairs based on the fitness rank of the ANN individuals in population.
        (b) Apply rank based mutation on each ANN individual from the best to the worst.
        (c) Perform neuron splitting or merging or none of these two local searches to fine-tune the number of hidden neurons
            of ANN architectures.
        (d) Apply one iteration of BP algorithm to fine-tune the connection weights of ANN architectures.
        (e) Calculate the fitness value for each offspring according to the fitness function.
        (f) Create a new population from the individuals of previous population and their offspring.
Step 4: Output the best ANN identified.
```



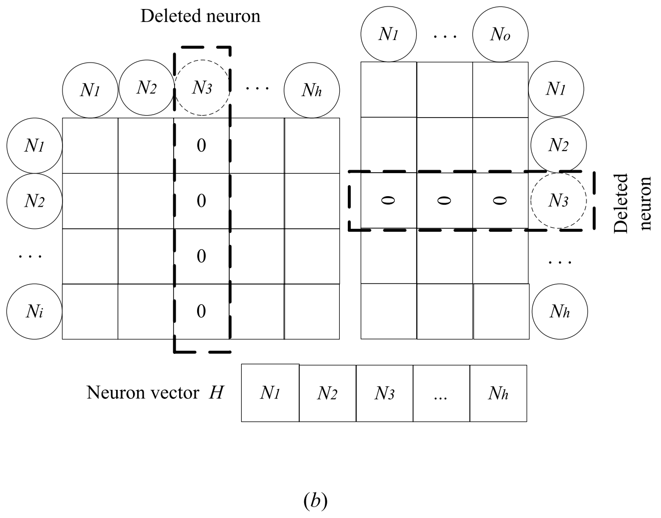


FIGURE 1. **(*a*) Three-layer feedforward neural network. (*b*) The direct encoding scheme.**

III. THE PROPOSED ALGORITHM

In this section, we present a memetic algorithm that combines multi-local searches with rank based mutation (AMARM) for designing ANN architectures. In our algorithm, multi-local searches are employed to efficiently exploit the decision space. The adaptive choice of multi-local searches is determined by their local search characteristics. Furthermore, a rank based mutation strategy is introduced in our algorithm to avoid premature convergence. Additionally, to avoid overfitting or uderfitting during evolution, we employ a ''sliding-windows'' based cross validation as the stopping criterion. The evolution of the ANN architectures is terminated when the stopping criterion is met. The output of the algorithm is the best ANN architecture identified during evolution. The procedure of our proposed algorithm is shown in Algorithm 1.

In the following subsection, we describe each component of the AMARM algorithm including the representation of solutions, fitness function, rank based mutation and stopping criteria, while details of the proposed adaptive multi-local search mechanism are presented in Section IV.

A. REPRESENTATION OF SOLUTIONS

Both binary [49] and real-value based representations [18], [57] are commonly employed to encode the connection weights and topology of ANNs. In a binary representation scheme [49], an ANN is encoded by concatenation of all connection weights, which are represented by a number of bits. The disadvantage of binary representation lies in the length of solution. Since the solution representing large ANNs will become extremely long, the evolution of ANN in turn will become inefficient. In this work, we adopt the real-value based representation, which is able to improve the efficiency of evolution [18], [57].

Fig. 1(a) shows a three-layer feedforward neural network structure, which consists of one input layer, one output layer and a set of hidden neurons employing the real-value based representation. Each ANN structure has $N_i+N_h+N_o$ neurons, where N_i, N_h and N_o are the numbers of input, hidden, and output neurons, respectively. The maximum number of hidden neurons N_{hmax} allowable in the ANN is a user-specified parameter, whereas the number of input and output neurons is problem dependent. Here, the dimensional vector H represents hidden neurons of a particular ANN, whose values are binary entries indicating the existence of these neurons.

Fig. 1(b) shows a sample of the neuron vector and two weight matrixes, which represents the architecture of three-layer-feedforward neural network. The matrix W_1 denotes connection weights, which is from the input layer to hidden layer, while the matrix W_2 denotes connection weights, which is from the hidden layer to output layer. The advantage of such an encoding scheme is that it supports implicitly both topology evolution and weight adaption. A zero entry in the neuron vector indicates that the particular neuron does not exist.

B. FITNESS FUNCTION

The fitness function in the proposed algorithm considers three criteria: training error (f_t), network complexity (f_c) and classification error rate on the validation set (f_e), optimizing of which aims to obtain compact NN architectures with good generalization capability. Specifically, the fitness function can be written as:

$$F = a_1 \times f_t + a_2 \times f_c + a_3 \times f_e \tag{1}$$

where a_1, a_2 and a_3 are user-defined parameters with values between 0 and 1. The training error (f_t), network complexity (f_c) and classification error rate (f_e) are defined as:

$$f_t = 100 \times \frac{(O_{max} - O_{min})}{NP} \sum_{j=1}^{P} \sum_{i=1}^{N} (T_i - O_i)^2 \tag{2}$$

$$f_c = \frac{C}{C_{tot}} \tag{3}$$

$$f_e = (1 - \frac{correct}{total}) \tag{4}$$

where O_{max} and O_{min} are the maximum and minimum values, respectively, of output coefficient. N and P are the number of output neurons and training patterns, respectively. T_i and O_i are the target and network output, respectively. For three-layered feed-forward architectures, the value of C_{tot} is set based on the size of its input, output as well as the user-specified maximum number of hidden neurons. Specifically, $C_{tot} = N_i^* N_{hmax} + N_{hmax}^* N_o$. The network complexity is measured using the equation (3) in term of the ratio between active connections C and total number of possible connections C_{tot}. The percentage error of classification on the validation set is calculated using equation (4). Note that the lower fitness value indicates a fitter individual.

In the above fitness function definition, the term f_c is used to penalize large networks while the f_e is employed to alleviate the issue of overfitting by minimizing the term f_t alone. The user-defined constant a_1, a_2 and a_3 are used to control the significance of three terms for fitness calculation and they are experimentally set to be 1.0, 0.1 and 0.3, respectively.

C. RANK-BASED MUTATION

During the evolution of ANN architectures, the mutation is regarded as a major operation to explore decision spaces. Existing EA based methods for designing ANN architectures are usually based on the mutation operation with a fixed probability. This, however, may not be capable of effectively exploring the solution space, thus leading to premature convergence. Here, we present a rank based mutation strategy, in which the probabilities of mutation operation are set based on both the individuals' fitness as well as their fitness ranks in the population. Specifically, in our strategy, the individual's fitness rank is used to adapt its mutation probability. While, the individual's fitness is used to penalize the severity of its mutation probability. The idea is to assign smaller mutation probabilities for exploiting the search space around fitter individuals and larger mutation probabilities to the worse individuals for exploring the search space. Hence, such a mutation strategy can properly maintain the balance between the exploration and exploitation of search space, thus help avoid premature convergence during evolution. Based on the above idea, the mutation probability for each individual in the population is computed as:

$$P_i = P_{min} + (P_{max} - P_{min}) \times r \times \frac{F_{ave}}{F(i)} \tag{5}$$

where P_{min} and P_{max} are the minimum and maximum mutation probability, respectively. Here, $r = (rank(i)-1)/(m-1)$, m is the size of the population and $rank(i)$ is the fitness rank of individual i in the current population. In the equation, $r = 0$ implies that the best individual has the minimum mutation probability, while $r = 1$ implies that the worst individual has the maximum mutation probability. F_{ave} is the average fitness value of the population at the current generation and $F(i)$ denotes the fitness value of i^{th} individual in population. The penalty term $F_{ave}/F(i)$ is used to normalize the individual's mutation probability. If the individual's fitness is larger than the average fitness, its mutation probability will be reduced. Otherwise, the individual's mutation probability will be increased.

D. STOPPING CRITERION

Terminating the evolution at an appropriate time is also essential. A "sliding-windows" based cross validation method has been used as the stopping criterion in our work. In this method, the training data is split into training and validation sets in a certain proportion. The classification error rate on the validation set is then used to terminate the evolutionary process, with the purpose to avoid the evolved ANNs losing its generalization. At the end of every S-generation (i.e., sliding-window), the generalization loss (GL) is computed as:

$$GL = 100 \times (\frac{E_{va} + 1}{E_{opt} + 1} - 1) \tag{6}$$

where E_{va} and E_{opt} denote the validation error rates of the fittest ANN at the current generation and the fittest ANN identified so far during evolution, respectively.

In our stopping criterion, a stopping counter T with an initial value of zero is used to flag the stopping signal. Once the stopping counter T is larger than a user-defined constant, the evolutionary process terminates. Specifically, the procedure of stopping criterion is implemented as follows. During evolution, at the end of every S-generation, the GL value is calculated. If $GL \geq 0$ is obtained, which means overfitting may appear in the population, then the stopping counter T increases by one. Since $GL \geq 0$ could be caused by premature convergence, terminating the evolutionary process based solely on the GL may not be appropriate in such a situation. To deal with this circumstance, we allow the evolution to continue for a few extra S-generations (i.e., sliding-window) until the stop counter T is larger than a user-defined constant. During the extra S-generations, if the best ANN in the population can be improved further, then the stopping counter T is reinitialized to zero and the present best ANN is updated. This process repeats until the stopping criterion is met. At the end of evolution, the best ANN with lowest classification error on the validation set is selected as the output.

IV. ADAPTIVE LOCAL SEARCHES

EAs are capable of exploring promising regions of the search space. However, they are not well suited to fine-tune solutions and usually require a large amount of time to locate the optima in a region of convergence [50]. To improve the time efficiency, incorporation of local searches into the regeneration steps of EAs, creating the so-called MAs [25], is essential. In this section, we present three local searches to design a MA for simultaneously fine-tuning the number of hidden neurons and connection weights of ANN architectures.

A. NEURON MERGING, NEURON SPLITTING, AND ADAPTIVE STRATEGY

In this subsection, we first introduce two local searches, neuron merging and splitting. Neuron merging is based on pruning algorithms, which start with an oversized network and then delete unnecessary neurons if necessary during training. On the other hand, neuron splitting is based on constructive algorithms that do the opposite by adding new neurons to a minimal network. Then, an adaptive strategy is given for employing these two local searches to refine the number of hidden neurons.

The two local searches are employed to merge or split the hidden neurons, whose significance is calculated as:

$$\rho_i = \partial_i + \beta \sum_{ij} \frac{W_{ij}^2}{1 + W_{ij}^2} \tag{7}$$

where ∂_i is the standard deviation computed based on the outputs of i^{th} neuron on the training set. W_{ij} is the connection weight from neuron j to i, while β is a user-defined constant.

The significance ρ_i is based on the variation of hidden neurons' output and weight decay. A smaller value of ρ_i means the i^{th} hidden neuron is less significance. If the value ∂_i is close to zero, the i^{th} hidden neuron almost delivers constant information to output neurons, thus unable to distinguish different training examples. In other words, the characteristics of such a hidden neuron become redundant. The weight decay is used to alleviate the problem of noisy pattern, with the purpose of improving the ANN's capability to deal with noisy data.

The neuron merging local search works by choosing two insignificant hidden neurons whose value of ρ_i are lowest, then merging them into one new hidden neuron. Suppose to merge two neurons h_a and h_b, the input and output connection weights of the new neuron h_m are set using the following equations:

$$w_{mi} = \frac{w_{ai} + w_{bi}}{2}, \quad i = 1, 2, \ldots p \tag{8}$$

$$w_{jm} = w_{ja} + w_{jb}, \quad j = 1, 2, \ldots q \tag{9}$$

where p and q are the number of neurons in input and output layers, respectively, of the ANN. The weights w_{ai} and w_{bi} are i^{th} input connection weights of h_a and h_b, respectively, while w_{ja} and w_{jb} are the j^{th} output connection weights. The weights w_{mi} and w_{jm} are the i^{th} input and j^{th} output connection weights, respectively, of h_m.

The neuron splitting local search works by first choosing an existing hidden neuron, which has the highest value of ρ_i. The selected hidden neuron is then split into two new hidden neurons. The two newly generated neurons have the same number of weight connections as their parents. The weight connections of new neurons are computed as:

$$w^1 = (1 + \theta) \times w \tag{10}$$

$$w^2 = -\theta \times w \tag{11}$$

where w is the weight of existing neuron, w^1 and w^2 are the weights of the two generated neurons. The value of θ should be within a small range to avoid a large change in the existing network's functionality.

Neuron merging reduces the network size by pruning redundant hidden neurons. This local search encourages decorrelation among hidden neurons in ANN architectures and helps reduce the amount of training epochs for the modified ANN architectures. On the other hand, the neuron splitting operation increases the network size by splitting one existing hidden neuron. This operation can increase the information processing capacity while preserving the behavioral link between the parent and offspring ANNs. The above two local searches generate local improvements by merging redundant hidden neurons or splitting significant hidden neurons in ANN architectures, thus speeding up the search process of identifying the proper number of hidden neurons in ANN architectures. Note, these local searches improve the solutions from rather different aspects and they should not be simultaneously used on the same individual.

To appropriately use these two local searches during evolution, an adaptation strategy is then devised. Before describing the strategy, we first introduce a criterion used for the local

TABLE 1. Characteristics of the six benchmark classification problems.

Data sets	Input variables	Output class	No. of training examples	No. of validation examples	No. of testing examples
Iris	4	3	75	38	37
Wine	13	3	89	45	44
Diabetes	8	2	384	192	192
Card	14	2	345	173	172
Blood	4	2	374	187	187
Glass	9	6	107	54	53

search selection. In many constructive algorithms [9], [10], the criterion $E(t) - E(t+1) < \varepsilon$ is typically used to test the neuron addition. However, when using such a criterion the number of hidden neurons in the ANN architectures is usually unknown beforehand. Furthermore, in order to search for a near optimal or optimal ANN architecture, we encounter a situation where the processing information of ANNs depends largely on the number of hidden neurons. Hence, we introduce a criterion by adding a penalty term $(k-1)/(N_{hmax} - k)$ into the traditional criterion, as follows:

$$(E(t) - E(t+1)) \times (k-1)/(N_{hmax} - k) \leq \varepsilon \qquad (12)$$

where $E(t)$ and $E(t+1)$ are the classification error rates on training set at t and $t+1$ generation, respectively. Here, k is an input variable denoting the number of hidden neurons in ANN architectures at $t+1$ generation and ε is a user-specified parameter, which is set to be 0.05 in this work. The resulting criterion considers relation between the number of hidden neurons and processing information capability of the ANN architecture. If there are many hidden neurons in ANN architectures, the neuron addition will be given a large penalty for preventing neuron addition, thus obtaining a compact ANN architecture. Based on the above criterion, our devised adaptive strategy works as follows: For each individual, when the above criterion is satisfied during evolution, the neuron splitting operation will be applied on the individual. On the other hand, if more than two hidden neurons whose ρ_i values are found to be close to zero and the value of above criterion is greater than ε, then the neuron merging operation will be activated. Otherwise, neither of these two local searches will be employed, since both of them become ineffective for the individuals.

B. BP ALGORITHM

Next, we introduce a local search, which is employed to fine-tune the connection weights of ANN architectures. When ANN architectures are modified by a local search such as neuron merging or splitting, the weight training then becomes an important issue [10], [51]. To address this issue, the BP algorithm [52] has been further employed. BP algorithm is an iterative algorithm for optimizing connection weights by minimizing the sum squared error (SSE) criterion. Generally, as a local search, intensive application of the BP algorithm will lead to the problem of noisy fitness evaluation. In addition, this algorithm could consume a large amount of time. Here, one iteration of BP algorithm has thus been used as the third local search to fine-tune connection weights of ANN architectures, after performing either neuron merging or splitting or none of these two local searches. Such a local search can exploit the search space from a different but complementary perspective that offered by the neuron merging or splitting local searches. These three local searches can cooperate for the common goal of identifying the optimal or near optimal solution.

V. EXPERIMENTS

In this section, we evaluate the performance of proposed algorithm and compare it with related work on six benchmark classification problems [53], which have been the subject of many studies in the field of ANNs and machine learning. These six problems have various numbers of input attributes, output class and data patterns, which are summarized in Table 1. To illustrate the capability of our proposed algorithm for ANN design, we first examine the significance of multi-local search mechanism. Then, the impact of rank based mutation is investigated. Finally, we compare the proposed method with related methods. Unless otherwise stated, all simulations of the results were averaged over 50 independent runs. The test error rate (TER) refers to the percentage of classification error produced by the evolved ANNs on the testing set. The "epochs" reported in this work is the total number of fitness evaluation taken by the algorithm.

A. EXPERIMENTAL SETTINGS

In this work, we conduct experiments in accordance with benchmarking methodologies, which have been popularly employed in previous studies [17], [18], [21]. The data set of each problem is partitioned into three sets based on the partition rule used in [54]. The training set is used for training and modifying ANN architectures. While the validation and testing sets are used for terminating the training process and measuring the generalization ability of an evolved ANN, respectively. The number of examples in these subsets is

TABLE 2. Parameter setting of the proposed method.

Description	Setting
N_{hmax} (maximum number of hidden neurons allowable in ANN architectures)	N_{hmax}=10
a_1, a_2 and a_3 (constants used to control the strength of three terms in fitness function)	a_1=1, a_2=0.1 and a_3=0.3
P_{min} and P_{max} (minimum and maximum of the mutation probability)	P_{min}=0.01, P_{max}=0.05
S and T (interval generation of sliding-window and the maximum stopping counter, respectively, used in the stopping criterion)	S=5, T=3
β (coefficient of weight decay)	β=0.01
θ (coefficient of weight of new hidden neurons)	θ=0.01
ε (error threshold used in neuron merging or splitting)	ε=0.05
M (population size)	M=100
The learning and momentum rates used in BP algorithm.	0.1, 0.8

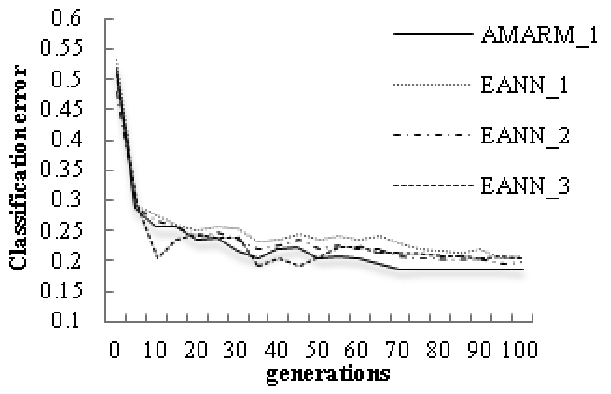
TABLE 3. Comparing the performance of AMARM and its variants on six benchmark classification problems.

Problems	Algorithms	Connections (Std. Dev.)	Epochs (Std. Dev.)	TER (Std. Dev.)
Iris	AMARM	**26.42(7.08)**	**6637.5(364.5)**	**1.64(1.16)**
	AMARM_1	28.42(7.35)	9916.5(996.0)	1.67(1.71)
	AMARM_2	29.75(4.56)	7100.6(453.0)	2.07(1.32)
	AMARM_3	29.50(9.83)	8916.5(194.5)	1.69(1.21)
Wine	AMARM	70.75(10.99)	**9937.5(495.5)**	**2.56(1.06)**
	AMARM_1	**66.35(9.62)**	11328.6(519.2)	3.64(1.32)
	AMARM_2	75.30(11.61)	10126.8(358.4)	3.31(1.25)
	AMARM_3	72.19(10.29)	13684.0(462.9)	3.17(1.09)
Card	AMARM	69.55(11.91)	**5772.5(410.0)**	**13.79(1.67)**
	AMARM_1	**57.33(25.39)**	9543.0(838.5)	15.36(1.29)
	AMARM_2	64.33(12.44)	9541.5(144.5)	14.73(0.72)
	AMARM_3	81.92(14.41)	8208.5(334.5)	14.05(0.55)
Diabetes	AMARM	80.23(11.84)	12231.0(483.5)	**21.19(1.07)**
	AMARM_1	**38.75(4.39)**	15458.5(689.0)	21.70(2.43)
	AMARM_2	43.75(10.04)	13833.5(589.0)	21.31(1.28)
	AMARM_3	52.58(19.16)	**11791.5(534.5)**	21.96(1.99)
Blood	AMARM	44.29(4.46)	**4956.0(451.0)**	**12.75(0.80)**
	AMARM_1	46.74(6.41)	8966.5(637.4)	14.39(2.56)
	AMARM_2	45.35(6.29)	6954.4(622.9)	13.44(1.37)
	AMARM_3	**39.66(4.23)**	6336.1(563.9)	12.94(1.43)
Glass	AMARM	**85.74(10.73)**	**9890.2(655.5)**	**44.45(3.23)**
	AMARM_1	129.04(11.35)	12692.3(723.2)	63.91(3.32)
	AMARM_2	103.66(12.28)	11965.9(839.5)	60.25(4.62)
	AMARM_3	92.95(10.58)	12118.3(685.3)	55.21(3.16)

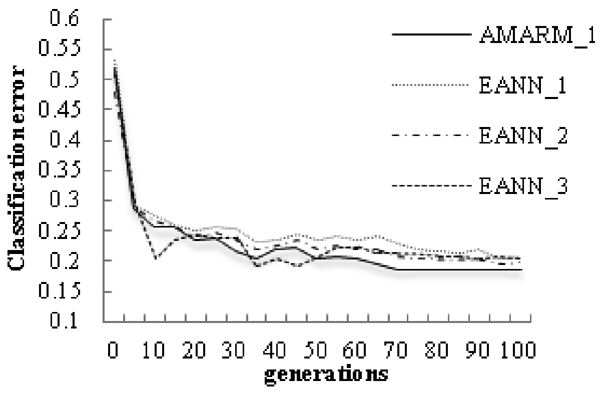
shown in the Table 1. The input attribute values of all problems are normalized within 0 and 1 using a linear function. The output attributes of all problems are encoded using a *1-of-m* output representation for the *m* classes. We employ the *logistic* and *softmax* activation functions for hidden and output layers, respectively. The output class is obtained using the winner-take-all strategy. Table 2 shows the setting of parameters of the proposed algorithm. These values are set based on preliminary runs of the proposed algorithm, they are not meant to be the optimal values.

B. EFFECT OF MULTI-LOCAL SEARCHES

We first examine the significance of multi-local searches in the proposed algorithm. For this purpose, we implement a set of experiments and compare the proposed algorithm (AMARM) with its three variants: AMARM incorporates no multi-local searches (denoted as AMARM_1), AMARM incorporates the BP algorithm for partial training (denoted as AMARM_2) and the iteration number of partial training is set to be 20, AMARM incorporates partial training, neuron merging and neuron splitting as the local



(*a*)



(*b*)

FIGURE 2. Comparison of the classification error on validation set between AMARM_1 and three variants of EANN on two classification problems: (*a*) card data and (*b*) diabetes data.

searches (denoted as AMARM_3). To make a fair comparison, AMARM and its variants are compared using the same parameter settings.

Table 3 shows the results of AMARM and its variants on six benchmark classification problems. The results clearly show the significance of multi-local searches in the proposed algorithm. The average TER achieved by AMARM is lower than that of its variants on the six problems to be tested. For example, on the Wine problem, the AMARM delivers an average TER of 2.56 while the three variants (AMARM_1, AMARM_2 and AMARM_3) give 3.64, 3.31 and 3.17, respectively. The AMARM_1 can easily be trapped into local optima, with high TER values. The AMARM_2, which incorporates the BP algorithm for partial training to optimize the connection weights, helps to locate a near optimal ANN. However, the partial training may cause the noisy fitness evaluation problem. As a result, the AMARM_2 has a worse performance in terms of the TER than AMARM. For the AMARM_3, although it can generally achieve lower TERs on the six problems compared with the other two AMARM variants, this algorithm still suffers from the noisy evaluation problem on diabetes data set, thus resulting in the worst TER.

Moreover, it can also be observed that AMARM is efficient. AMARM_1, incorporating no local searches, takes a large amount of time to locate the optima in a region of convergence. Both AMARM_2 and AMARM_3 are faster than AMARM_1. However, employing one single local search in AMARM_2 and multiple local searches without cooperation in AMARM_3 is not well suited for optimizing the ANN architectures. By employing neuron merging or neuron splitting followed by the one iteration of BP algorithm as local searches, the AMARM is generally able to quickly identify proper solutions.

C. EFFECT OF RANK BASED MUTATION

Then, we evaluate the impact of rank based mutation in the proposed algorithm. For this purpose, we perform a set of experiments and compare the AMARM without local searches (denoted as AMARM_1) with three variants of evolutionary artificial neural network (EANN): EANN_1 with a fixed mutation probability of 0.01, EANN_2 with a fixed mutation probability of 0.03 and EANN_3 with fixed mutation probability of 0.05. The Card and Diabetes problems, which have different degree of difficulty for classification, are used as the examples for testing. The same parameter settings are used for all algorithms for comparison.

Fig. 2 shows the average classification error of AMARM_1 and the three EANN variants on the Card and Diabetes problems. During the run of four algorithms, it can be observed that the classification error of solutions of all four algorithms reduces significantly at the early stage of evolution. However, EANN_1 and EANN_2 can be easily trapped

TABLE 4. The parameter settings of the six methods to be compared.

Method	Description	Value
EPNet	Mutation rate	0.03
	Learning rate	0.1
	Momentum rate	0.8
HEANN	Local mutation rate	0.02
	Local step size of weight mutation rate	2
	Global mutation rate	0.02
	Global step size of weight mutation rate	2
MGNN-ep	Range of scheduled mutation rate	[0.01, 0.05]
HGAPSO	Mutation rate in GA	0.1
	Crossover rate in GA	0.5
	Learning parameters c1, c2 in PSO	1, 1
	Control parameter χ in PSO	0.8
XLCC	Crossover rate	0.1
	No. of sub-populations	2
	No. of individuals in each sub-population	50
	Local search interval	1
	Local search intensity	10
	No. of individuals in local search population	10
DE-BP	Learning rate	0.1
	Momentum rate	0.8
	Mutation rate	0.01
	Crossover rate	0.1

TABLE 5. Comparison of AMARM, EPNet, HEANN, MGNN-ep HGAPSO, XLCC, and DE-BP on six benchmark classification problems.

Problems	Algorithm	Connections (Std. Dev.)	Epochs (Std. Dev.)	TER (Std. Dev.)
Iris	AMARM	**26.42(7.08)**	**6637.5(364.5)**	1.64(1.16)
	EPNet	37.42(6.14)	22345.2(829.1)	2.77(2.30)
	HEANN	30.36(8.75)	17336.1(851.6)	**1.17(1.19)**
	MGNN-ep	43.72(8.19)	37860.3(939.8)	3.68(1.80)
	HGAPSO	30.81(5.92)	26731.4(852.1)	3.26(1.90)
	XLCC	30.94(6.10)	19375.2(758.6)	2.41(2.11)
	DE-BP	29.8(5.40)	22351.9(732.4)	2.33(1.44)
Wine	AMARM	70.75(10.99)	**9937.5(495.5)**	**2.56(1.06)**
	EPNet	89.48(11.46)	84214.0(819.7)	5.04(1.30)
	HEANN	**55.42(11.10)**	12838.3(817.5)	3.06(1.18)
	MGNN-ep	129.70(32.19)	41920.4(991.7)	3.68(1.21)
	HGAPSO	106.2(13.1)	31821.1(812.9)	3.14(1.15)
	XLCC	82.1(17.1)	23566.0(735.5)	2.88(1.32)
	DE-BP	79.6(12.2)	33497.5(814.0)	3.09(1.10)
Card	AMARM	69.55(11.91)	**5772.5(410.0)**	**13.79(1.67)**
	EPNet	90.25(13.10)	59512.1(1031.5)	16.52(1.30)
	HEANN	**66.40(11.73)**	24941.5(831.6)	14.27(0.84)
	MGNN-ep	113.82(22.51)	42153.1(872.2)	14.94(1.24)
	HGAPSO	93.17(18.32)	39213.8(891.0)	15.73(1.03)
	XLCC	86.44(13.27)	26884.2(736.2)	14.05(1.10)
	DE-BP	89.40(14.90)	31294.7(812.9)	14.67(1.08)
Diabetes	AMARM	80.23(11.84)	**12231.0(483.5)**	**21.19(1.07)**
	EPNet	73.85(13.37)	81689.2(946.3)	24.57(1.90)
	HEANN	**67.56(11.32)**	40404.0(835.5)	21.33(1.68)
	MGNN-ep	91.80(11.92)	56436.9(875.9)	23.06(1.21)
	HGAPSO	121.80(14.84)	48198.1(873.8)	23.89(1.99)
	XLCC	98.43(13.18)	31967.6(732.4)	21.96(1.22)
	DE-BP	92.70(12.53)	42367.6(824.1)	23.19(1.16)
Blood	AMARM	33.29(4.46)	**4956.0(451.0)**	**12.75(0.80)**
	EPNet	41.86(3.73)	62456.1(892.9)	13.44(0.90)
	HEANN	**22.04(7.26)**	20318.0(818.5)	13.84(0.86)
	MGNN-ep	31.29(5.91)	38525.1(856.3)	14.23(0.81)
	HGAPSO	42.31(4.91)	35982.9(813.5)	13.98(0.91)
	XLCC	39.56(5.11)	24619.2(768.8)	13.82(1.13)
	DE-BP	38.16(5.11)	31851.5(806.4)	13.56(1.33)
Glass	AMARM	**85.74(10.73)**	**9890.2(655.5)**	**44.45(3.23)**
	EPNet	102.90(16.61)	93625.4(918.5)	52.12(2.93)
	HEANN	91.17(11.70)	51328.1(883.1)	55.14(3.08)
	MGNN-ep	114.1(12.20)	58319.7(894.8)	49.1(3.32)
	HGAPSO	119.90(13.80)	51948.2(903.8)	46.9(3.91)
	XLCC	98.68(12.79)	35973.9(762.5)	53.21(4.03)
	DE-BP	102.90(11.20)	41574.0(765.9)	50.39(3.68)

into local optima. EANN_3 can explore the solution space more effectively than EANN_1 and EANN_2 but fail to focus on promising optima due to the large mutation probability. By employing rank based mutation, AMARM_1 can achieve the best performance among the four algorithms. During the evolutionary process of AMARM_1, different mutation probabilities are applied to different individuals depending on their fitness values and ranks. The best individual has a small mutation probability for exploiting the local optima, while the worst individual is assigned with the largest mutation probability to explore the space. As a result, the solution space could be appropriately searched, thus avoid premature convergence. Similar results can also be found on the rest of data sets.

D. COMPARISON WITH RELATED WORK

Finally, we compare the proposed algorithm with related methods (i.e., EPNet [17], HEANN [21], MGNN-ep [18], HGAPSO [45], XLCC [66] and DE-BP [59]), which use different strategies for designing ANN architectures. Before discussing the comparative results, we first briefly describe the methods to be compared. The EPNet [17] is based on EP, which focuses on evolving ANNs' behaviors. In this method, five mutation operators as well as partial training are employed for closing behavioral links between parents and their offspring. The HEANN is based on a hybrid evolutionary algorithm, which emphasizes on the balance of global and local search during evolution. This method introduces a novel mutation technique for adapting the mutation probability as well as the step size of weight perturbation. The MGNN-ep is an "invasive" approach that is based on the EP with a scheduled mutation probability for evolving ANNs. The HGAPSO is a hybrid method that combines GA with particle swarm optimization (PSO). In this method, a GA is firstly applied to reproduce solutions individuals, and then the upper-half of best individuals are regarded as elites and further enhanced by PSO. The XLCC is also a hybrid method, in which a crossover-based local search is incorporated with a cooperative coevolution framework to train ANNs. In this method, the local search is applied according to a user-specified intensity for evolving a population called local search population and the best individual in which will be transferred to the sub-populations in cooperative coevolution after every user-specific local search interval. The DE-BP, on the other hand, combines the DE with BP algorithm for optimizing ANNs. In this method, the DE is first employed to identify promising regions of the search space. Then, the BP algorithm is applied to move solutions towards the optima.

To make the comparison fair and meaningful, the same population size (i.e., 100) and termination criterion is used for all methods to be compared. Other parameter values of the six methods are specified or chosen according to the original source papers for the best performance. The details of the parameter settings are listed in Table 4.

The results are summarized in Table 5, which show that the AMARM generally outperforms related methods to be compared. Compared with HGAPSO, XLCC and DE-BP, AMARM has a significantly better performance in terms of the TER, network complexity and number of epochs on all data sets. For example, on the Blood problem, the AMARM takes 4956.0 epochs by average to deliver ANNs with an average TER and network complexity of 12.75 and 33.29, respectively. By contrast, the XLCC requires 24619.2 epochs to obtain ANNs with an average TER and network complexity of 13.82 and 39.56, respectively. Compared with EPNet and MGNN-ep, AMARM also has a better performance

TABLE 6. Comparison of AMARM with ODB and VNP on six benchmark classification problems.

Problems	Algorithm	No. of hidden neurons	TER
Iris	AMARM	2.67	**1.64**
	OBD	4.20	1.91
	VNP	**2.25**	2.18
Wine	AMARM	6.93	**2.56**
	OBD	7.06	5.03
	VNP	**5.72**	5.26
Card	AMARM	6.88	**13.79**
	OBD	6.32	16.21
	VNP	**4.51**	15.79
Diabetes	AMARM	8.23	**21.19**
	OBD	13.4	32.30
	VNP	**8.09**	31.51
Blood	AMARM	3.82	**12.75**
	OBD	5.74	15.47
	VNP	**3.44**	14.83
Glass	AMARM	**8.72**	**44.45**
	OBD	10.21	60.21
	VNP	9.13	58.37

on all data sets, except the EPNet on Diabetes data and MGNN-ep on Blood data in term of network complexity. Compared with HEANN, AMARM achieves lower TERs on five out of six data sets to be tested and can be more efficient to do so. The better performance of AMARM is mainly due to the employment of multi-local search and rank based mutation mechanisms.

Moreover, two classical pruning based methods for ANN architecture design, i.e., Optimal Brain Damage (ODB) [12] and Variance Nullity Pruning (VNP) [13], have been considered for comparison. The results are reported in Table 6, which show that AMARM significantly outperforms ODB and VNP in term of average TERs on all six problems. For instance, the AMARM achieves an average TER of 21.19 on the Diabetes problem while the OBD and VNP give 32.30 and 31.51, respectively. In term of average number of hidden neurons, AMARM can deliver more compact architectures than ODB and has comparable performance to VNP. In term of time efficiency, AMARM requires far more training time than ODB and VNP.

VI. CONCLUSIONS

This paper presents an adaptive memetic algorithm with rank based mutation for designing well-generalized ANN architectures. In the proposed algorithm, we present multi-local searches, each has different search characteristics, to simultaneously and complementally fine-tune the number of hidden neurons and connection weights. Further, an adaptive strategy is devised to promote competition and corporation among the local searches for exploiting the search space. In addition, a rank based mutation strategy is introduced to avoid premature convergence. Extensive experiments have been carried out to evaluate the performance of proposed algorithm and compared with related methods. The results show that our method is capable of delivering compact ANN architectures with good generalization ability and generally outperforms related methods to be compared.

There are several directions in which this work can be extended further. Firstly, it is desirable to dynamically control the parameters of AMARM during evolution to improve its performance further. Secondly, it would also be interesting to employ and test other adaptation strategies such as subproblem decomposition [55] and biased roulette wheel [56] for selecting and applying multiple local searches. In addition, a population of ANNs contains more information than a single ANN, applying the proposed algorithm to evolve neural network ensembles could also be carried out in the future.

REFERENCES

[1] X. Ma and X. Gan, "Condition monitoring and faults recognizing of dish centrifugal separator by artifical neural network combined with expert system," in *Proc. 5th Int. Conf. Natural Comput.*, vol. 2. Aug. 2009, pp. 203–207.

[2] A. Quteishat, C. P. Lim, and K. S. Tan, "A modified fuzzy min–max neural network with a genetic-algorithm-based rule extractor for pattern classification," *IEEE Trans. Syst., Man, Cybern. A, Syst., Humans*, vol. 40, no. 3, pp. 641–650, May 2010.

[3] H. Altun, A. Bilgil, and B. C. Fidan, "Treatment of multi-dimensional data to enhance neural network estimators in regression problems," *Expert Syst. Appl.*, vol. 32, no. 2, pp. 599–605, 2007.

[4] W. He, A. O. David, Z. Yin, and C. Sun, "Neural network control of a robotic manipulator with input deadzone and output constraint," *IEEE Trans. Syst., Man, Cybern., Syst.*, vol. 46, no. 6, pp. 759–770, Jun. 2016.

[5] D. Wang, D. Liu, Q. Zhang, and D. Zhao, "Data-based adaptive critic designs for nonlinear robust optimal control with uncertain dynamics," *IEEE Trans. Syst., Man, Cybern., Syst.*, vol. 46, no. 11, pp. 1544–1555, Nov. 2016.

[6] M. Chen, P. Shi, and C.-C. Lim, "Adaptive neural fault-tolerant control of a 3-DOF model helicopter system," *IEEE Trans. Syst., Man, Cybern., Syst.*, vol. 46, no. 2, pp. 260–270, Feb. 2016.

[7] P. D. Heermann and N. Khazenie, "Classification of multispectral remote sensing data using a back-propagation neural network," *IEEE Trans. Geosci. Remote Sens.*, vol. 30, no. 1, pp. 81–88, Jan. 1992.

[8] M. Gori and A. Tesi, "On the problem of local minima in backpropagation," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 14, no. 1, pp. 76–86, Jan. 1992.

[9] M. A. Sattar, M. M. Islam, and K. Murase, "A new constructive algorithm for designing and training artificial neural networks," in *Proc. Int. Conf. Neural Inf. Process.*, 2007, pp. 317-327.

[10] M. M. Islam, M. A. Sattar, M. F. Amin, X. Yao, and K. Murase, "A new constructive algorithm for architectural and functional adaptation of artificial neural networks," *IEEE Trans. Syst. Man, Cybern. B, Cybern.*, vol. 39, no. 6, pp. 1590–1605, Dec. 2009.

[11] J. L. Subirats, L. Franco, and J. M. Jerez, "C-Mantec: A novel constructive neural network algorithm incorporating competition between neurons," *Neural Netw.*, vol. 26, no. 2, pp. 130–140, Feb. 2012.

[12] Y. Le Cun, J. S. Denker, and S. A. Solla, "Optimal brain damage," in *Proc. NIPS*, 1989, pp. 598–605.

[13] A. P. Engelbrecht, "A new pruning heuristic based on variance analysis of sensitivity information," *IEEE Trans. Neural Netw.*, vol. 12, no. 6, pp. 1386–1399, Nov. 2001.

[14] P. Lauret, E. Fock, and T. A. Mara, "A node pruning algorithm based on a Fourier amplitude sensitivity test method," *IEEE Trans. Neural Netw.*, vol. 17, no. 2, pp. 273–293, Mar. 2006.

[15] T.-Y. Kwok and D.-Y. Yeung, "Constructive algorithms for structure learning in feedforward neural networks for regression problems," *IEEE Trans. Neural Netw.*, vol. 8, no. 3, pp. 630–645, May 1997.

[16] A. Che, P. Wu, F. Chu, and M. C. Zhou, "Improved quantum-inspired evolutionary algorithm for large-size lane reservation," *IEEE Trans. Syst., Man, Cybern., Syst.*, vol. 45, no. 12, pp. 1535–1548, Dec. 2015.

[17] X. Yao and Y. Liu, "A new evolutionary system for evolving artificial neural networks," *IEEE Trans. Neural Netw.*, vol. 8, no. 3, pp. 694–713, May 1997.

[18] P. P. Palmes, T. Hayasaka, and S. Usui, "Mutation-based genetic neural network," *IEEE Trans. Neural Netw.*, vol. 16, no. 3, pp. 587–600, May. 2005.

[19] N. García-Pedrajas, C. Hervás-Martínez, and J. Muñoz-Pérez, "COVNET: A cooperative coevolutionary model for evolving artificial neural networks," *IEEE Trans. Neural Netw.*, vol. 14, no. 3, pp. 575–596, May 2003.

[20] C.-Y. Lee and X. Yao, "Evolutionary programming using mutations based on the Levy probability distribution," *IEEE Trans. Evol. Comput.*, vol. 8, no. 1, pp. 1–13, Feb. 2004.

[21] T. H. Ong and N. A. M. Isa, "Adaptive evolutionary artificial neural networks for pattern classification," *IEEE Trans. Neural Netw.*, vol. 22, no. 11, pp. 1823–1836, Nov. 2011.

[22] X. Chen, Y.-S. Ong, M.-H. Lim, and K. C. Tan, "A multi-facet survey on memetic computation," *IEEE Trans. Evol. Comput.*, vol. 15, no. 5, pp. 591–607, Oct. 2011.

[23] Y.-S. Ong, M.-H. Lim, N. Zhu, and K.-W. Wong, "Classification of adaptive memetic algorithms: A comparative study," *IEEE Trans. Syst., Man, Cybern. B, Cybern.*, vol. 36, no. 1, pp. 141–152, Feb. 2006.

[24] P. Rakshit *et al.*, "Realization of an adaptive memetic algorithm using differential evolution and Q-learning: A case study in multirobot path planning," *IEEE Trans. Syst., Man, Cybern., Syst.*, vol. 43, no. 4, pp. 814–831, Jul. 2013.

[25] W. E. Hart, N. Krasnogor, and J. E. Smith, "Memetic evolutionary algorithms," in *Recent Advances in Memetic Algorithms* (Studies in Fuzziness and Soft Computing). Berlin, Germany: Springer, 2005, pp. 3–27.

[26] M. Srinivas and L. M. Patnaik, "Adaptive probabilities of crossover and mutation in genetic algorithms," *IEEE Trans. Syst., Man, Cybern.*, vol. 24, no. 4, pp. 656–667, Apr. 1994.

[27] H. Lu and G. G. Yen, "Rank-density-based multiobjective genetic algorithm and benchmark test function study," *IEEE Trans. Evol. Comput.*, vol. 7, no. 4, pp. 325–343, Aug. 2003.

[28] P. S. Oliveto, P. K. Lehre, and F. Neumann, "Theoretical analysis of rank-based mutation—Combining exploration and exploitation," in *Proc. IEEE Congr. Evol. Comput.*, May 2009, pp. 1455–1462.

[29] P. Shan and W. Sheng, "An adaptive memetic algorithm for designing artificial neural networks," in *Proc. IEEE 12th Int. Conf. Ubiquitous Intell. Comput.*, Aug. 2015, pp. 320–323.

[30] M. Frean, "The upstart algorithm: A method for constructing and training feedforward neural networks," *Neural Comput.*, vol. 2, no. 2, pp. 198–209, 1990.

[31] R. Parekh, J. Yang, and V. Honavar, "Constructive neural-network learning algorithms for pattern classification," *IEEE Trans. Neural Netw.*, vol. 11, no. 2, pp. 436–451, Mar. 2000.

[32] R. Reed, "Pruning algorithms—A survey," *IEEE Trans. Neural Netw.*, vol. 4, no. 5, pp. 740–747, Sep. 1993.

[33] M. M. Islam, M. A. Sattar, M. F. Amin, X. Yao, and K. Murase, "A new adaptive merging and growing algorithm for designing artificial neural networks," *IEEE Trans. Syst., Man, Cybern. B, Cybern.*, vol. 39, no. 3, pp. 705–722, Jun. 2009.

[34] S.-H. Yang and Y.-P. Chen, "An evolutionary constructive and pruning algorithm for artificial neural networks and its prediction applications," *Neurocomputing*, vol. 86, no. 4, pp. 140–149, 2012.

[35] X. Yao, "Evolving artificial neural networks," *Proc. IEEE*, vol. 87, no. 9, pp. 1423–1447, Sep. 1999.

[36] E. Cantu-Paz and C. Kamath, "An empirical comparison of combinations of evolutionary algorithms and neural networks for classification problems," *IEEE Trans. Syst., Man, Cybern. B, Cybern.*, vol. 35, no. 5, pp. 915–927, Oct. 2005.

[37] A. Abraham, "Meta learning evolutionary artificial neural networks," *Neurocomputing*, vol. 56, pp. 1–38, Jan. 2003.

[38] J. Yu, S. Wang, and L. Xi, "Evolving artificial neural networks using an improved PSO and DPSO," *Neurocomputing*, vol. 71, nos. 4–6, pp. 1054–1060, 2008.

[39] C. Lu, B. Shi, and L. Chen, "Hybrid BP-GA for multilayer feedforward neural networks," in *Proc. 7th IEEE Int. Conf. Electron., Circuits Syst. (ICECS)*, Dec. 2000, pp. 958–961.

[40] N. Y. Nikolaev and H. Iba, "Learning polynomial feedforward neural networks by genetic programming and backpropagation," *IEEE Trans. Neural Netw.*, vol. 14, no. 2, pp. 337–350, Mar. 2003.

[41] A. C. Martinez-Estudillo, C. Hervas-Martinez, F. J. Martinez-Estudillo, and N. Garcia-Pedrajas, "Hybridization of evolutionary algorithms and local search by means of a clustering method," *IEEE Trans. Syst., Man, Cybern. B, Cybern.*, vol. 36, no. 3, pp. 534–545, Jun. 2005.

[42] J.-T. Tsai, J.-H. Chou, and T.-K. Liu, "Tuning the structure and parameters of a neural network by using hybrid Taguchi-genetic algorithm," *IEEE Trans. Neural Netw.*, vol. 17, no. 1, pp. 69–80, Jan. 2006.

[43] F. H. F. Leung, H. K. Lam, S. H. Ling, and P. K. S. Tam, "Tuning of the structure and parameters of a neural network using an improved genetic algorithm," *IEEE Trans. Neural Netw.*, vol. 14, no. 1, pp. 79–88, Jan. 2003.

[44] L. M. Almeida and T. B. Ludermir, "A multi-objective memetic and hybrid methodology for optimizing the parameters and performance of artificial neural networks," *Neurocomputing*, vol. 73, nos. 7–9, pp. 1438–1450, 2010.

[45] C.-F. Juang, "A hybrid of genetic algorithm and particle swarm optimization for recurrent network design," *IEEE Trans. Syst., Man, Cybern. B, Cybern.*, vol. 34, no. 2, pp. 997–1006, Apr. 2004.

[46] A. V. Kononova, K. J. Hughes, M. Pourkashanian, and D. B. Ingham, "Fitness Diversity Based Adaptive Memetic Algorithm for solving inverse problems of chemical kinetics," in *Proc. IEEE Congr. Evol. Comput.*, Sep. 2007, pp. 2366–2373.

[47] A. Caponio, G. L. Cascella, F. Neri, N. Salvatore, and M. Sumner, "A fast adaptive memetic algorithm for online and offline control design of PMSM drives," *IEEE Trans. Syst., Man, Cybern. B, Cybern.*, vol. 37, no. 1, pp. 28–41, Feb. 2007.

[48] W. Sheng, S. Chen, M. Fairhurst, G. Xiao, and J. Mao, "Multilocal search and adaptive niching based memetic algorithm with a consensus criterion for data clustering," *IEEE Trans. Evol. Comput.*, vol. 18, no. 5, pp. 721–741, Oct. 2014.

[49] A. P. Wieland, "Evolving neural network controllers for unstable systems," in *Proc. Int. Joint Conf. Neural Netw. (IJCNN-Seattle)*, vol. 2. Jul. 1991, pp. 667–673.

[50] J.-Y. Lin and Y.-P. Chen, "Analysis on the collaboration between global search and local search in memetic computation," *IEEE Trans. Evol. Comput.*, vol. 15, no. 5, pp. 608–623, Oct. 2011.

[51] M. M. Islam, X. Yao, and K. Murase, "A constructive algorithm for training cooperative neural network ensembles," *IEEE Trans. Neural Netw.*, vol. 14, no. 4, pp. 820–834, Jul. 2003.

[52] C. Zanchettin, T. B. Ludermir, and L. M. Almeida, "Hybrid training method for MLP: Optimization of architecture and training," *IEEE Trans. Syst. Man, Cybern. B, Cybern.*, vol. 41, no. 4, pp. 1097–1109, Aug. 2011.

[53] A. Asuncion and D. Newman. (2007). "UCI machine learning repository," School Inf. Comput. Sci., Univ. California, Irvine, Irvine, CA, USA, Tech. Rep. [Online]. Available: http://archive.ics.uci.edu/ml

[54] L. Prechelt, "A quantitative study of experimental evaluations of neural network learning algorithms: Current research practice," *Neural Netw.*, vol. 9, no. 3, pp. 457–462, 1996.

[55] Y. S. Ong, "Artificial intelligence technologies in complex engineering design," Ph.D. dissertation, Univ. Southampton, Southampton, U.K., 2002.

[56] Y. S. Ong and A. J. Keane, "Meta-Lamarckian learning in memetic algorithms," *IEEE Trans. Evol. Comput.*, vol. 8, no. 2, pp. 99–110, Apr. 2004.

[57] C.-T. Lin, M. Prasad, and A. Saxena, "An improved polynomial neural network classifier using real-coded genetic algorithm," *IEEE Trans. Syst., Man, Cybern., Syst.*, vol. 45, no. 11, pp. 1389–1401, Nov. 2015.

[58] S.-K. Oh, W. Pedrycz, and B.-J. Park, "Self-organizing neurofuzzy networks based on evolutionary fuzzy granulation," *IEEE Trans. Syst., Man, Cybern. A, Syst., Humans*, vol. 33, no. 2, pp. 271–277, Mar. 2003.

[59] P. P. Sarangi, A. Sahu, and M. Panda, "A hybrid differential evolution and back-propagation algorithm for feedforward neural network training," *Int. J. Comput. Appl.*, vol. 14, pp. 1–9, Jan. 2013.

[60] L. Zhang, H. Li, and D. Feng, *ODE-LM: A Hybrid Training Algorithm for Feedforward Neural Networks*. Berlin, Germany: Springer, 2014, pp. 986–995.

[61] M. Yaghini, M. M. Khoshraftar, and M. Fallahi, "A hybrid algorithm for artificial neural network training," *Eng. Appl. Artif. Intell.*, vol. 26, no. 1, pp. 293–301, 2013.

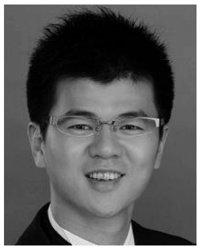
[62] S. Nandy, P. P. Sarkar, and A. Das, "Training a feed-forward neural network with artificial bee colony based backpropagation method," *Int. J. Comput. Sci. Inf. Technol.*, vol. 4, no. 4, pp. 652–665, 2014.

[63] K. Socha and C. Blum, "An ant colony optimization algorithm for continuous optimization: Application to feed-forward neural network training," *Neural Comput. Appl.*, vol. 16, no. 3, pp. 235–247, 2007.

[64] B. A. Garro and R. A. Vázquez, "Designing artificial neural networks using particle swarm optimization algorithms," *Comput. Intell. Neurosci.*, vol. 2015, Jan. 2015, Art. no. 369298.

[65] M. J. Dinneen and K. Wei, "A (1+1) adaptive memetic algorithm for the maximum clique problem," in *Proc. IEEE Congr. Evol. Comput.*, Jun. 2013, pp. 1626–1634.

[66] R. Chandra, M. Frean, and M. Zhang, "Crossover-based local search in cooperative co-evolutionary feedforward neural networks," *Appl. Soft Comput.*, vol. 12, no. 9, pp. 2924–2932, 2012.



WEIGUO SHENG received the M.Sc. degree in information technology from the University of Nottingham, U.K., in 2002, and the Ph.D. degree in computer science from Brunel University, U.K., in 2005. He was a Researcher with the University of Kent, U.K., and Royal Holloway, University of London, U.K. He is currently a Professor with Hangzhou Normal University. His research interests include evolutionary computation, data mining/clustering, pattern recognition, and machine learning.

PENGXIAO SHAN received the B.Sc. degree in software engineering and the M.Sc. degree in information technology from the Zhejiang University of Technology, Hangzhou, China, in 2013 and 2016, respectively. He is currently with Zhejiang University of Technology, Hangzhou, China and China Merchants Bank, Hangzhou, China. His main research interests include neural network, evolutionary computation, and data mining/classification.

JIAFA MAO received the Ph.D. degree in pattern recognition from the East China University of Science and Technology, China, in 2009. Since 2009, he has been a Post-Doctoral Researcher with the Beijing University of Posts and Telecommunications, China. In 2011, he joined the Zhejiang University of Technology, China, where he is currently an Associate Professor with the School of Computer Science and Technology. He has published over 30 papers in the scientific literature. His research interests include pattern recognition, digital image processing, and information hiding.

YUJUN ZHENG (M'06) received the Ph.D. degree from the Institute of Software, Chinese Academy of Sciences, in 2010. He is currently an Associate Professor and the Ph.D. Advisor with Hangzhou Normal University. He has published over 60 scientific papers in journals, including the IEEE TRANSACTIONS ON EVOLUTIONARY COMPUTATION, the IEEE TRANSACTIONS ON FUZZY SYSTEMS, and the IEEE TRANSACTIONS ON INTELLIGENT TRANSPORTATION SYSTEMS. His research interests include nature-inspired computation and its applications. He is a member of the ACM. He received the 2014 IFORS Prize for the development (runner-up) due to his work on intelligent scheduling of emergency engineering rescue in China.

SHENGYONG CHEN (M'01–SM'10) received the Ph.D. degree in computer vision from the City University of Hong Kong, Hong Kong, in 2003. He joined the Zhejiang University of Technology, China, in 2004, where he is currently a Professor with the Department of Computer Science. He was with the University of Hamburg from 2006 to 2007. He has published over 100 scientific papers in international journals and conferences. His research interests include computer vision, robotics, and image analysis. He is a fellow of IET and a Committee Member of the IET Shanghai Branch. He received the National Outstanding Youth Foundation Award of China in 2013. He received a fellowship from the Alexander von Humboldt Foundation of Germany.

ZIDONG WANG (SM'03–F'14) was born in Jiangsu, China, in 1966. He received the B.Sc. degree in mathematics from Suzhou University, Suzhou, China, in 1986, and the M.Sc. degree in applied mathematics and the Ph.D. degree in electrical engineering from the Nanjing University of Science and Technology, Nanjing, China, in 1990 and 1994, respectively.

From 1990 to 2002, he held teaching and research appointments in universities in China, Germany, and U.K. He is currently a Professor of dynamical systems and computing with the Department of Information Systems and Computing, Brunel University London, U.K. He has published over 300 papers in refereed international journals. His research interests include dynamical systems, signal processing, bioinformatics, and control theory and applications. He is a holder of the Alexander von Humboldt Research Fellowship of Germany, the JSPS Research Fellowship of Japan, and the William Mong Visiting Research Fellowship of Hong Kong.

Dr. Wang is a fellow of the Royal Statistical Society and a member of the program committee for many international conferences. He serves (or has served) as an Editor-in-Chief of *Neurocomputing* and an Associate Editor for 12 international journals, including the IEEE TRANSACTIONS ON AUTOMATIC CONTROL, the IEEE TRANSACTIONS ON CONTROL SYSTEMS TECHNOLOGY, the IEEE TRANSACTIONS ON NEURAL NETWORKS, the IEEE TRANSACTIONS ON SIGNAL PROCESSING, and the IEEE TRANSACTIONS ON SYSTEMS, MAN, AND CYBERNETICS–PART C.