An Improved Genetic Algorithm with Average-Bound Crossover and Wavelet Mutation Operations

S.H. Ling and F.H.F. Leung

Centre for Multimedia Signal Processing, Department of Electronic and Information Engineering,

The Hong Kong Polytechnic University, Hung Hom, Kowloon, Hong Kong

Abstract: This paper presents a real-coded genetic algorithm (RCGA) with new genetic operations (crossover and mutation). They are called the average-bound crossover (ABX) and wavelet mutation (WM). By introducing the proposed genetic operations, both the solution quality and stability are better than the RCGA with conventional genetic operations. A suite of benchmark test functions are used to evaluate the performance of the proposed algorithm. Application examples on economic load dispatch and tuning an associative-memory neural network are used to show the performance of the proposed RCGA.

Keywords: Crossover, mutation, real-coded genetic algorithm, associative-memory neural network, and economic load dispatch.

I. INTRODUCTION

Genetic algorithm (GA) is one evolutionary computation technique [9] that can tackle complex optimization problems [9, 17, 23]. It has been applied in different areas such as fuzzy control [13-14], tuning of neural or neural fuzzy network [15-16], path planning [11], greenhouse climate control [1], economic load dispatch [2, 27], etc. Traditional binary GA [5, 9, 19, 25] has some drawbacks when applying to multidimensional and high-precision numerical problems. The situation can be improved if GA in real numbers is used. Each chromosome is coded as a vector of floating point numbers that has the same length as the solution vector. A large domain can thus be handled. Much research effort has been spent to improve the performance of real-coded GA (RCGA). In general, RCGA involves three operations: selection, crossover and mutation. The selection operation is used to select the chromosomes from the population with respect to some probability distribution based on fitness values. The crossover operation is used to combine the information of the selected chromosomes (parents) and generate the offspring. The mutation

operation is used to change the offspring genes. Selection schemes such as rank-based selection, elitist strategies, steady-state election and tournament selection were reported [5]. Recently, different crossover operations for RCGA have been proposed to improve the efficiency of the algorithm. The extended intermediate recombination (crossover) (EIX) was proposed by Mühlenbein *et. al.* [20]. The genes (variables) of the offspring are chosen somewhere between the genes of the parents. It is capable of producing any point within a hypercube slightly larger than that defined by the parents. The unimodal normal distribution crossover (UNDX) was proposed by Ono *et. al.* [12, 22] for handling multimodal functions and non-separability problems. UNDX mixes the parental information and shows a good searching ability. However, it changes the fundamental concept that the crossover operation should combine the parents to generate offspring, not mixing the parents. The blend crossover (BLX-α) was proposed by Eshelman *et. al.* [7], which combines the parents to reproduce offspring. It shows a good searching ability for separable functions. However, BLX-α has difficulty in handling non-separability optimisation problems. Also, the above crossover operations are not suitable for optimisation problems with the optimal point located near the domain boundary. For mutation operations, the uniform mutation and non-uniform mutation can be found [19, 21]. The uniform mutation is to change the value of a randomly selected gene to a value between its upper and lower bounds. The non-uniform mutation is capable of fine-tuning the parameters by increasing or decreasing the value of a randomly selected gene with respect to a weighted random number. The weight is usually a monotonic decreasing function of the number of iteration.

In this paper, new genetic operations of crossover and mutation are proposed. The crossover operation is called the average-bound crossover (ABX), which combines the average crossover and bound crossover. The average crossover manipulates the genes of the selected parents, the minimum, and the maximum possible values of the genes. The bound crossover is capable of moving the offspring near the domain boundary. On realizing the ABX operation, the offspring spreads over the domain so that a higher chance of reaching the global optimum can be obtained. The proposed mutation operation is called the wavelet mutation (WM), which applies the wavelet theory [3-4, 18] to realize the mutation. Wavelet is a tool to model seismic signals by combining dilations and translations of a simple, oscillatory function (mother wavelet) of a finite duration. The wavelet function has two properties: 1) the function integrates to zero, and 2) it is square integrable, or equivalently has finite energy. Thanks to the properties of the wavelet, the convergence and solution stability are improved. By introducing these genetic operations, the RCGA performs more efficiently and provides a faster convergence than the RCGA with conventional genetic operations in a suite of 18 benchmark test functions [6, 8, 24, 30]. In addition, the RCGA with the proposed operations gives smaller standard deviations of results, i.e. the

solution quality of the RCGA with the proposed operations is more stable. An experimental study will be made to evaluate the searching ability of the proposed mutation. Also, the sensitivity of the parameter in WM and the sensitivity of the genes' initial range for the proposed RCGA to the searching performance will be discussed. Application examples on economic load dispatch and tuning an associative-memory neural network are also given to show the performance of the proposed RCGA.

This paper is organized as follows. Section II presents the operation of the proposed genetic operations. Experimental studies and analysis are discussed in Section III. 18 benchmark test functions will be used to evaluate the performance of the proposed method. Application examples on economic load dispatch and tuning an associative memory neural network are given in Section IV. A conclusion will be drawn in Section V.

II. Average-Bound Crossover and Wavelet Mutation for RCGA

The Real-Coded Genetic Algorithm (RCGA) process [5, 19, 25] is shown in Fig. 1. First, a set of population of chromosomes P is created. Each chromosome **p** contains some genes (variables). Second, the chromosomes are evaluated by a defined fitness function. The better chromosomes will return higher fitness function values in this process. Third, some of the chromosomes are selected to undergo genetic operations for reproduction by the method of normalized geometric ranking [10]. Normalized geometric ranking is a selection based on a non-stationary penalty function, which is a function of the generation number. As the number of generation increases, the penalty increases that puts more and more selective pressure on the RCGA to find the feasible solution. In general, a higher-rank chromosome will have a higher chance to be selected. Fourth, genetic operations of crossover are performed. The crossover operation is mainly for exchanging information between two parents that are obtained by the selection operation. In the crossover operation, one of the parameters is the probability of crossover p_c which gives the expected number $p_c \times pop\_size$ (where pop_size is the number of chromosomes in the population) of chromosomes that undergo the crossover operation in a generation. We propose a new crossover operation here. First, four chromosomes are generated (instead of two chromosomes in the conventional RCGA) from two selected parents. Second, the best two offspring in terms of the fitness value will be selected to replace their parents. After the crossover operation, the mutation operation follows. It operates with a parameter called the probability of mutation (p_m). The mutation operation is to change the genes of the chromosomes in the population such that the features inherited from their parents can be changed. After going through the mutation operation,

the new offspring will be evaluated using the fitness function. The new population will be formed when the new offspring replaces the chromosome with the smallest fitness value. After the operations of selection, crossover and mutation, a new population is generated. This new population will repeat the same process. Such an iterative process will be terminated when a defined condition is met. The details about the proposed crossover and mutation operations are given below.

A. Average-bound crossover operation

The crossover operation is mainly for exchanging information from the two parents, chromosomes $\mathbf{p}_1$ and $\mathbf{p}_2$, obtained in the selection process. The two parents will eventually produce two offspring. The average-bound crossover (ABX) comprises two operations: average crossover and bound crossover.

1) Average crossover

$$\mathbf{o}_{\mathbf{s}_c^1} = \frac{(\mathbf{p}_1 + \mathbf{p}_2)}{2} \tag{1}$$

$$\mathbf{o}_{\mathbf{s}_c^2} = \frac{(\mathbf{p}_{\max} + \mathbf{p}_{\min})(1 - w_a) + (\mathbf{p}_1 + \mathbf{p}_2) w_a}{2} \tag{2}$$

2) Bound crossover

$$\mathbf{o}_{\mathbf{s}_c^3} = \mathbf{p}_{\max}(1 - w_b) + \max(\mathbf{p}_1, \mathbf{p}_2) w_b \tag{3}$$

$$\mathbf{o}_{\mathbf{s}_c^4} = \mathbf{p}_{\min}(1 - w_b) + \min(\mathbf{p}_1, \mathbf{p}_2) w_b \tag{4}$$

where

$\mathbf{o}_{\mathbf{s}_c^k} = \left[o_{s_1^k} \quad o_{s_2^k} \quad \cdots \quad o_{s_{no\_vars}^k} \right]$, k = 1, 2, 3, 4.

$$\mathbf{p}_i = \left[p_{i_1} \quad p_{i_2} \quad \cdots \quad p_{i_j} \quad \cdots \quad p_{i_{\text{no\_vars}}} \right], \; i = 1, 2; \; j = 1, 2, \ldots, no\_vars, \tag{5}$$

$$para_{\min}^j \le p_{i_j} \le para_{\max}^j, \tag{6}$$

$$\mathbf{p}_{\max} = \left[para_{\max}^1 \quad para_{\max}^2 \quad \cdots \quad para_{\max}^{no\_vars} \right], \tag{7}$$

$$\mathbf{p}_{\min} = \left[para_{\min}^1 \quad para_{\min}^2 \quad \cdots \quad para_{\min}^{no\_vars} \right], \tag{8}$$

no_vars denotes the number of variables to be tuned; $para_{\min}^j$ and $para_{\max}^j$ are the minimum and maximum values of p_{i_j} respectively for all i; $w_a, w_b \in [0 \quad 1]$ denotes the user-defined weight for average crossover and bound crossover respectively, $\max(\mathbf{p}_1, \mathbf{p}_2)$ denotes the vector with each element obtained by taking the maximum between the corresponding element of $\mathbf{p}_1$ and $\mathbf{p}_2$. For

instance, $\max([1 \quad -2 \quad 3],[2 \quad 3 \quad 1]) = [2 \quad 3 \quad 3]$. Similarly, $\min(\mathbf{p}_1, \mathbf{p}_2)$ gives a vector by taking the minimum value. For instance, $\min([1 \quad -2 \quad 3],[2 \quad 3 \quad 1]) = [1 \quad -2 \quad 1]$. Among $\mathbf{o}_{\mathbf{s}_c^1}$ to $\mathbf{o}_{\mathbf{s}_c^4}$, the two with the largest fitness values are used as the offspring of the crossover operation. These two offspring are put back into the population to replace their parents.

The rationale behind the ABX is that if the offspring spreads over the domain, a higher chance of reaching the global optimum can be obtained. As seen from (1) to (4): The average crossover will move the offspring near the centre region of the concerned domain (as w_a in (2) approaches 1, $\mathbf{o}_{\mathbf{s}_c^2}$ approaches $(\mathbf{p}_1 + \mathbf{p}_2)/2$, which is the average of the selected parents; and as w_a approaches 0, $\mathbf{o}_{\mathbf{s}_c^2}$ approaches $(\mathbf{p}_{\max} + \mathbf{p}_{\min})/2$, which is the average of the domain boundary), while bound crossover will move the offspring near the domain boundary (as w_b in (3) and (4) approaches 0, $\mathbf{o}_{\mathbf{s}_c^3}$ and $\mathbf{o}_{\mathbf{s}_c^4}$ approaches $\mathbf{p}_{\max}$ and $\mathbf{p}_{\min}$ respectively). The result of the crossover depends on the values of the weights w_a and w_b. Their values depend on the optimisation problem and are chosen by trial and error. Fig. 2 shows an example indicating the relationship between the parents and the offspring under different values of the weights. In this figure, the line represents the domain of a gene. The end points of the line represent the minimum and maximum values of the gene. The dot (•) represents the parents and the circle-dot (⊙) represents the offspring. The values in brackets represent the values of the genes under different values of the weights. For example, when $p_1 = 1$ and $p_2 = 4$, referring to (1) and (2), the offspring $\mathbf{o}_{\mathbf{s}_c^1}$ and $\mathbf{o}_{\mathbf{s}_c^2}$ should be equal to 2.5 and 3.75 respectively when $w_a = 0.5$. According to (1) to (4), the offspring is generated. We can see how the offspring spreads over the domain under different values of w_a and w_b. Changing the value of the weight w_a will change the characteristics of the average crossover operation. In this paper, the value of w_a is arbitrarily set at 0.5. On the other hand, changing the value of the weight w_b will change the characteristics of the bound crossover operation.

B. Wavelet mutation operation

Before presenting the wavelet mutation operation, we first discuss the basic wavelet theory.

1) Wavelet theory

Certain seismic signals can be modelled by combining translations and dilations of an oscillatory function with a finite duration called a “wavelet”. A continuous-time function $\psi(x)$ is called a “mother wavelet” or “wavelet” if it satisfies the following properties:

Property 1:

$$\int_{-\infty}^{+\infty}\psi(x)dx = 0 \tag{9}$$

In other words, the total positive momentum of $\psi(x)$ is equal to the total negative momentum of $\psi(x)$.

Property 2:

$$\int_{-\infty}^{+\infty}\left|\psi(x)\right|^2 dx < \infty \tag{10}$$

where most of the energy in $\psi(x)$ is confined to a finite duration and bounded. The Morlet wavelet (as shown in Fig .3) is an example mother wavelet, which was proposed by Daubechies [4]:

$$\psi(x) = e^{-x^2/2}\cos(5x) \tag{11}$$

The Morlet wavelet integrates to zero (Property 1). Over 99% of the total energy of the function is contained in the interval of $-2.5 \le x \le 2.5$ (Property 2).

In order to control the magnitude and the position of $\psi(x)$, we define $\psi_{a,b}(x)$ as follows:

$$\psi_{a,b}(x) = \frac{1}{\sqrt{a}}\psi\left(\frac{x-b}{a}\right) \tag{12}$$

where *a* is the dilation parameter and *b* is the translation parameter. Notice that

$$\psi_{1,0}(x) = \psi(x) \tag{13}$$

As

$$\psi_{a,0}(x) = \frac{1}{\sqrt{a}}\psi\left(\frac{x}{a}\right), \tag{14}$$

it follows that $\psi_{a,0}(x)$ is an amplitude-scaled version of $\psi(x)$. Fig. 4 shows different dilations of the Morlet wavelet. The amplitude of $\psi_{a,0}(x)$ will be scaled down as the dilation parameter *a* increases. This property is used to do the mutation operation in order to enhance the searching performance.

2) Wavelet mutation

The mutation operation is to change the genes of the chromosomes inherited from their parents. In general, various methods like uniform mutation or non-uniform mutation [19, 21] can be employed to realize the mutation operation. We propose a Wavelet Mutation (WM) operation based on the wavelet theory, which exhibits a fine-tuning ability. The details of the operation are as follows. Every gene of the chromosomes will have a chance to mutate governed by a probability of mutation, $p_m \in [0 \quad 1]$, which is defined by the user. This probability gives an expected number ($p_m \times pop\_size \times no\_vars$) of genes that undergo the mutation. For each gene, a random number

between 0 and 1 will be generated such that if it is less than or equal to p_m, the mutation will take place on that gene which is updated instantly. If $\mathbf{o_s} = \left[o_{s_1}, \ o_{s_2}, \ \ldots \ , o_{s_{no\_vars}}\right]$ is the selected chromosome and the element o_{s_j} is randomly selected for mutation (the value of o_{s_j} is inside $[\ para^j_{\min}, para^j_{\max}\ ]$), the resulting chromosome is given by $\hat{\mathbf{o}}_\mathbf{s} = \left[o_{s_1}, \ \ldots \ , \hat{o}_{s_j}, \ \ldots \ , o_{s_{no\_vars}}\right]$, where $j \in 1, 2, \ldots no\_vars$, and

$$\hat{o}_{s_j} = \begin{cases} o_{s_j} + \delta \times \left(para^j_{\max} - o_{s_j}\right) \text{ if } \delta > 0 \\ o_{s_j} + \delta \times \left(o_{s_j} - para^j_{\min}\right) \text{ if } \delta \le 0 \end{cases}, \tag{15}$$

$$\delta = \psi_{a,0}(\varphi) \tag{16}$$

$$\delta = \frac{1}{\sqrt{a}} \psi\left(\frac{\varphi}{a}\right) \tag{17}$$

By using the Morlet wavelet in (11) as the mother wavelet,

$$\delta = \frac{1}{\sqrt{a}} e^{-\left(\frac{\varphi}{a}\right)^2 / 2} \cos\left(5\left(\frac{\varphi}{a}\right)\right) \tag{18}$$

where $\varphi \in [-2.5, 2.5]$ is randomly generated. If δ is positive ($\delta > 0$) approaching 1, the mutated gene will tend to the maximum value of o_{s_j}. Conversely, when δ is negative ($\delta \le 0$) approaching −1, the mutated gene will tend to the minimum value of o_{s_j}. A larger value of $|\delta|$ gives a larger searching space for o_{s_j}. When $|\delta|$ is small, it gives a smaller searching space for fine-tuning the gene. Referring to Property 1 of the wavelet, the sum of the positive δ is equal to the sum of the negative δ when the number of samples is large and φ is randomly generated. That is,

$$\frac{1}{N} \sum_N \delta = 0 \text{ for } N \to \infty, \tag{19}$$

where N is the number of samples.

Hence, the overall positive mutation and the overall negative mutation throughout the evolution are nearly the same. This property gives better solution stability (smaller standard deviation of the solution values upon many trials). As over 99% of the total energy of the mother wavelet function is contained in the interval [−2.5, 2.5], φ can be generated from [−2.5, 2.5] randomly. The value of the dilation parameter a can be set to vary with the value of $\frac{\tau}{T}$ in order to meet the fine-tuning purpose, where T is the total number of iteration and τ is the current number of iteration. In order to perform a local search when τ is large, the value of a should increase as $\frac{\tau}{T}$ increases so as to

reduce the significance of the mutation. Hence, a monotonic increasing function governing a and $\frac{\tau}{T}$ is proposed as follows.

$$a = e^{-\ln(g)\times\left(1-\frac{\tau}{T}\right)^{\zeta}+\ln(g)} \tag{20}$$

where ζ is the shape parameter of the monotonic increasing function, g is the upper limit of the parameter a. In this paper, g is set as 10000. The effects of the various values of the shape parameter ζ to a with respect to $\frac{\tau}{T}$ are shown in Fig. 5. The value of a is between 1 and 10000. Referring to (18), the maximum value of δ is 1 when the random number of φ=0 and a=1 ($\frac{\tau}{T}=0$). Then referring to (15), the offspring gene $\hat{o}_{s_j} = o_{s_j} + 1\times\left(para^{j}_{\max} - o_{s_j}\right) = para^{j}_{\max}$. It ensures that a large search space for the mutated gene is given. When the value $\frac{\tau}{T}$ is near to 1, the value of a is so large that the maximum value of δ will become very small. For example, at $\frac{\tau}{T}=0.9$ and $\zeta=1$, the dilation parameter a = 4000. If the random value of φ is zero, the value of δ will be equal to 0.0158. With $\hat{o}_{s_j} = o_{s_j} + 0.0158\times\left(para^{j}_{\max} - o_{s_j}\right)$, a small searching space for the mutated gene is given for fine-tuning.

C. Choosing the parameters

We can regard the RCGA is seeking a balance between the exploration of new regions and the exploitation of the already sampled regions in the search space. This balance, which critically controls the performance of the RCGA, is governed by the right choices of the control parameters: the probability of crossover (p_c), the probability of mutation (p_m), the population size (*pop_size*), the weights of the proposed crossover (w_a, w_b) and the shape parameter ζ of WM. Some views about these parameters are included as follows:

- The probability of crossover (p_c) gives us an expected number ($p_c \times pop\_size$) of chromosomes which undergo the crossover operation in a generation. When p_c = 1, all chromosomes in a generation will undergo the crossover operation.
- Increasing the probability of mutation (p_m) tends to transform the genetic search into a random search. This probability gives us an expected number ($p_m \times pop\_size \times no\_vars$) of genes that undergo the mutation. When $p_m = 1$, all genes will mutate. The value of p_m depends on the desired number of genes that undergo the mutation operation.

- Increasing the population size will increase the diversity of the search space, and reduce the probability that GA will prematurely converge to a local optimum. However, it also increases the time required for the population to converge to the optimal region in the search space.
- Changing the value of the weight w_a in the average-bounded crossover will change the characteristics of the average crossover operations. It is chosen by trial and error, which depends on the kind of the optimisation problem. As the value of w_a tends to 1, the offspring tends to be the average of the selected parents. As the value of w_a tends to 0, the offspring tends to be the average of the domain boundary. For many optimisation problems, the value of the weight w_a can be set as 0.5.
- Changing the value of the weight w_b in the average-bound crossover will change the characteristics of the bound crossover operations. It is also chosen by trial and error, which depends on the kind of the optimisation problem. A value of w_b approaching 1 will make the offspring to be near the selected parents. As the value of w_b tends to 0, the offspring will become near the domain boundary.
- Changing the parameter ζ will change the characteristics of the monotonic increasing function of the wavelet mutation. The dilation parameter a will take a value so as to perform fine-tuning faster as ζ is increasing. It is chosen by trial and error, which depends on the kind of the optimisation problem. When ζ becomes larger, the decreasing speed of the step size (δ) of the mutation becomes faster. In general, if the optimisation problem is smooth and symmetric, the searching algorithm is easier to find the solution and process the fine-tuning in early iteration. Thus, a larger value of ζ can be used to increase the step size of the early mutation. More details about the sensitivity of ζ to WM will be discussed in the next section.

III. Experimental Studies and Analysis

A. Benchmark test function

A suite of 18 benchmark test functions [6, 8, 24, 30] are used to test the performance of the RCGA with the proposed genetic operations. Many different kinds of optimization problems are covered by these benchmark test functions. They are divided into three categories: unimodal functions, multimodal functions with only a few local minima, and multimodal functions with many local minima. The 18 benchmark test functions are detailed in Appendix A. They can test

the searching ability of the proposed searching algorithm comprehensively. To avoid the proposed crossover operation introducing a strong bias to the optimal location at $(\mathbf{p}_{\max} + \mathbf{p}_{\min})/2$, the ranges of the domain boundary for some test functions are set different from those in [6, 8, 24, 30]. Functions f_1 to f_7 are unimodal functions. Functions f_8 to f_{13} are multimodal functions with only a few local minima. Functions f_{14} to f_{18} are multimodal functions with many local minima.

B. Experimental setup

The crossover operation for comparison is the UNDXBXover, which consists of two published crossover operations: Unimodal normal distribution crossover (UNDX) [12, 22] and Blend crossover (BLX-α) [7]. The details of these two crossovers are shown in Appendix B and Appendix C respectively. The mutation operation for comparison is the non-uniform mutation (NUM) [19, 21]. The details of NUM are shown in Appendix D. The simulation conditions are described as follows.

- The shape parameter of NUM: It is chosen by trial and error through experiments for good performance for all functions.
- The parameters ζ of WM: It is chosen by trial and error through experiments for good performance for all functions.
- The weight of the ABX w_a: 0.5 for all functions.
- The weight of the ABX w_b: 0.5 for f_1 to f_8 and f_{15} to f_{17}; 1.0 for f_9 to f_{14}, and f_{18}.
- Population size: 100.
- Number of runs: 50.
- Selection operation: Normalized geometric ranking [10].
- The probability of selecting the best chromosome [10]: 0.08.
- Crossover operation: For UNDX, the parameters β and μ are set at 1 and 0.35 respectively; for BLX-α, the parameter α is set at 0.336 [26].
- Probability of crossover p_c: 0.8.
- Probability of mutation p_m: 0.5 for f_1 to f_6 and f_{14} to f_{18}; 0.8 for f_7 to f_{13}.
- Initial population: It is generated uniformly at random.

In this paper, RCGA with Avergae-Bound Crossover and Wavelet Mutation (ABX+WM), RCGA with Avergae-Bound Crossover and Non-Uniform Mutation (ABX+NUM), RCGA with Unimodal Normal Distribution and Blend Crossover and Wavelet Mutation, (UNDXBXover+WM), and RCGA with Unimodal Normal Distribution and Blend crossover and Non-Uniform Mutation (UNDXBXover+NUM) are used to test the benchmark test functions.

C. Experiment results

1. Unimodel Functions

Functions f_1 to f_6 are unimodal functions. The experiment results in terms of the mean cost value, best cost value, standard deviation, and the *t*-test value for f_1 to f_6 are tabulated in Table. I. The comparison between different genetic operations on f_1 to f_6 is shown in Fig. 6. The *t*-test is a statistical method to evaluate the significant difference between two algorithms. The *t*-value will be negative if the first algorithm is better than the second, and positive if it is poorer. When the *t*-value is smaller than −1.645 (degree of freedom = 49), there is a significant difference between the two algorithms with a 95% confidence level. Function f_1 is a sphere model which is probably the most widely used test function. It is smooth and symmetric. The performance on this function is a measure of the convergence rate of a searching algorithm. For f_1, the results in terms of the mean and the best cost value of ABX with WM or NUM are better than those of the corresponding UNDXBXover. Comparing ABX with WM to UNDXBXover with WM, the mean cost value is 2.5 times better. A much smaller standard deviation is given by the ABX+WM, which means the solution is more stable. Comparing the mutation operations WM and NUM, the proposed WM is more effective than NUM in term of the cost value and standard deviation. Both the solution quality and stability offered by WM are better than those offered by NUM. In addition, the *t* value of −10.62 implies that the improved genetic operations (AveXover with WM) are better than the conventional genetic operations (UNDXBXover with NUM). In Fig. 6, ABX with WM displays a faster convergence rate than UNDXBXover with NUM thanks to its better searching ability. It reaches approximately 0.01 in around 250 times of iteration, while it is about 3.0 for UNDXBXover with NUM. Function f_2 is a generalized Rosenbrock's function which is strongly non-separable and the optimum is located in a very narrow ridge. The tip of the ridge is very sharp, and it runs around a parabola. Algorithms that are unable to discover good searching directions will perform poorly in this problem. The proposed genetic operations (ABX with WM) outperforms the UNDXBXover with NUM. The *t* value is −313.3. Although the best cost values on using WM with different crossover operations are a bit worse than those on using NUM, the mean value, standard deviation and convergence rate offered by WM are better. Function f_3 is a step function that is a representative of flat surfaces. Flat surfaces are obstacles for optimization algorithms because they do not give any information about the search direction. Unless the algorithm has a variable step size, it can get stuck in one of the flat surfaces. UNDXBXover performs poorly for f_3 because it mainly searches in a small local neighbourhood, but the flat

surfaces do not give any searching direction for UNDXBXover. On the other hand, the proposed ABX is good for f_3 because it can generate longer jump than UNDXBXover. Comparing WM to NUM with UNDXBXover, the former also gives a better solution. Function f_4 is a quartic function, which is a simple unimodal function padded with noise. The Gaussian noise causes the algorithm never getting the same value at the same point. Many algorithms that do not do well in this function are due to the noisy data. In this function, the mean cost value, best cost value, standard deviation and the convergence rate brought by the proposed ABX and WM are significantly better than the conventional genetic operations. Function f_5 is the Schwefel's problem 2.21, function f_6 is the Schwefel's problem 2.22 and function f_7 is the Eason's function. In these problems, the performance on using the proposed crossover ABX and mutation WM is better than that on using the UNDXBXover and NUM. The rapid convergence of the proposed genetic operations shown in Fig. 6 supports our argument. In short, the proposed genetic operations (ABX and WM) are good to tackle unimodal functions/problems when compared with the conventional genetic operations (UNDXBXover and NUM). Both the solution quality and stability are satisfactory.

2. Multimodel functions with a few local minima

Functions f_8 to f_{13} are multimodal functions with only a few local minima. The experimental results for f_8 to f_{13} are tabulated in Table II. Fig. 7 shows the average values for f_8 to f_{13}. Among these functions, five of them (f_8, f_{10} - f_{13}) do not show statistically significant differences for different genetic operations. They all reach or get near to the global optima. For the function f_9, we obtain statistically different results from the proposed genetic operations and the conventional genetic operations. The proposed ABX performs better than the UNDXBXover in terms of the mean, best value, standard deviation and the convergence rate. In addition, the results offered by WM are better than those by NUM in terms of the mean and the best cost values. Furthermore, WM gives a faster convergence rate.

3. Multimodel functions with many local minima

Functions f_{14} to f_{18} are multimodal functions with many local minima, and the dimension of each function is comparatively larger than that of f_8 to f_{13}. The dimension of these functions is 30. The experimental results for f_{14} to f_{18} are tabulated in Table III. The comparison between different genetic operations is shown in Fig. 8. It can be seen from Table III that the mean results and the best results offered by the proposed genetic operations (ABX and WM) are better than

those offered by the conventional genetic operations (UNDXBXover and NUM). Also, they have smaller standard deviations. Therefore, in terms of the solution quality and stability, the proposed genetic operations are better than the conventional operations. In addition, the *t*-test value of all functions is smaller than –1.645. Therefore, the proposed genetic operations are significantly better than the conventional operations for solving the optimization problems. From Fig. 8, we can see that the convergence rate offered by the proposed genetic operations is better than that offered by the conventional genetic operations.

D. The searching ability of wavelet mutation

In this section, we give an analysis based on experimental results to illustrate that the searching ability of WM is better than that of NUM. The experimental settings are the same as before, except the probability of crossover is set at 0 and the probability of mutation is set at 1. By using this setting, no chromosomes will undergo the crossover operation, and all genes in the population will mutate under the mutation operation. Hence, the searching ability of the mutation operation can be evaluated. The experimental results on using WM and NUM for f_1 to f_{18} (except f_3 and f_7) without crossover operation are summarized in Table IV. The comparison between WM and NUM is given in Fig. 9 and Fig. 10. Function f_3 and f_7 are not included in this experiment because they do not perform well with mutation operation only. As seen from Table IV, the average performance of WM is better than NUM. WM gives smaller standard deviations of results for all test functions than NUM, and hence the solution stability offered by WM is better. From Fig. 9 and Fig. 10, the convergence of WM is found faster than that of NUM. In conclusion, the searching ability of WM is better than NUM.

E. Sensitivity of the parameter for wavelet mutation

The mean cost values offered by WM using different shape parameter ζ for all test functions are tabulated in Table V. As can be seen from the table, all functions are tested by using ζ =0.2, ζ =0.5, ζ =1, ζ =2, and ζ =5. If the optimization problem needs a more significant mutation to reach the optimal point, a smaller ζ should be given. Conversely, if the RCGA needs to perform the fine-tuning faster, a larger ζ should be used. For example, f_1 is a sphere model which is smooth and symmetric. Searching algorithms are fast to jump to the area near the global optimum and then perform fine-tuning. Therefore, a larger ζ is set (ζ =5) so that the RCGA will go to perform fine-tuning faster. On the other hand, ζ is set as 0.2 for f_{13} when the mutation operation is playing a significant role at the later stage. In some cases, ζ's value is not very critical, e.g. f_3 and f_{11}. For f_3, the mean cost value for different ζ is the same. We say that the best

performance is obtained when ζ=0.5 because the standard deviation of the RCGA for ζ=0.5 is the smallest. However, in some cases, the parameter ζ is so sensitive as to affect the performance of the searching, e.g. f_1 and f_{16}. In conclusion, no formal method is available to choose the parameter ζ; it depends on the characteristics of the optimization problems.

F. Sensitivity of the initial range of variables

Additional experiments are carried out to test the sensitivity of the initial range of the variables to the RCGA with the improved genetic operations. The settings of these experiments are exactly the same as before (section III B). The experiment results for f_4 are tabulated in Table VI. Fig. 11 shows the results for different genetic operations on f_4. The initial population is generated uniformly at random in the ranges of $-2.56 \le x_i \le 5.12$ (twice the original range), $-6.4 \le x_i \le 12.8$ (5 times of the original range), $-12.8 \le x_i \le 25.6$ (10 times of the original range), and $-25.6 \le x_i \le 51.2$ (20 times of the original range), making the average distance to the global optimum increasingly large. The enlarged searching space is expected to make the problem more difficult to solve. As can be seen from the table and the figures, the mean cost values offered by the proposed genetic operations are better than those by the conventional genetic operations. From Fig. 11, ABX and WM offer faster convergence than UNDXBXover and NUM. In addition, ABX and WM give smaller standard deviation for all initial ranges than UNDXBXover and NUM. Hence, the solution quality is more stable. Two more test functions are then used to test the sensitivity to the initial range of variables. The experiment results for f_7 and f_{16} are tabulated in Table VII to VIII respectively. Fig. 12 and Fig. 13 show the results for different genetic operations on f_7 and f_{16} respectively. In these tables and figures, the results of the improved genetic operations in terms of the mean cost value, convergence rate, and standard deviation are better than those of the conventional algorithms.

IV. APPLICATION EXAMPLES

Application examples on economic load dispatch and tuning of associative memory are given in this section.

A. Economic load dispatch

In a power system, minimizing the operation cost is important. Economic load dispatch (ELD) is a method to schedule power generator outputs with respect to the load demands, and to operate a power system economically. The input-output characteristics of modern generators are

nonlinear by nature because of the valve-point loadings and rate limits. The problem of ELD is multimodal, discontinuous and highly nonlinear. RCGAs had been employed to solve the ELD problems [2, 27].

1. Mathematic modelling of economic load dispatch with valve-point loading

The economic load dispatch with valve-point loading problem can be formulated into the following objective function:

$$Min\sum_{i=1}^{n} C_i\left(P_{L_i}\right), \tag{21}$$

where $C_i\left(P_{L_i}\right)$ is the operation fuel cost of generator *i*, and *n* denotes the number of generators. The problem is subject to a balance constraint and generating capacity constraints as follows:

$$D = \sum_{i=1}^{n} P_{L_i} - P_{Loss}\ , \tag{22}$$

$$P_{L_{i,\min}} \le P_{L_i} \le P_{L_{i,\max}}\ ,\ i = 1, 2, \ldots, n. \tag{23}$$

where D is the load demand, P_{L_i} is the output power of the *i*-th generator, P_{Loss} is the transmission loss, $P_{L_{i,\max}}$ and $P_{L_{i,\min}}$ are the maximum and minimum output powers of the *i*-th generator respectively. The operation fuel cost function with valve-point loadings of the generators is given by,

$$C_i\left(P_{L_i}\right) = a_i P_{L_i}^2 + b_i P_{L_i} + c_i + \left| e_i \times \sin\left(f_i \times \left(P_{L_{i,\min}} - P_{L_i}\right)\right)\right|\ , \tag{24}$$

where a_i , b_i , and c_i are coefficients of the cost curve of the *i*-th generator, e_i and f_i are coefficients of the valve-point loadings. （The generating units with multivalve steam turbines exhibit a great variation in the fuel-cost functions. The valve-point effects introduce ripples in the heat-rate curves.)

RCGA can be used to solve the economic load dispatch problem. The chromosomes **p** is defined as follows:

$$\mathbf{p} = \left[P_{L_1} \quad P_{L_2} \quad P_{L_3} \quad \cdots \quad P_{L_{n-1}}\right]\ , \tag{25}$$

From (22), we have,

$$P_{L_n} = D - \sum_{i=1}^{n-1} P_{L_i} + P_{Loss}\ . \tag{26}$$

In this paper, the power loss is not considered. Therefore,

$$P_{L_n} = D - \sum_{i=1}^{n-1} P_{L_i}\ . \tag{27}$$

To ensure P_{L_n} falls within the range $\left[P_{L_{n,\min}}, P_{L_{n,\max}}\right]$, the following conditions are considered:

$$\text{if } P_{L_n} > P_{L_{n,\max}} \begin{cases} P_{L_1 New} = P_{L_1} + \left(P_{L_{n,\max}} - P_{L_n}\right) \\ P_{L_n} = P_{L_{n,\max}} \end{cases}, \tag{28}$$

$$\text{if } P_{L_n} < P_{L_{n,\min}} \begin{cases} P_{L_1 New} = P_{L_1} - \left(P_{L_n} - P_{L_{n,\min}}\right) \\ P_{L_n} = P_{L_{n,\min}} \end{cases}. \tag{29}$$

It should be noted from (28) and (29) that if the value of P_{L_1} is also outside the constraint boundaries. The exceeding portion of the power will further be shared by other generators in order to make sure that all generators' output power is within the safety range. Referring to (21), the fitness function for this ELD problem is defined as:

$$fitness = -\sum_{i=1}^{n} C_i\left(P_{L_i}\right), \tag{30}$$

where $C_i\left(P_{L_i}\right)$ is defined in (24). The objective is to maximize the fitness function (30).

2. Case Study

The RCGA with the proposed genetic operations and the RCGA with the conventional genetic operations are applied to a 40-generator system, which was adopted as an example in [2]. The system is a very large one with nonlinearities. The data of the units for this example with valve-point loadings are tabulated in Table IX. The load demand (*D*) is 10500MW. For comparison purpose, RCGA with ABX and WM, RCGA with ABX and NUM, RCGA with UNDXBXover and WM, and RCGA with UNDXBXover and NUM are used to solve the ELD problem. The population size used for all RCGAs is 100. All the simulation results are averaged ones out of 50 runs. For the proposed ABX, the parameters w_a and w_b are set at 0.5. For the UNDXBXover, the parameters β, μ, and α are set at 1, 0.35, and 0.336 respectively. For the proposed mutation WM, the parameter ζ is set at 1. For the NUM, the shape parameter is set at 1. The probabilities of crossover and mutation for all approaches are set at 0.6 by trial and error. For all approaches, the number of iteration is 1000. The statistical results are shown in Table X and Fig. 14. It can be seen that the RCGA with the proposed ABX and WM performs better than other RCGAs with conventional genetic operations (UNDXBXover and NUM) in terms of cost, *t* value, and standard deviation. Both the solution quality and stability are good. The average cost is \$122811.41 and the best (minimum) cost is \$121915.93. The optimal dispatch solution is summarized in Table XI.

B. Tuning associative memory

Learning or training is one of the important issues of neural networks. The learning process aims to find a set of optimal network parameters. The widely-used gradient methods [28, 31], such as MRI, MRII, MRIII rules, and back-propagation techniques, adjust the network parameters based on the gradient information of the fitness function in order to reduce the mean square error over all input patterns. One major weakness of the gradient methods is that the derivative information of the fitness function has to be known, meaning that the fitness function has to be continuous. Also, the learning process is easily trapped in a local optimum, especially when the problems are multimodal and the learning rules are network structure dependent. To tackle this problem, the real-code genetic algorithm (RCGA) [5, 19, 25], was proposed for the optimization problem in a large, complex, non-differentiable and multimodal domain [29]. RCGA is a good training algorithm for neural or neural-fuzzy networks [15-16]. The same RCGA can be used to train many different networks regardless of whether they are feed-forward one, recurrent one, associative memory or of other structure types. This generally saves a lot of human efforts in developing training algorithms for different types of networks.

Associative memory is one type of neural network that maps its input vector into itself. Thus, the desired output vector is its input vector. 50 input vectors are used for the learning. The function of the associative memory is given by:

$$y_k(t) = \sum_{j=1}^{10} w_{jk} z_j(t),\ k = 1, 2, \ldots, 10 \tag{31}$$

where $z(t)$ is the input vector and w_{jk} is the weight of the link between the i-th input and the k-th output. The objective is to minimize the mean square error (MSE), which is defined as follows:

$$\text{MSE} = \frac{\sum_{k=1}^{10}\sum_{t=1}^{50}(z_k(t) - y_k(t))^2}{10 \times 50} \tag{32}$$

The initial range of the weight w_{jk} is from −2 to 2. For comparison purpose, RCGA with ABX and WM, RCGA with ABX and NUM, RCGA with UNDXBXover and WM, and RCGA with UNDXBXover and NUM are used to solve this problem. The population size used for all RCGAs is 100. All the simulation results are averaged ones out of 50 runs. For the proposed ABX, the parameters w_a and w_b are set at 0.5 and 1 respectively. For the UNDXBXover, the parameters β, μ, and α are set at 1, 0.35, and 0.336 respectively. For the proposed mutation WM, the parameter ζ is set at 2. For the NUM, the shape parameter is set at 2. The probabilities of crossover and mutation for all approaches are set at 0.8 and 0.2 by trial and error. For all approaches, the number of iteration is 2000. The experimental results are tabulated in Table XII, and the comparison between different genetic operations is shown in Fig. 15. As can be seen from the table, the mean and the best cost value offered by ABX and WM are better. In addition, the smaller standard

deviation implies a more stable solution. The *t*-value for this function is −24.67, which is a relatively large figure. In short, the proposed RCGA is good for tuning associative memory.

V. Conclusion

An RCGA with improved genetic operations (average-bound crossover and wavelet mutation) has been presented. By using the proposed crossover operation, the offspring spreads over the domain so that the probability of reproducing good offspring is increased. In the proposed mutation operation, the wavelet theory is applied. Thanks to the properties of the wavelet, both the solution quality and stability are improved. A suit of benchmark test functions has been used to illustrate the merits of the improved genetic operations. Examples on economic load dispatch and tuning associative memory have also been given.

Acknowledgement

The work described in this paper was substantially supported by a grant from the Hong Kong Polytechnic University (PhD Student Account Code: RG9T).

References

[1] Caponetto R., Fortuna L., Nunnari G., Occhipinti L., and Xibilia M.G., "Soft computing for greenhouse climate control," *IEEE Trans., Fuzzy Systems*, vol. 8, no. 6, pp. 753-760, Dec. 2000.

[2] Chen P.H. and Chang H.C., "Large-scale economic dispatch by genetic algorithm," *IEEE Trans. Power Syst*., vol. 10, pp. 117-124, Feb. 1995.

[3] Daubechies I., "The wavelet transform, time-frequency localization and signal analysis," *IEEE Trans. Information Theory*, vol. 36, no.5, pp. 961-1005, Sep. 1990.

[4] Daubechies I., *Ten lectures on wavelets*. Philadelphia, PA: Society for Industrical and Applied Mathematics, 1992.

[5] Davis L., *Handbook of genetic algorithms.* NY: Van Nostrand Reinhold, 1991.

[6] De Jong K.A., "An analysis of the behavior of a class of genetic adaptive systems," *Ph.D. Thesis*, University of Michigan, Ann Arbor, MI, 1975.

[7] Eshelman L.J. and Schaffer J.D., "Real-coded genetic algorithms and interval-schemata," *Foundations of Genetic Algorithms 2*, pp. 187-202, 1993.

[8] Goldstein A.A. and Price I.F., "On descent from local minima," *Math. Comput.*, vol. 25, no. 115, 1971.

[9] Holland J.H., *Adaptation in natural and artificial systems*. University of Michigan Press, Ann Arbor, MI, 1975.

[10] Joines J. and Houck C., “On the use of non-stationary penalty functions to solve constrained optimization problems with genetic algorithm,” in *Proc. 1994 Int. Symp. Evolutionary Computation*, Ordando, 1994, pp. 579-584.

[11] Juidette H. and Youlal H., “Fuzzy dynamic path planning using genetic algorithms,” *Electronics Letters*, vol. 36, no. 4, pp. 374-376, Feb. 2000.

[12] Kita H., Ono I., and Kobayashi S., “Theoretical analysis of the unimodal normal distribution crossover for real-coded genetic algorithms,” in *Proc. of the Congress on Evolutionary Computational (CEC1998), World Congress on Computational Intelligence (WCCI 1998)*, May 4-9, 1998, pp. 529-534.

[13] Lam H.K., Leung F.H.F., and Tam P.K.S., “Design and stability analysis of fuzzy model based nonlinear controller for nonlinear systems using genetic algorithm,” *IEEE Trans. Syst., Man and Cybern, Part B: Cybernetics*, vol. 33, no. 2, pp. 250-257, Feb. 2003.

[14] Leung F.H.F., Lam H.K., Ling S.H., and Tam P.K.S., "Optimal and stable fuzzy controllers for nonlinear systems using an improved genetic algorithm," *IEEE Trans. Industrial Electronics*, vol. 51, no. 1, pp.172-182, Feb. 2004.

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

[16] Ling S.H., Leung F.H.F, Lam H.K., and Tam P.K.S., "Short-term electric load forecasting based on a neural fuzzy network," *IEEE Trans. Industrial Electronics*, vol. 50, no. 6, pp.1305-1316, Dec. 2003.

[17] Liu B.D., Chen C.Y., and Tsao J.Y., “Design of adaptive fuzzy logic controller based on linguistic-hedge concepts and genetic algorithms,” *IEEE Trans. Systems, Man and Cybernetics, Part B*, vol. 31 no. 1, pp. 32-53, Feb. 2001.

[18] Mallat S.G., “A theory for multiresolution signal decomposition: the wavelet representation,” *IEEE Trans. Pattern analysis and machine intelligence*, vol. 11, no.7, pp. 674-693, Jul. 1989.

[19] Michalewicz Z., *Genetic Algorithm + Data Structures = Evolution Programs*, 2nd extended ed. Springer-Verlag, 1994.

[20] Mühlenkein H. and Schlierkamp-Voosen D., “Predictive models for the breeder genetic algorithm I. continuous parameter optimization,” *Evolutionary Computation*, vol.1, no.1, pp. 25-49, 1993.

[21] Neubauer A., “A theoretical analysis of the non-uniform mutation operator for the modified genetic algorithm,” in *Proc. IEEE Int. Conf. Evolutionary Computation, 1997*, Indianapolis, pp. 93-96.

[22] Ono I. and Kobayashi S., “A real-coded genetic algorithm for function optimization using unimodal normal distribution crossover,” in *Proc. 7th ICGA*, 1997, pp. 246-253.

[23] Pham D.T. and Karaboga D., *Intelligent Optimization Techniques, Genetic Algorithms, Tabu Search, Simulated Annealing and Neural Networks*. Springer, 2000.

[24] Schwefel H.P., *Numerical optimization of computer models.* Chichester, Wiley & Sons, 1981.

[25] Srinivas M.and Patnaik L.M., "Genetic algorithms: a survey," *IEEE Computer*, vol. 27, issue 6, pp. 17-26, June 1994.

[26] Takahashi M. and Kita H., "A crossover operator using independent component analysis for real-coded genetic algorithms," in *Proc. of the Congress on Evolutionary Computation, (CEC2001),* 2001, pp.643-638.

[27] Walter D.C. and Sheble G.B., "Genetic algorithm solution of economic dispatch with valve point loading," *IEEE Trans. Power Syst.*, vol. 8, pp.1325-1332, Aug. 1993.

[28] Widrow B. and Lehr M.A., "30 years of adaptive neural networks: Perceptron, madaline, and backpropagation," *Proceedings of the IEEE,* vol. 78, no. 9, pp. 1415-1442, Sept. 1990.

[29] Yao X., "Evolving artificial networks," *Proceedings of the IEEE,* vol. 87, no. 7, pp. 1423-1447, 1999.

[30] Yao X. and Liu Y., "Evolutionary programming made faster," *IEEE Trans. Evolutionary Computation*, vol. 3, no. 2, pp. 82-102, July 1999.

[31] Zurada J.M., *Introduction to Artificial Neural Systems.* West Info Access, 1992.

```
Procedure of the RCGA
begin
    τ→0                    // τ : iteration number
    Initialize P(τ)        // P(τ) : population for iteration τ
    Evaluate f(P(τ))       // f(P(τ)) :fitness function
while (not termination condition) do
      begin
          τ→τ+1
          Select 2 parents p1 and p2 from P(τ–1)
          Perform crossover operation with pc
          Four chromosomes will be generated
          Select the best two offspring in terms of the fitness value
          Perform mutation operation with pm
          Reproduce a new P(τ)
          Evaluate f(P(τ))
      end
end
```

Fig. 1. RCGA process.

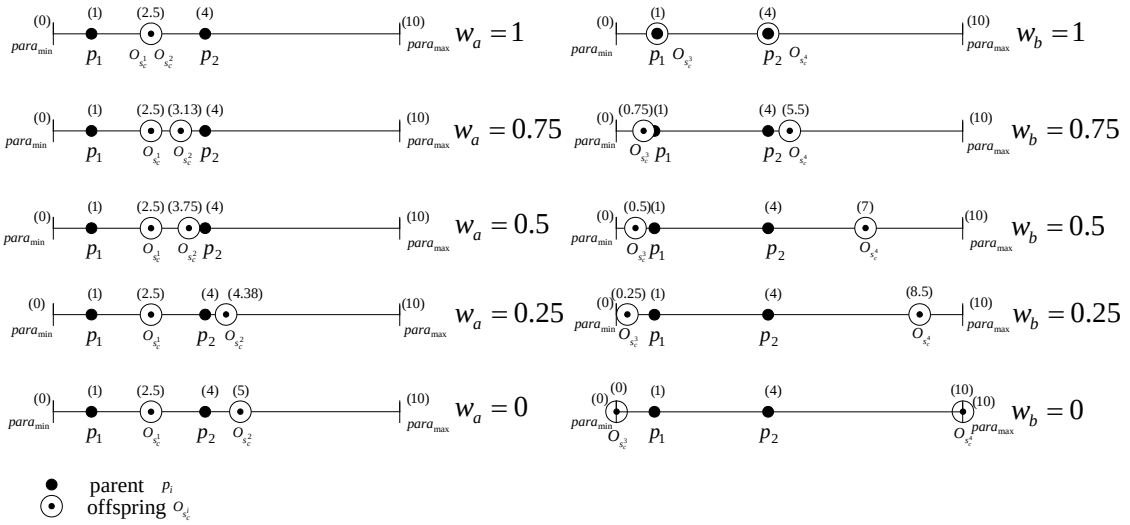


Fig. 2. Parents and offspring under different values of the weights w_a and w_b (w_a , w_b = 0, 0.25, 0.5, 0.75 and 1.)

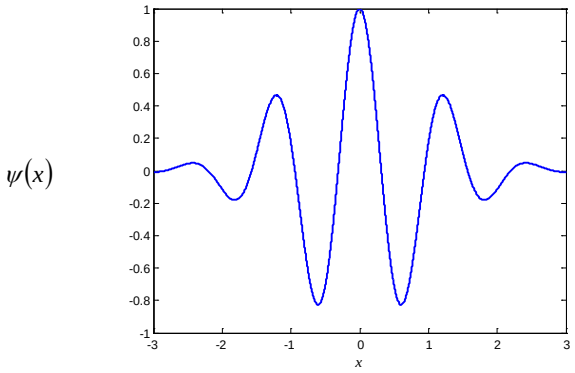


Fig. 3. The Morlet Wavelet.

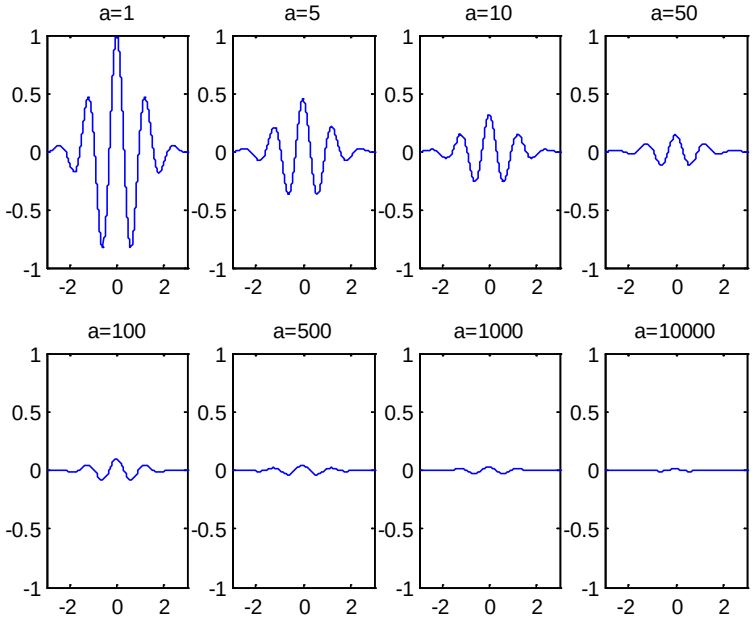


Fig. 4. A Morlet wavelet dilated by different values of the parameter a (x-axis: x, y-axis: $\psi_{a,0}(x)$.)

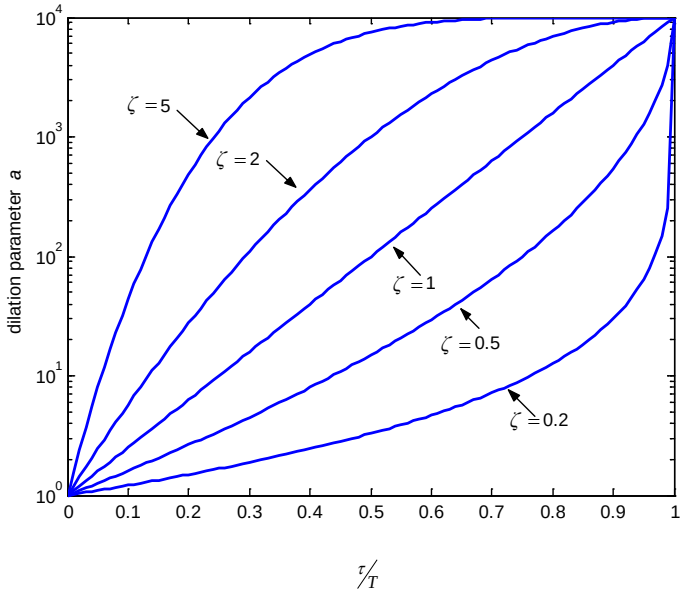


Fig. 5. The effect of the shape parameter ζ to a with respected to τ/T.

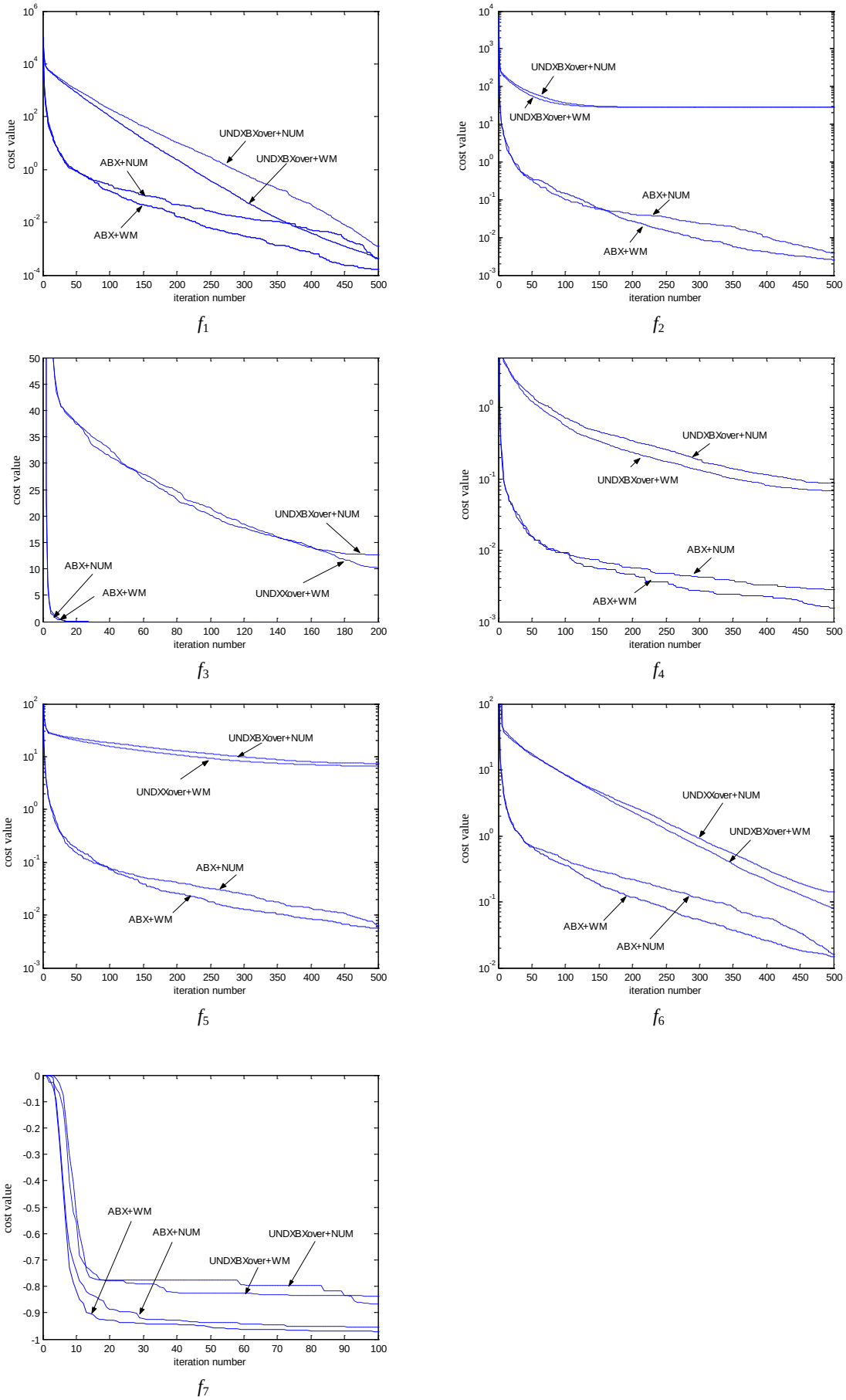


Fig. 6. Comparisons between different genetic operations for f_1 to f_7. All results are averaged ones over 50 runs.

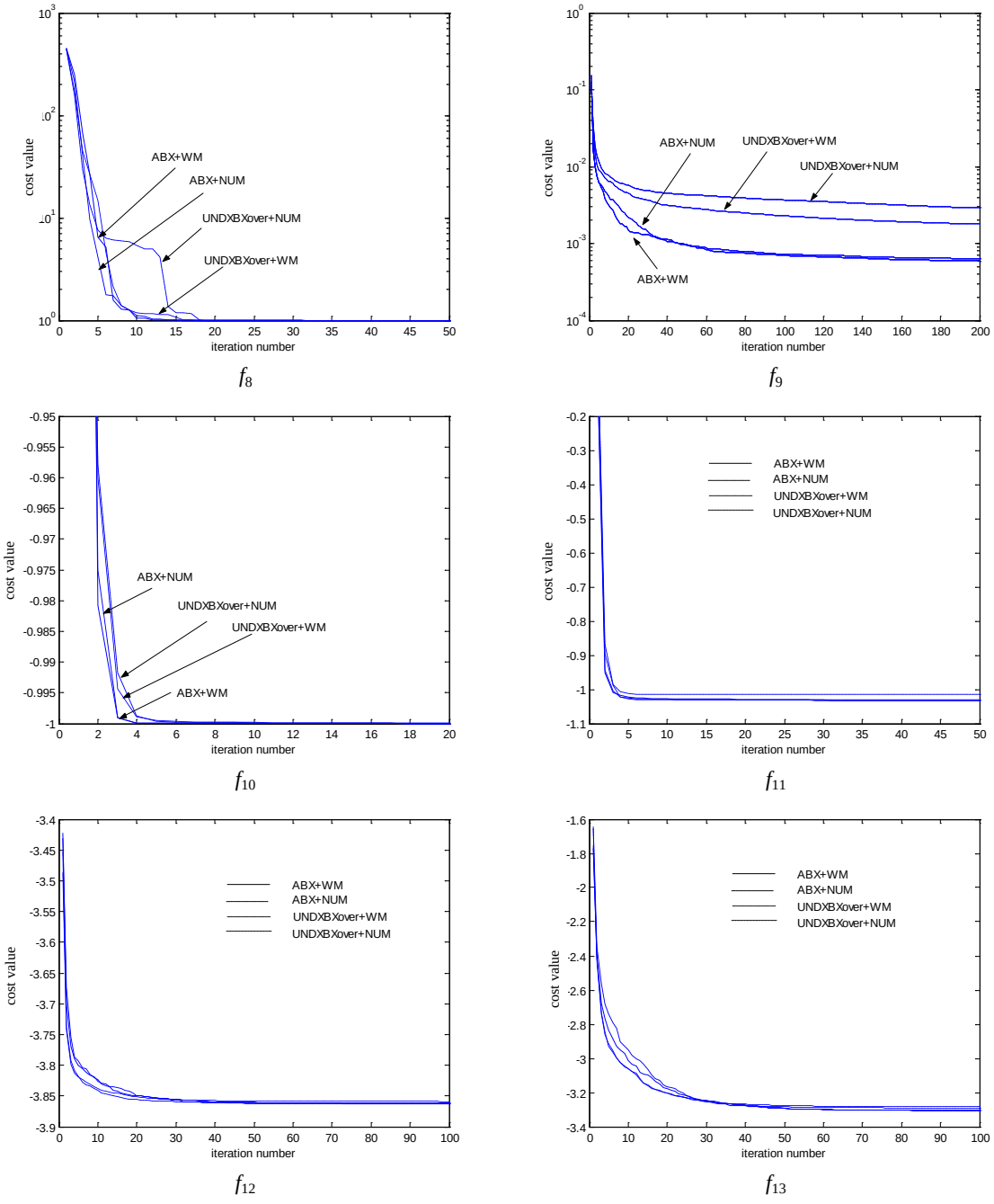
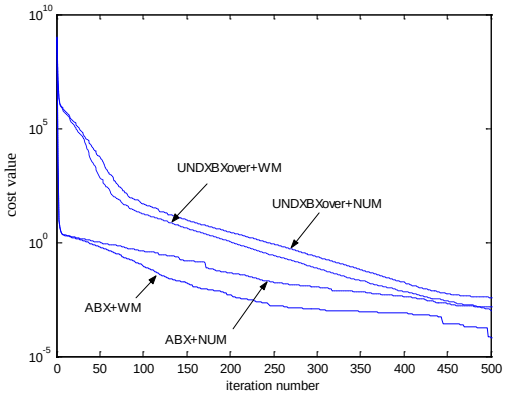
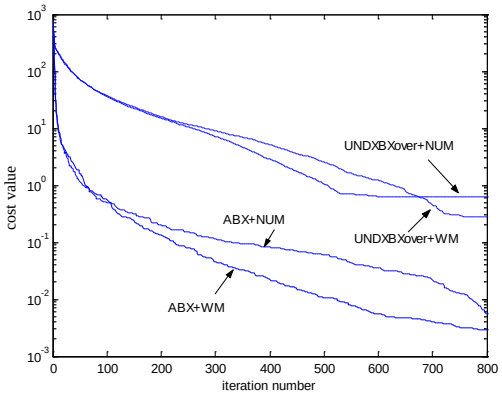


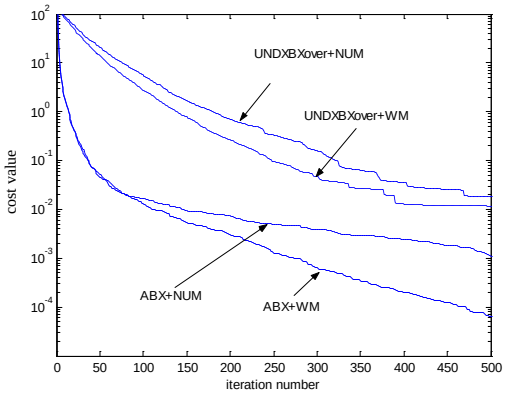
Fig. 7. Comparison between different genetic operations for f_8 to f_{13}. All results are averaged ones over 50 runs.



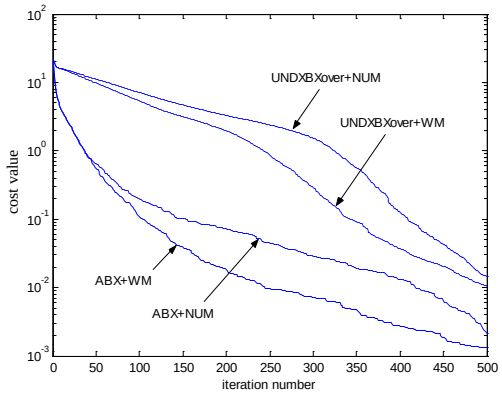
f_{14}



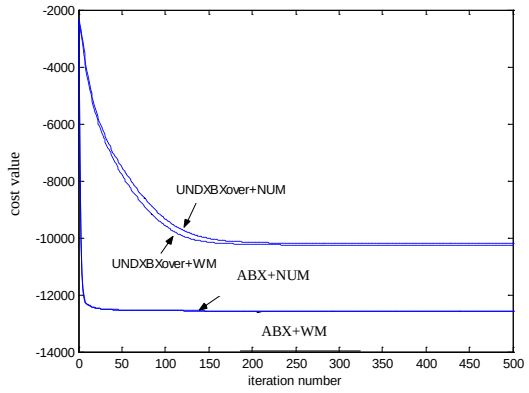
f_{15}



f_{16}



f_{17}



f_{18}

Fig. 8. Comparison between different genetic operations for f_{14} to f_{18}. All results are averaged ones over 50 runs.

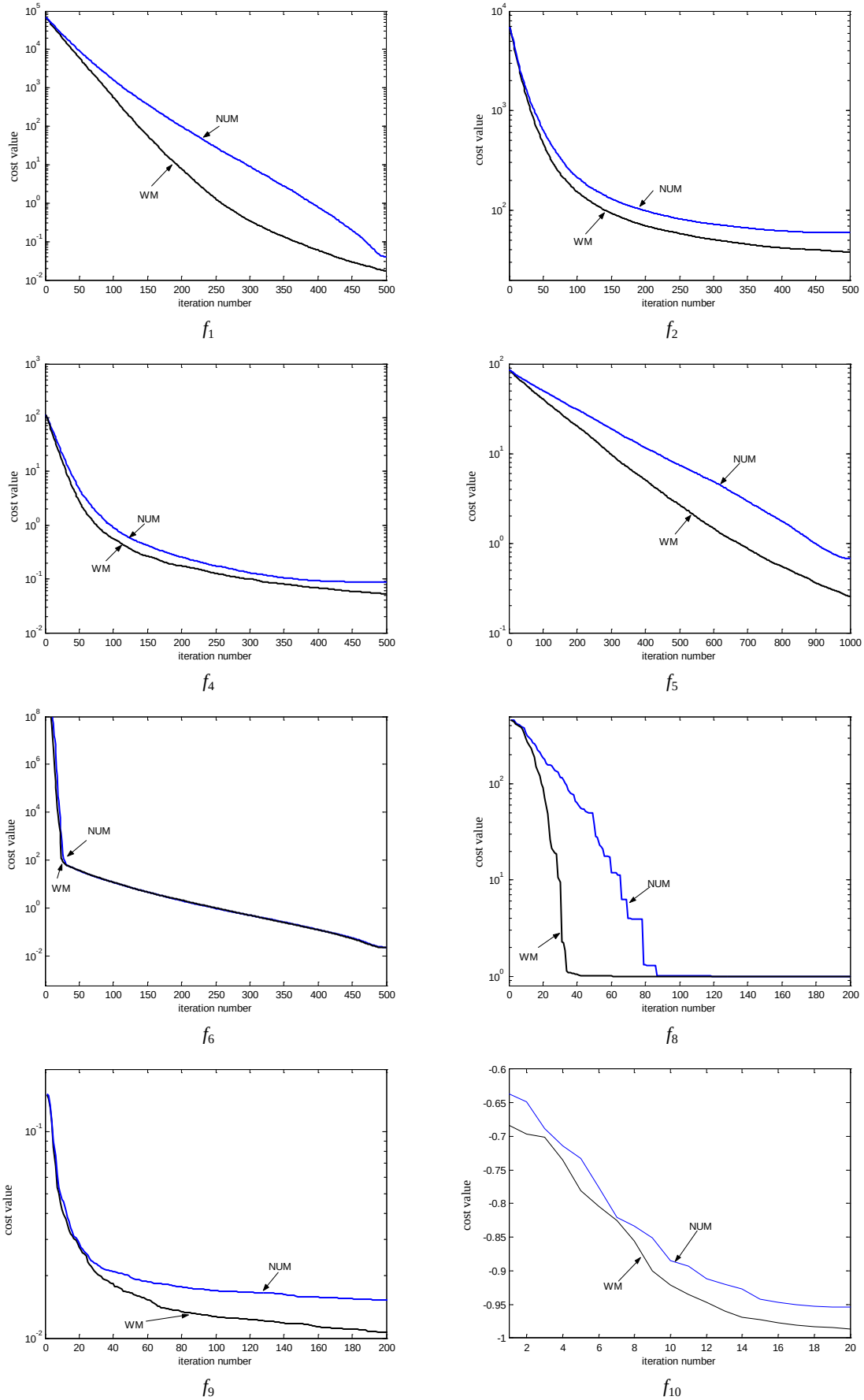
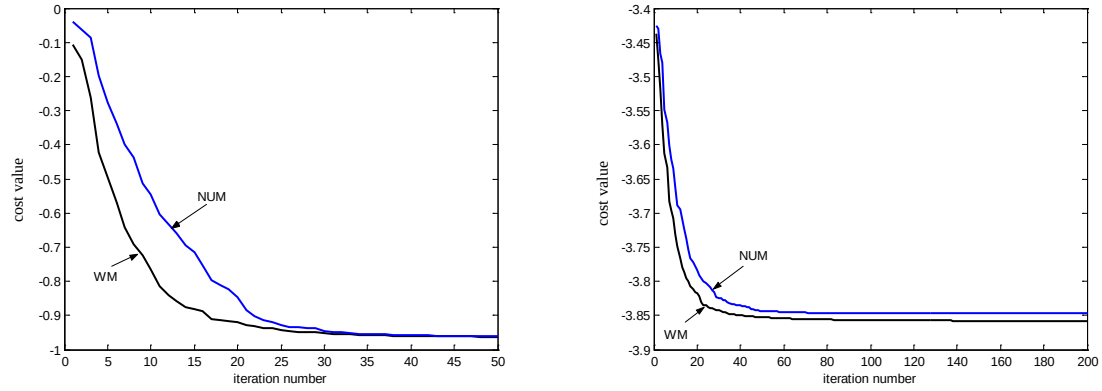
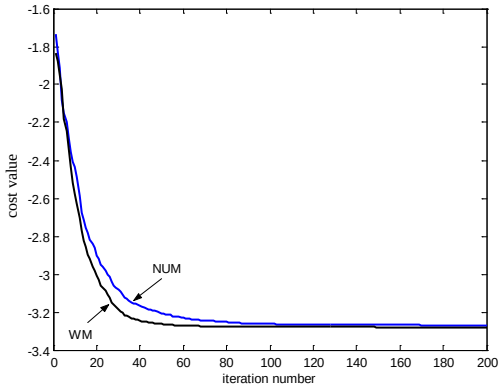


Fig. 9. Comparison between WM and NUM for f_1 to f_{10} (except f_3 and f_7) without the crossover operation. All results are averaged ones over 50 runs.

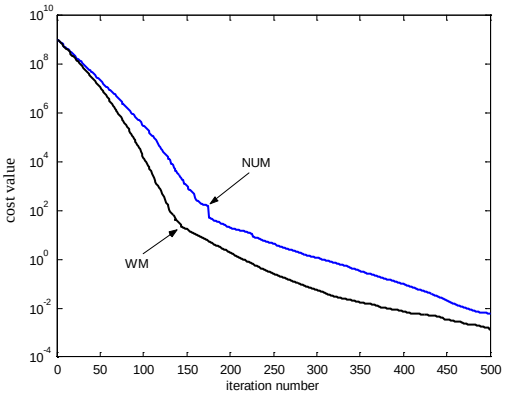


f_{11}

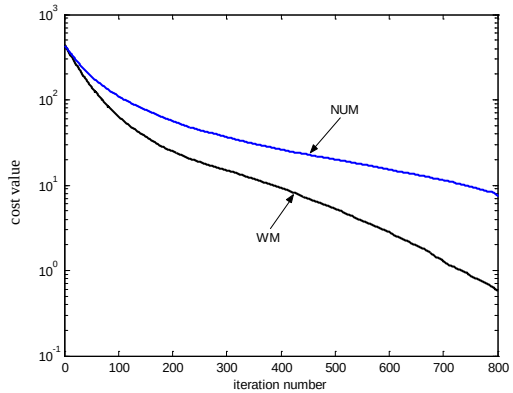
f_{12}



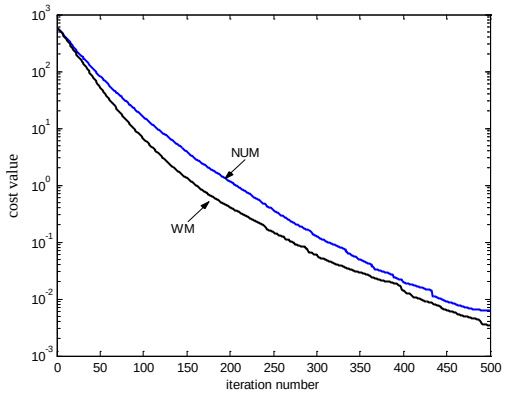
f_{13}



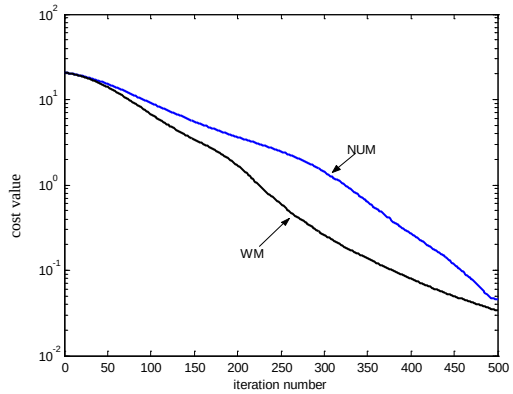
f_{14}



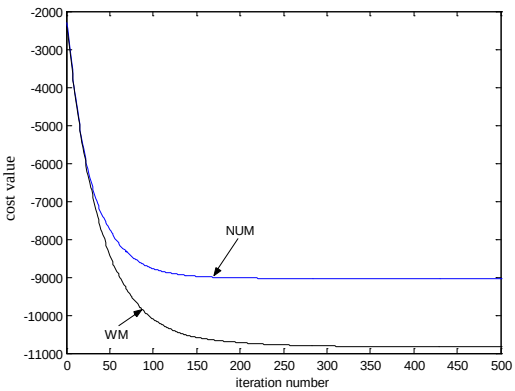
f_{15}



f_{16}



f_{17}



f_{18}

Fig. 10. Comparison between WM and NUM for f_{11} to f_{18} without the crossover operation. All results are averaged ones over 50 runs.

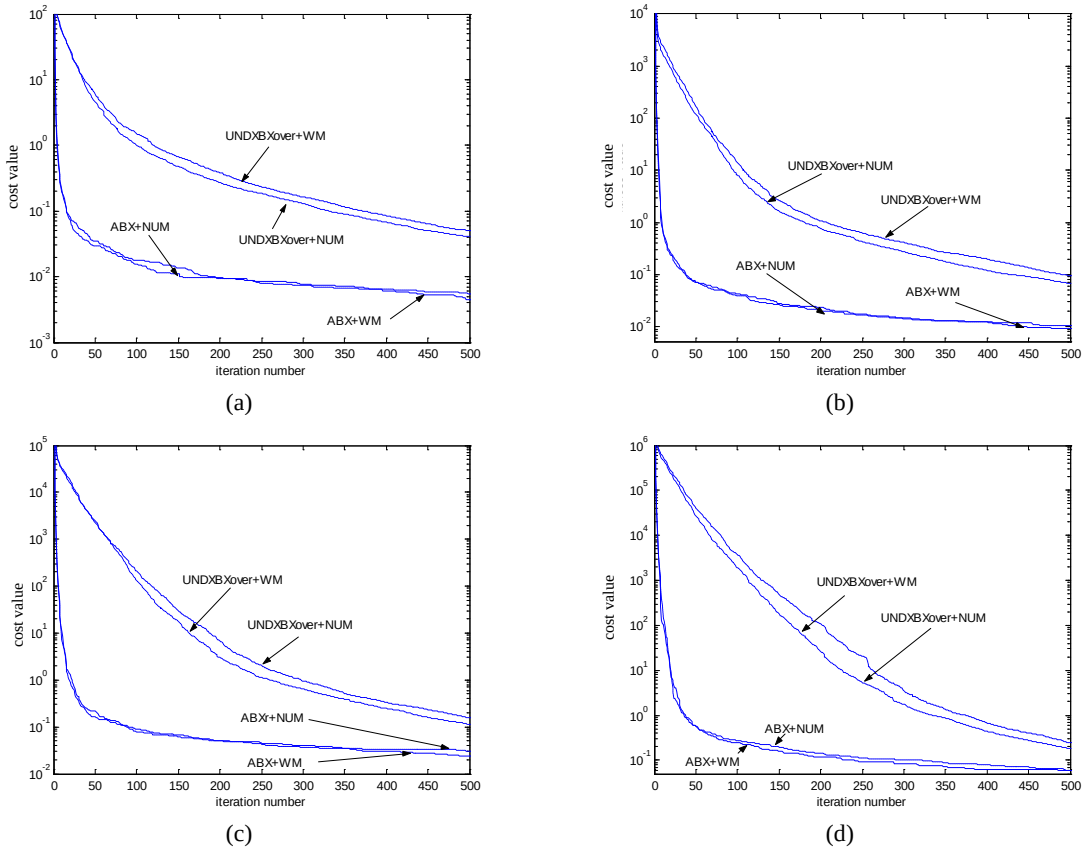


Fig. 11. Comparison between different genetic operations for f_4 with different initial ranges of variables. All results are averaged over 50 runs.

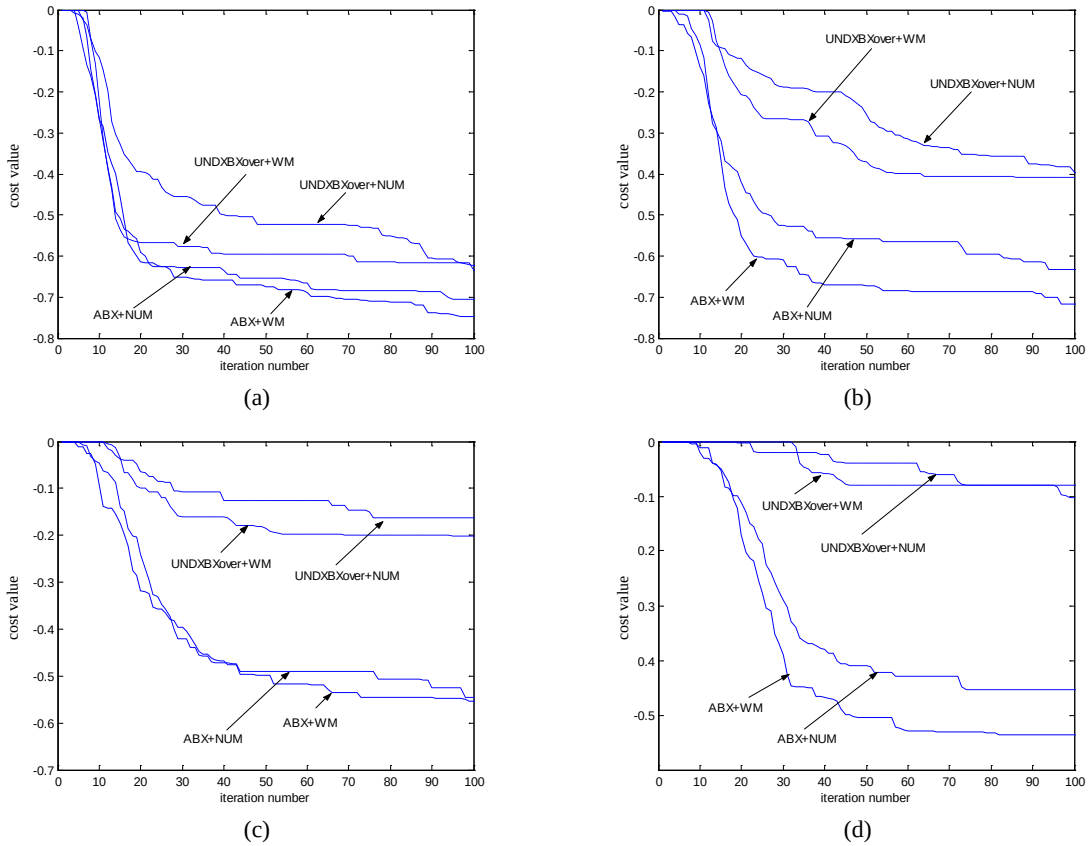


Fig. 12. Comparison between different genetic operations for f_7 with different initial ranges of variables. All results are averaged ones over 50 runs.

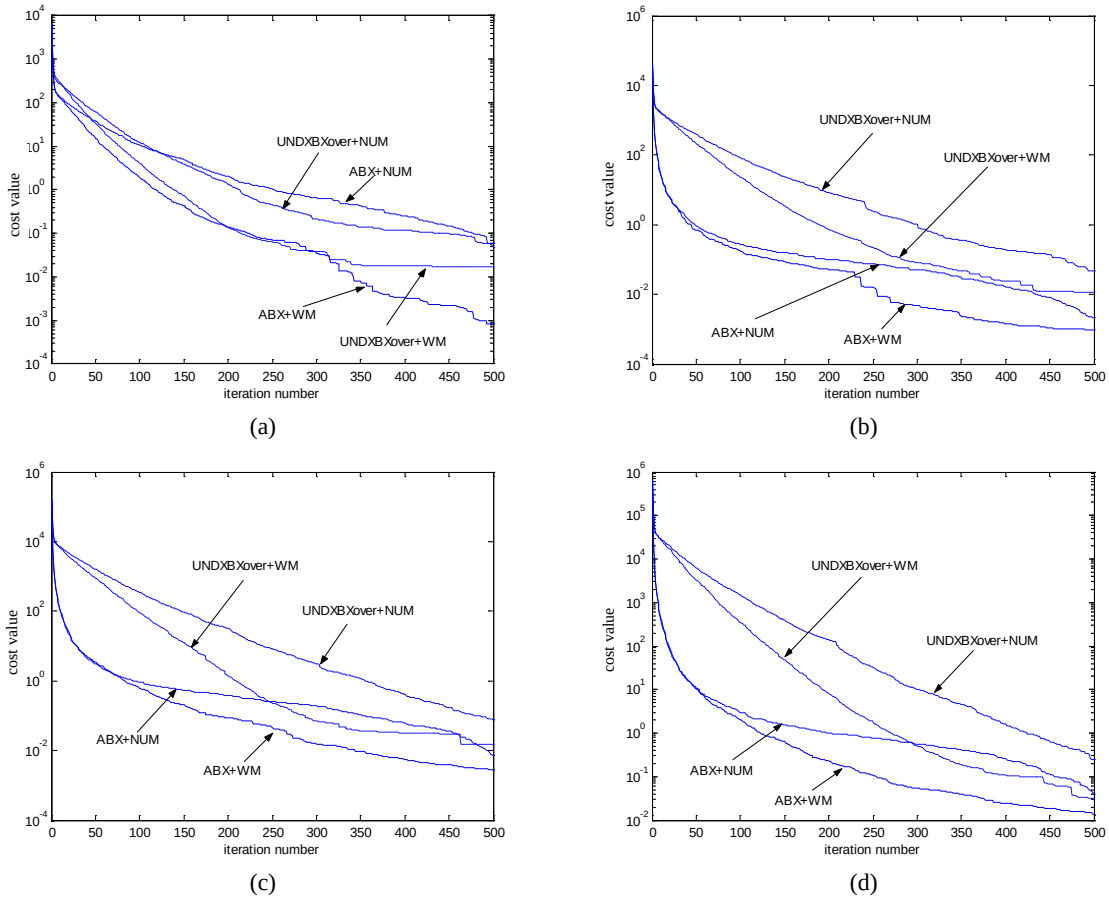


Fig. 13. Comparison between different genetic operations for f_{16} with different initial ranges of vriables. All results are averaged ones over 50 runs.

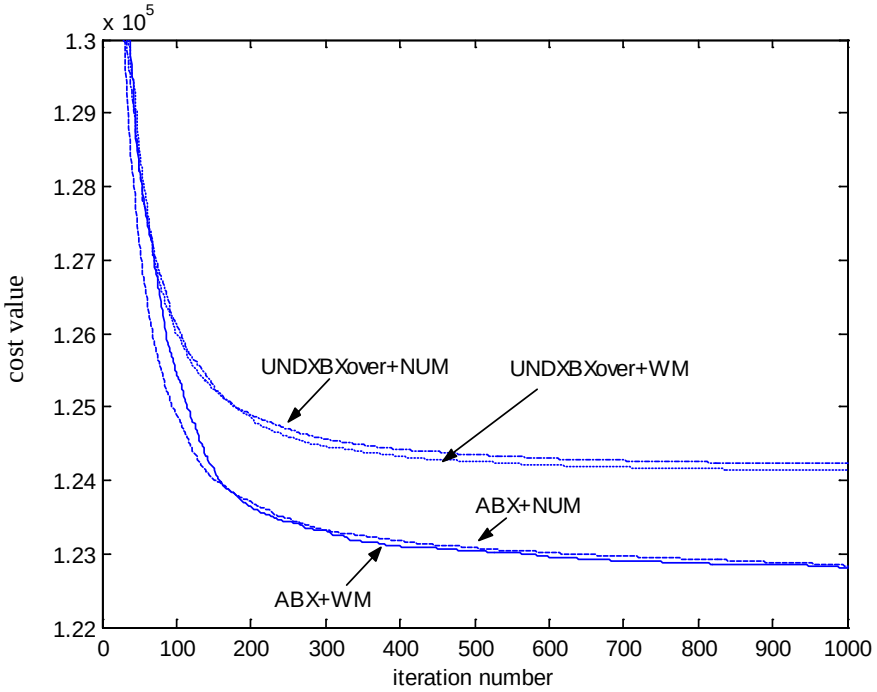


Fig. 14. Comparisons between different genetic operations for ELD. All results are averaged ones over 50 runs.

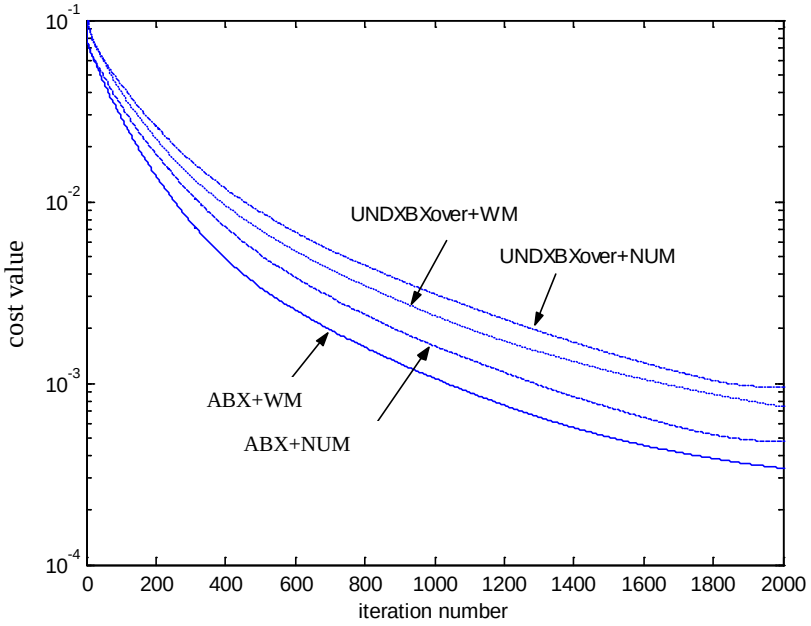


Fig. 15. Comparisons between different genetic operations for tuning associative memory. All results are averaged ones over 50 runs.

f_1 ($\times 10^{-4}$), number of iteration: 500				
	ABX+WM	ABX+NUM	UNDXBXover+WM	UNDXBXover+NUM
Mean	**1.6139**	4.1263	4.0840	12.614
Best	**0.00048**	0.0205	0.6095	4.5084
Std Dev	**2.9147**	5.1841	4.0513	6.7184
t-test ([ABX+WM]–[UNDXBXover+NUM]) = –10.62				

f_2 ($\times 10^{-2}$), number of iteration: 500				
	ABX+WM	ABX+NUM	UNDXBXover+WM	UNDXBXover+NUM
Mean	**0.24781**	0.38064	2768.9	2785.1
Best	0.02396	**0.01624**	2660.0	2638.0
Std Dev	**0.16678**	0.35854	59.999	62.845
t-test ([ABX+WM]–[UNDXBXover+NUM]) = –313.3				

f_3 ($\times 10^{0}$), number of iteration: 200				
	ABX+WM	ABX+NUM	UNDXBXover+WM	UNDXBXover+NUM
Mean	**0**	**0**	10.180	12.640
Best	**0**	**0**	1.0000	3.0000
Std Dev	**0**	**0**	5.0130	5.6524
t-test ([ABX+WM]–[UNDXBXover+NUM]) = –15.81				

f_4 ($\times 10^{-3}$), number of iteration: 500				
	ABX+WM	ABX+NUM	UNDXBXover+WM	UNDXBXover+NUM
Mean	**1.5503**	2.7952	67.758	87.028
Best	**0.2132**	0.4589	15.052	20.588
Std Dev	**1.0533**	2.1315	76.725	62.560
t-test ([ABX+WM]–[UNDXBXover+NUM]) = –9.66				

f_5 ($\times 10^{-2}$), number of iteration: 500				
	ABX+WM	ABX+NUM	UNDXBXover+WM	UNDXBXover+NUM
Mean	**0.5596**	0.6207	654.10	747.30
Best	**0.0072**	0.0503	201.66	214.20
Std Dev	**0.4483**	0.5905	378.04	510.33
t-test ([ABX+WM]–[UNDXBXover+NUM]) = –10.35				

f_6 ($\times 10^{-2}$), number of iteration: 500				
	ABX+WM	ABX+NUM	UNDXBXover+WM	UNDXBXover+NUM
Mean	**1.4924**	1.6115	7.9237	14.158
Best	0.1194	**0.0769**	0.5154	0.8621
Std Dev	**0.8047**	1.2105	6.5454	11.366
t-test ([ABX+WM]–[UNDXBXover+NUM]) = –7.86				

f_7 ($\times 10^{0}$), number of iteration: 100				
	ABX+WM	ABX+NUM	UNDXBXover+WM	UNDXBXover+NUM
Mean	**–0.9721**	–0.9549	–0.8679	–0.8365
Best	**–1.0000**	–0.9999	–1.0000	–1.0000
Std Dev	**0.0530**	0.1198	0.3277	0.3689
t-test ([ABX+WM]–[UNDXBXover+NUM]) = –2.57				

Table I. Comparison between different genetic operations for f_1 to f_7. All results are averaged ones over 50 runs.

f_8 ($\times 10^0$), number of iteration: 50				
	ABX+WM	ABX+NUM	UNDXBXover+WM	UNDXBXover+NUM
Mean	**0.9980**	**0.9980**	**0.9980**	**0.9980**
Best	**0.9980**	**0.9980**	**0.9980**	**0.9980**
Std Dev	$\mathbf{1.0277 \times 10^{-7}}$	1.3308×10^{-5}	1.3370×10^{-7}	2.5832×10^{-5}
t-test ([ABX+WM]–[UNDXBXover+NUM]) = N/A				

f_9 ($\times 10^{-4}$), number of iteration: 200				
	ABX+WM	ABX+NUM	UNDXBXover+WM	UNDXBXover+NUM
Mean	**5.9125**	6.3380	17.863	29.451
Best	**3.1002**	3.3428	3.3147	5.1236
Std Dev	2.7085	**2.6445**	36.986	5.5539
t-test ([ABX+WM]–[UNDXBXover+NUM]) = –26.94				

f_{10} ($\times 10^0$), number of iteration: 20				
	ABX+WM	ABX+NUM	UNDXBXover+WM	UNDXBXover+NUM
Mean	**–1**	**–1**	**–1**	**–1**
Best	**–1**	**–1**	**–1**	**–1**
Std Dev	**0**	**0**	3.2770×10^{-4}	5.3212×10^{-4}
t-test ([ABX+WM]–[UNDXBXover+NUM]) = N/A				

f_{11} ($\times 10^0$), number of iteration: 50				
	ABX+WM	ABX+NUM	UNDXBXover+WM	UNDXBXover+NUM
Mean	**–1.0316**	–1.0315	**–1.0316**	–1.0153
Best	**–1.0316**	**–1.0316**	**–1.0316**	**–1.0316**
Std Dev	$\mathbf{1.5724 \times 10^{-5}}$	1.5767×10^{-5}	7.7364×10^{-4}	1.1542×10^{-1}
t-test ([ABX+WM]–[UNDXBXover+NUM]) = –1.00				

f_{12} ($\times 10^0$), number of iteration: 100				
	ABX+WM	ABX+NUM	UNDXBXover+WM	UNDXBXover+NUM
Mean	**–3.8628**	–3.8627	–3.8591	**–3.8628**
Best	**–3.8628**	**–3.8628**	**–3.8628**	**–3.8628**
Std Dev	$\mathbf{2.9850 \times 10^{-4}}$	2.0403×10^{-3}	1.0288×10^{-1}	1.0858×10^{-3}
t-test ([ABX+WM]–[UNDXBXover+NUM]) = N/A				

f_{13} ($\times 10^0$), number of iteration: 100				
	ABX+WM	ABX+NUM	UNDXBXover+WM	UNDXBXover+NUM
Mean	–3.3051	**–3.3061**	–3.2837	–3.2905
Best	**–3.3220**	**–3.3220**	**–3.3220**	**–3.3220**
Std Dev	4.1617×10^{-2}	$\mathbf{4.0406 \times 10^{-2}}$	5.6158×10^{-2}	5.3560×10^{-2}
t-test ([ABX+WM]–[UNDXBXover+NUM]) = –1.55				

Table II. Comparison between different genetic operations on f_8 to f_{13}. All results are averaged ones over 50 runs.

f_{14} ($\times 10^{-4}$), number of iteration: 500				
	ABX+WM	ABX+NUM	UNDXBXover+WM	UNDXBXover+NUM
Mean	**0.6791**	14.340	10.181	38.643
Best	**0.0155**	0.0413	0.7273	0.3654
Std Dev	**1.1437**	36.254	22.759	51.452
t-test ([ABX+WM]–[UNDXBXover+NUM]) = –5.22				

f_{15} ($\times 10^{-3}$), number of iteration: 800				
	ABX+WM	ABX+NUM	UNDXBXover+WM	UNDXBXover+NUM
Mean	**2.8729**	5.6010	283.16	616.92
Best	**0.1817**	0.2907	0.8134	0.6561
Std Dev	**2.4540**	5.5464	534.34	940.46
t-test ([ABX+WM]–[UNDXBXover+NUM]) = –4.62				

f_{16} ($\times 10^{-5}$), number of iteration: 500				
	ABX+WM	ABX+NUM	UNDXBXover+WM	UNDXBXover+NUM
Mean	**6.3267**	109.63	1174.3	1797.4
Best	**0.1832**	5.0020	1.7688	6.0472
Std Dev	**6.1773**	81.434	5773.2	6998.2
t-test ([ABX+WM]–[UNDXBXover+NUM]) = –1.81				

f_{17} ($\times 10^{-3}$), number of iteration: 500				
	ABX+WM	ABX+NUM	UNDXBXover+WM	UNDXBXover+NUM
Mean	**1.3444**	1.7323	10.543	14.665
Best	**0.0203**	0.0655	5.2554	4.1783
Std Dev	**1.7323**	1.9284	4.0336	6.6873
t-test ([ABX+WM]–[UNDXBXover+NUM]) = –13.63				

f_{18} ($\times 10^{0}$), number of iteration: 500				
	ABX+WM	ABX+NUM	UNDXBXover+WM	UNDXBXover+NUM
Mean	**–12569.3**	–12596.2	–10261.9	–10188.9
Best	**–12569.5**	**–12569.5**	–11168.0	–11089.0
Std Dev	**0.1520**	0.2914	431.80	473.84
t-test ([ABX+WM]–[UNDXBXover+NUM]) = –35.52				

Table III. Comparison between different genetic operations for f_{14} to f_{18}. All results are averaged ones over 50 runs.

Function	Number of iteration	Wavelet Mutation (WM)			Non-Uniform Mutation (NUM)			*t*-test (WM-NUM)
		Mean	Best	Std Dev	Mean	Best	Std Dev	
$f_1\ (\times 10^{-3})$	500	**4.0356**	**1.0634**	**0.9161**	40.071	12.832	19.287	−13.197
$f_2\ (\times 10^{1})$	500	**3.8008**	**0.4671**	**2.3051**	5.9549	0.9366	3.2204	−3.8461
$f_4\ (\times 10^{-2})$	500	**5.2538**	**1.9215**	**1.8577**	8.8118	2.9487	2.9853	−7.1553
$f_5\ (\times 10^{-1})$	1000	**2.5280**	**1.5811**	**0.5138**	6.7490	3.4751	2.0596	−14.061
$f_6\ (\times 10^{-2})$	500	**2.2759**	**1.0900**	**0.6494**	2.4184	1.4767	0.6972	−1.0576
$f_8\ (\times 10^{0})$	200	**0.9980**	**0.9980**	$\mathbf{7.3982\times 10^{-11}}$	**0.9980**	**0.9980**	2.9079×10^{-10}	N/A
$f_9\ (\times 10^{-3})$	200	**9.3282**	0.6078	**9.6548**	13.393	**0.5910**	17.639	−1.4294
$f_{10}\ (\times 10^{0})$	20	**−0.9861**	**−1.0000**	**0.0212**	−0.9537	**−1.0000**	0.1283	−1.7618
$f_{11}\ (\times 10^{0})$	50	**−0.9627**	**−1.0316**	0.2614	−0.9599	**−1.0316**	**0.2239**	−0.0575
$f_{12}\ (\times 10^{0})$	200	**−3.8583**	**−3.8628**	**0.0078**	−3.8473	**−3.8628**	0.1093	−0.7098
$f_{13}\ (\times 10^{0})$	200	**−3.2791**	**−3.3220**	**0.0577**	−3.2695	**−3.3220**	0.0597	−0.8176
$f_{14}\ (\times 10^{-3})$	500	**1.2887**	**0.3558**	**0.8485**	6.3054	0.5732	5.5154	−6.3569
$f_{15}\ (\times 10^{0})$	800	**0.5777**	**0.1783**	**0.3784**	7.4349	4.6367	1.4969	−31.404
$f_{16}\ (\times 10^{-3})$	500	**3.3964**	0.6255	**2.7778**	6.3268	**0.1243**	41.201	−0.5018
$f_{17}\ (\times 10^{-2})$	500	**3.3688**	**2.1860**	**0.4944**	4.6260	2.3877	1.0430	−7.7018
$f_{18}\ (\times 10^{0})$	500	**−10837.4**	**−11502.9**	**315.1**	−9035.1	−10101.9	472.7	−8.7163

Table IV. Comparison between WM and NUM for f_1 to f_{18} (except f_3 and f_7) without crossover operations. All results are averaged ones over 50 runs.

Function	Number of iteration	$\zeta = 0.2$	$\zeta = 0.5$	$\zeta = 1.0$	$\zeta = 2.0$	$\zeta = 5.0$
$f_1\ (\times 10^{-4})$	500	92.259	32.283	10.997	7.9005	**1.6139**
$f_2\ (\times 10^{-2})$	500	4.3312	2.1228	1.4416	0.9221	**0.2478**
$f_3\ (\times 10^{0})$	200	0	**0**	0	0	0
$f_4\ (\times 10^{-3})$	500	2.2760	2.0321	1.9075	**1.5503**	1.7121
$f_5\ (\times 10^{-2})$	500	1.5283	1.4216	1.0127	**0.5596**	0.6012
$f_6\ (\times 10^{-2})$	500	14.552	10.062	5.2128	2.1627	**1.4924**
$f_7\ (\times 10^{0})$	100	−0.7511	**−0.9721**	−0.9592	−0.9296	−0.8918
$f_8\ (\times 10^{0})$	50	1.0065	0.9983	0.9980	**0.9980**	1.8193
$f_9\ (\times 10^{-4})$	200	7.4659	6.1928	6.3950	**5.9125**	6.4783
$f_{10}\ (\times 10^{0})$	20	−1.0000	−1.0000	−1.0000	**−1.0000**	−1.0000
$f_{11}\ (\times 10^{0})$	50	−1.0300	−1.0314	−1.0316	−1.0316	**−1.0316**
$f_{12}\ (\times 10^{0})$	100	−3.8627	**−3.8627**	−3.8302	−3.8509	−3.8471
$f_{13}\ (\times 10^{0})$	100	**−3.3051**	−3.3026	−3.2868	−3.2639	−3.1642
$f_{14}\ (\times 10^{-4})$	500	**0.6791**	1.0792	3.2004	2.4721	7.8236
$f_{15}\ (\times 10^{-3})$	800	42.220	10.684	5.2720	3.3008	**2.8729**
$f_{16}\ (\times 10^{-5})$	500	115.42	59.412	18.244	8.0794	**6.3267**
$f_{17}\ (\times 10^{-3})$	500	12.798	3.8139	3.5067	2.1405	**1.3444**
$f_{18}\ (\times 10^{-5})$	500	−12564.4	−12568.8	−12569.1	**−12569.4**	−12569.4

Table V. The mean cost values offered by wavelet mutation with different shape parameter ζ for function f_1 to f_{18}.

f_4 ($\times 10^{-3}$), number of iteration: 500				
Initial range: $-2.56 \le x_i \le 5.12$				
	ABX+WM	ABX+NUM	UNDXBXover+WM	UNDXBXover+NUM
Mean	**4.4350**	5.4386	41.007	50.831
Best	**0.2907**	0.5751	17.663	23.658
Std Dev	**3.2943**	4.0096	14.109	14.128
Initial range: $-6.4 \le x_i \le 12.8$				
	ABX+WM	ABX+NUM	UNDXBXover+WM	UNDXBXover+NUM
Mean	**9.1618**	10.376	68.112	96.057
Best	0.4297	**0.3694**	16.741	31.761
Std Dev	**5.6406**	8.2521	23.898	40.070
Initial range: $-12.8 \le x_i \le 25.6$				
	ABX+WM	ABX+NUM	UNDXBXover+WM	UNDXBXover+NUM
Mean	**23.982**	30.784	113.54	154.85
Best	3.0393	**0.8271**	47.182	62.557
Std Dev	**14.305**	26.989	36.270	62.135
Initial range: $-25.6 \le x_i \le 51.2$				
	ABX+WM	ABX+NUM	UNDXBXover+WM	UNDXBXover+NUM
Mean	**59.699**	60.358	182.00	241.12
Best	6.0383	**5.3199**	56.256	94.396
Std Dev	**35.603**	37.199	72.225	81.600

Table VI. Comparison between different operations for f_4 with different initial ranges of variables. All results are averaged ones over 50 runs.

f_7 ($\times 10^0$), number of iteration: 100				
Initial range: $-600 \le x_i \le 600$				
	ABX+WM	ABX+NUM	UNDXBXover+WM	UNDXBXover+NUM
Mean	**0.7471**	0.7062	0.6355	0.6227
Best	**1.0000**	**1.0000**	**1.0000**	**1.0000**
Std Dev	**0.3417**	0.3476	0.4820	0.4768
Initial range: $-1500 \le x_i \le 1500$				
	ABX+WM	ABX+NUM	UNDXBXover+WM	UNDXBXover+NUM
Mean	**0.7168**	0.6338	0.4079	0.3968
Best	**1.0000**	**1.0000**	**1.0000**	**1.0000**
Std Dev	**0.3669**	0.3863	0.4899	0.4656
Initial range: $-3000 \le x_i \le 3000$				
	ABX+WM	ABX+NUM	UNDXBXover+WM	UNDXBXover+NUM
Mean	**0.5527**	0.5451	0.2019	0.1626
Best	**1.0000**	**1.0000**	**1.0000**	**1.0000**
Std Dev	0.4206	0.4114	0.4030	**0.3544**
Initial range: $-60000 \le x_i \le 60000$				
	ABX+WM	ABX+NUM	UNDXBXover+WM	UNDXBXover+NUM
Mean	**0.5353**	0.1029	0.4533	0.0799
Best	**1.0000**	**1.0000**	**1.0000**	**1.0000**
Std Dev	0.4078	0.3027	0.4314	**0.2737**

Table VII. Comparison between different operations for f_7 with different initial ranges of variables. All results are averaged ones over 50 runs.

f_{16} ($\times 10^{-2}$), number of iteration: 500				
Initial range: $-2400 \le x_i \le 1200$				
	ABX+WM	ABX+NUM	UNDXBXover+WM	UNDXBXover+NUM
Mean	**0.0837**	4.7569	1.7523	5.9533
Best	**0.0013**	0.0996	0.0029	0.0043
Std Dev	**0.2108**	9.1365	6.9979	11.762
Initial range: $-6000 \le x_i \le 3000$				
	ABX+WM	ABX+NUM	UNDXBXover+WM	UNDXBXover+NUM
Mean	**0.0935**	0.2158	1.1821	4.5950
Best	0.0174	0.0048	**0.0011**	0.0146
Std Dev	**0.0871**	0.2408	5.7730	10.320
Initial range: $-12000 \le x_i \le 6000$				
	ABX+WM	ABX+NUM	UNDXBXover+WM	UNDXBXover+NUM
Mean	**0.2869**	0.7036	1.4933	7.5643
Best	0.0142	0.0351	**0.0072**	0.1285
Std Dev	**0.4371**	0.5949	6.5196	11.531
Initial range: $-24000 \le x_i \le 12000$				
	ABX+WM	ABX+NUM	UNDXBXover+WM	UNDXBXover+NUM
Mean	**1.3310**	3.9503	3.2419	2.4231
Best	0.0775	0.1872	**0.0329**	1.0387
Std Dev	**1.3098**	5.0784	8.9199	30.433

Table VIII. Comparison between different operations for f_{16} with different initial ranges of variables. All results are averaged ones over 50 runs.

Unit	$P_{L,\min}$ (MW)	$P_{L,\max}$ (MW)	a	b	c	e	f
1	36	114	0.00690	6.73	94.705	100	100
2	36	114	0.00690	6.73	94.705	100	100
3	60	120	0.02028	7.07	309.54	100	100
4	80	190	0.00942	8.18	369.03	150	150
5	47	97	0.01142	5.35	148.89	120	120
6	68	140	0.01142	8.05	222.33	100	100
7	110	300	0.00357	8.03	287.71	200	200
8	135	300	0.00492	6.99	391.98	200	200
9	135	300	0.00573	6.60	455.76	200	200
10	130	300	0.00605	12.9	722.82	200	200
11	94	375	0.00515	12.9	635.20	200	200
12	94	375	0.00569	12.8	654.69	200	200
13	125	500	0.00421	12.5	913.40	300	300
14	125	500	0.00752	8.84	1760.4	300	300
15	125	500	0.00708	9.15	1728.3	300	300
16	125	500	0.00708	9.15	1728.3	300	300
17	220	500	0.00313	7.97	647.85	300	300
18	220	500	0.00313	7.95	649.69	300	300
19	242	550	0.00313	7.97	647.83	300	300
20	242	550	0.00313	7.97	647.81	300	300
21	254	550	0.00298	6.63	785.96	300	300
22	254	550	0.00298	6.63	785.96	300	300
23	254	550	0.00284	6.66	794.53	300	300
24	254	550	0.00284	6.66	794.53	300	300
25	254	550	0.00277	7.10	801.32	300	300
26	254	550	0.00277	7.10	801.32	300	300
27	10	150	0.52124	3.33	1055.1	120	120
28	10	150	0.52124	3.33	1055.1	120	120
29	10	150	0.52124	3.33	1055.1	120	120
30	47	97	0.01140	5.35	148.89	120	120
31	60	190	0.00160	6.43	222.92	150	150
32	60	190	0.00160	6.43	222.92	150	150
33	60	190	0.00160	6.43	222.92	150	150
34	90	200	0.00010	8.95	107.87	200	200
35	90	200	0.00010	8.62	116.58	200	200
36	90	200	0.00010	8.62	116.58	200	200
37	25	110	0.01610	5.88	307.45	80	80
38	25	110	0.01610	5.88	307.45	80	80
39	25	110	0.01610	5.88	307.45	80	80
40	242	550	0.00313	7.97	647.83	300	300

Table IX. Units Data (40 systems with valve-point loadings): a (\$/MW^{2}h), b (\$/MWh), c (\$/h), e (\$/h), and f (rad/MW) are cost coefficients.

	ABX+WM	ABX+NUM	UNDXBXover+WM	UNDXBXover+NUM
Ave. Cost	**122811.41**	122840.26	124130.58	124223.25
Best Cost	**121915.93**	122232.21	122763.44	123642.92
Worst Cost	**123334.00**	123532.57	125092.30	124636.30
Std. Dev.	**313.79**	333.36	818.10	805.12
t-test ([ABX+WM]–[UNDXBXover+NUM]) = –11.55				

Table X. Statistical results for ELD with a load demand of 10500MW.

P_{L_i} (MW), i =	ABX+WM	ABX+NUM	UNDXBXover+WM	UNDXBXover+NUM
1	110.97	81.82	110.95	83.56
2	110.88	112.69	111.20	110.84
3	98.17	98.03	99.25	97.44
4	178.85	179.84	179.75	129.88
5	87.78	87.92	90.01	87.80
6	140.00	105.49	139.96	105.44
7	260.37	259.57	265.71	259.67
8	286.83	285.45	286.60	285.00
9	285.14	284.23	284.64	211.63
10	204.86	203.59	204.82	258.57
11	165.98	168.81	168.81	168.68
12	167.75	94.00	243.62	243.57
13	214.31	304.53	304.56	394.27
14	305.65	394.11	304.54	304.47
15	393.66	394.43	394.28	304.52
16	394.60	394.53	394.28	484.02
17	489.22	489.47	400.03	399.61
18	489.25	489.25	399.70	489.27
19	511.23	511.18	511.33	511.27
20	510.69	511.26	511.58	511.28
21	524.74	523.39	523.55	523.33
22	525.52	523.43	526.32	523.28
23	522.98	524.18	523.52	523.39
24	523.22	524.24	530.03	523.41
25	523.26	523.96	523.28	523.48
26	523.32	523.33	523.39	523.30
27	10.00	10.00	10.92	10.38
28	10.00	10.00	10.29	10.49
29	10.00	10.00	11.14	10.24
30	88.86	89.50	94.38	89.33
31	162.30	160.10	170.87	160.34
32	177.94	159.96	161.74	159.78
33	160.18	163.09	182.87	165.00
34	166.54	165.05	172.89	164.80
35	164.80	169.30	177.58	169.58
36	170.68	170.48	165.67	183.22
37	108.17	89.16	89.78	90.24
38	100.68	108.73	93.91	108.69
39	109.34	90.19	90.76	85.46
40	511.28	511.51	511.29	511.27
Total Power	10500	10500	10500	10500
Total Cost ($h)	121915.93	122232.21	122763.44	123642.92

Table XI. The optimal dispatch solution for different approaches.

MSE ($\times 10^{-4}$), number of iteration: 2000				
	ABX+WM	ABX+NUM	UNDXBXover+WM	UNDXBXover+NUM
Mean	**3.3939**	4.7965	7.4178	9.5312
Best	**2.5154**	2.8983	4.1872	5.8292
Std Dev	**0.4728**	0.9566	2.1383	1.6943
t-test ([ABX+WM]–[UNDXBXover+NUM]) = –24.67				

Table XII. Statistical results for the example of associative memory

APPENDIX

A. Benchmark test function

The 18 benchmark test functions for testing the RCGA performance are listed below. In these functions, $\mathbf{x} = [x_1 \quad x_2 \quad \cdots \quad x_n]$.

Unimodal Functions:

1. Sphere Model

$$f_1(\mathbf{x}) = \sum_{i=1}^{30} x_i^2 \,, -50 \le x_i \le 150, \min(f_1) = f_1(\mathbf{0}) = 0$$

2. Generalized Rosenbrock's Function

$$f_2(\mathbf{x}) = \sum_{i=1}^{29} \left[100\left(x_{i+1} - x_i^2\right)^2 + (x_i - 1)^2\right], \; -2.048 \le x_i \le 2.048 \,, \min(f_2) = f_2(\mathbf{1}) = 0$$

3. Step Function

$$f_3(\mathbf{x}) = \sum_{i=1}^{30} \left(\lfloor x_i + 0.5 \rfloor\right)^2 \,, \; -5 \le x_i \le 10 \,, \min(f_3) = f_3(\mathbf{0}) = 0$$

4. Quartic Function (with noise)

$$f_4(\mathbf{x}) = \sum_{i=1}^{30} i x_i^4 + random[0, \quad 1), \; -1.28 \le x_i \le 2.56 \,, \min(f_4) = f_4(\mathbf{0}) = 0$$

where $random[0, \quad 1)$ generates uniformly a floating-point number between 0 and 1.

5. Schwefel's Problem 2.21

$$f_5(\mathbf{x}) = \max_i \{|x_i|, 1 \le i \le 30\}, -150 \le x_i \le 50 \,, \min(f_5) = f_5(\mathbf{0}) = 0$$

6. Schwefel's Problem 2.22

$$f_6(\mathbf{x}) = \sum_{i=1}^{30} |x_i| + \prod_{i=1}^{30} |x_i| \,, -5 \le x_i \le 15 \,, \min(f_6) = f_6(\mathbf{0}) = 0$$

Multimodal Functions with Only a Few Local Minima:

7. Eason's Function

$$f_7(\mathbf{x}) = -\cos(x_1)\cdot\cos(x_2)\cdot\exp\left(-\left((x_1-\pi)^2+(x_2-\pi)^2\right)\right), -300 \le x_1, x_2 \le 300,$$

$\min(f_7) = f_7([\pi, \quad \pi]) = -1,$

8. Shekel's Foxholes Function

$$f_8(\mathbf{x}) = \left[\frac{1}{500} + \sum_{j=1}^{25}\frac{1}{j+\sum_{i=1}^{2}(x_i - a_{ij})^6}\right]^{-1}, \; -65.536 \le x_i \le 65.536, \; \min(f_8) = f_8([-32, \quad -32]) \approx 1,$$

where

$$a_{ij} = \begin{pmatrix} -32 & -16 & 0 & 16 & 32 & -32 & \cdots & 0 & 16 & 32 \\ -32 & -32 & -32 & -32 & -32 & -16 & \cdots & 32 & 32 & 32 \end{pmatrix}$$

9. Kowalik's Function

$$f_9(\mathbf{x}) = \sum_{i=1}^{9}\left[a_i - \frac{x_1(b_i^2 + b_i x_2)}{b_i^2 + b_i x_3 + x_4}\right]^2, -5 \le x_i \le 5,$$

$\min(f_9) = f_9([0.1928, \quad 0.1908, \quad 0.1231, \quad 0.1358]) \approx 0.0003075$

where

i	a_i	b_i
1	0.1957	4
2	0.1947	2
3	0.1735	1
4	0.1600	1/2
5	0.0844	1/4
6	0.0627	1/6
7	0.0456	1/8
8	0.0342	1/10
9	0.0323	1/12
10	0.0235	1/14
11	0.0246	1/16

10. Maxican hat Function

$$f_{10}(\mathbf{x}) = -\frac{\sin(x_1)\sin(x_2)}{x_1 x_2}, -5 \le x_1, x_2 \le 15, \; \min(f_{10}) = \lim_{x\to(0,0)} f_{10}(\mathbf{x}) = 0$$

11. Six-Hump Camel Back Function

$$f_{11}(\mathbf{x}) = 4x_1^2 - 2.1x_1^4 + \frac{1}{3}x_1^6 + x_1x_2 - 4x_2^2 + 4x_2^4, -5 \le x_1, x_2 \le 5,$$

$\min(f_{11}) = f_{11}([0.08983, \quad -0.7126]) = f_{11}([-0.08983, \quad 0.7126]) \approx -1.0316$

12. Hartman's Family I

$$f_{12}(\mathbf{x}) = -\sum_{i=1}^{4} c_i \exp\left[-\sum_{j=1}^{3} a_{ij}\left(x_j - p_{ij}\right)^2\right], 0 \le x_i \le 1,$$

$\min(f_{12}) = f_{12}([0.114, \quad 0.556, \quad 0.852]) \approx -3.8628$,

where

	a_{ij}			p_{ij}		
i \ j	1	2	3	1	2	3
1	3	10	30	0.3689	0.1170	0.2673
2	0.1	10	35	0.4699	0.4387	0.7470
3	3	10	30	0.1091	0.8732	0.5547
4	0.1	10	35	003815	0.5743	0.8828

i	c_i
1	1
2	1.2
3	3
4	3.2

13. Hartman's Family II

$$f_{13}(\mathbf{x}) = -\sum_{i=1}^{4} c_i \exp\left[-\sum_{j=1}^{6} a_{ij}\left(x_j - p_{ij}\right)^2\right], 0 \le x_i \le 1,$$

$\min(f_{13}) = f_{13}([0.201, \quad 0.15, \quad 0.477, \quad 0.275, \quad 0.311 \quad 0.627]) \approx -3.32$, where

	a_{ij}						p_{ij}					
i \ j	1	2	3	4	5	6	1	2	3	4	5	6
1	10	3	17	3.5	1.7	8	0.1312	0.1696	0.5569	0.0124	0.8283	0.5886
2	0.05	10	17	0.1	8	14	0.2329	0.4135	0.8307	0.3736	0.1004	0.9991
3	3	3.5	1.7	10	17	8	0.2348	0.1415	0.3522	0.2883	0.3047	0.6650
4	17	8	0.05	10	0.1	14	0.4047	0.8828	0.8732	0.5743	0.1091	0.0381

i	c_i
1	1
2	1.2
3	3
4	3.2

Multimodal Functions with Many Local Minima:

14. Generalized Penalized Functions

$$f_{14}(\mathbf{x}) = 0.1\left\{\sin^2(\pi 3x_1) + \sum_{i=1}^{29}(x_i - 1)^2 \cdot \left[1 + \sin^2(3\pi x_{i+1})\right] + (x_{30} - 1)^2\left[1 + \sin^2(2\pi x_{30})\right]\right\}$$

$$+ \sum_{i=1}^{30} u(x_i, 5, 100, 4),$$

$-50 \le x_i \le 50$, $\min(f_{14}) = f_{14}(\mathbf{1}) = 0$

where

$$u(x_i, a, k, m) = \begin{cases} k(x_i - a)^m, & x_i > a, \\ 0, & -a \le x_1 \le a \\ k(-x_i - a)^m, & x_i < a \end{cases}$$

15. Generalized Rastrigin's Function

$$f_{15}(\mathbf{x}) = \sum_{i=1}^{30}\left[x_i^2 - 10\cos(2\pi x_i) + 10\right], -5.12 \le x_i \le 10.24, \min(f_{15}) = f_{15}(\mathbf{0}) = 0$$

16. Generalized Griewank Function

$$f_{16}(\mathbf{x}) = \frac{1}{4000}\sum_{i=1}^{30} x_i^2 - \prod_{i=1}^{30}\cos\left(\frac{x_i}{\sqrt{i}}\right) + 1, \quad -1200 \le x_i \le 600, \min(f_{16}) = f_{16}(\mathbf{0}) = 0$$

17. Ackley's Function

$$f_{17}(\mathbf{x}) = -20\exp\left(-0.2\sqrt{\frac{1}{30}\sum_{i=1}^{30} x_i^2}\right) - \exp\left(\frac{1}{30}\sum_{i=1}^{30}\cos 2\pi x_i\right) + 20 + e, \; -64 \le x_i \le 32,$$

$\min(f_{17}) = f_{17}(\mathbf{0}) = 0$

18. Schwefel's Function

$$f_{18}(\mathbf{x}) = -\sum_{i=1}^{30}\left(x_i \sin\left(\sqrt{|x_i|}\right)\right), \; -500 \le x_i \le 500,$$

$\min(f_{18}) = f_{18}([420.9687, \quad \cdots \quad 420.9687]) = -30 \times 418.9829 = -12569.5$

B. Unimodal normal distribution crossover (UNDX)

Unimodal normal distribution crossover is defined as a mixture of three selected parents $\mathbf{p}_1$, $\mathbf{p}_2$, and $\mathbf{p}_3$. The resulting offspring $\mathbf{o}_{\mathbf{s}_c}$ is defined as,

$$\mathbf{o}_{\mathbf{s}_c^1} = \left[o_{s_1^1} \quad o_{s_2^1} \quad \cdots \quad o_{s_{no\_vars}^1} \right] = \mathbf{m} + z_1 e_1 + \sum_{i=2}^{no\_vars} z_i e_i, \tag{A.1}$$

$$\mathbf{o}_{\mathbf{s}_c^2} = \left[o_{s_1^1} \quad o_{s_2^1} \quad \cdots \quad o_{s_{no\_vars}^1} \right] = \mathbf{m} - z_1 e_1 - \sum_{i=2}^{no\_vars} z_i e_i, \tag{A.2}$$

where

$$\mathbf{m} = \frac{(\mathbf{p}_1 + \mathbf{p}_2)}{2}, \tag{A.3}$$

$$z_1 = N(0, \sigma_1^2), \quad z_i = N(0, \sigma_2^2), \tag{A.4}$$

$$\sigma_1 = \beta d_1, \quad \sigma_2 = \frac{\mu d_2}{\sqrt{no\_vars}}, \tag{A.5}$$

$$e_1 = \frac{(\mathbf{p}_2 - \mathbf{p}_1)}{|\mathbf{p}_2 - \mathbf{p}_1|}, \tag{A.6}$$

$$e_m \perp e_n \ (m \neq n), \quad m, n = 1, \ldots, no\_vars, \tag{A.7}$$

where $N(\cdot)$ is a normal distributed random number, d_1 is the distance between the parents $\mathbf{p}_1$ and $\mathbf{p}_2$, d_2 is the distance of $\mathbf{p}_3$ from the line connecting $\mathbf{p}_1$ and $\mathbf{p}_2$, β and μ are constant.

C. Blend crossover (BLX-α)

Blend crossover is defined as a combination of two selected parents $\mathbf{p}_1$ and $\mathbf{p}_2$. The resulting offspring $\mathbf{o}_{\mathbf{s}_c} = \left[o_{s_1}, \ o_{s_2}, \ \cdots, o_{s_{no\_vars}} \right]$ is chosen randomly from the interval $\left[X_i^1, X_i^2 \right]$ following the uniform distribution, where

$$X_i^1 = \min\left(p_{1_i}, p_{2_i} \right) - \alpha d_i, \tag{A.8}$$

$$X_i^2 = \max\left(p_{1_i}, p_{2_i} \right) + \alpha d_i, \tag{A.9}$$

where

$d_i = \left| p_{1_i} - p_{2_i} \right|$, p_{1_i} and p_{2_i} are the *i*-th elements of $\mathbf{p}_1$ and $\mathbf{p}_2$ respectively, and α is a positive constant.

D. Non-uniform mutation (NUM)

Non-uniform mutation is an operation with a fine-tuning capability. Its action depends on the generation number of the population. The operation takes place as follows. If $\mathbf{o_s} = \left[o_{s_1}, \quad o_{s_2}, \quad \ldots, \quad o_{s_{no\_vars}} \right]$ is a chromosome and the element o_{s_k} is randomly selected for mutation (the value of o_{s_k} is inside [$para^k_{\min}, para^k_{\max}$]), the resulting chromosome is then given by $\hat{\mathbf{o}}_\mathbf{s} = \left[o_{s_1}, \quad \ldots \quad , \hat{o}_{s_k}, \quad \ldots \quad , o_{s_{no\_vars}} \right]$, $k \in 1, 2, \ldots no\_vars$, and

$$\hat{o}_{s_k} = \begin{cases} o_{s_k} + \Delta\left(\tau, para^k_{\max} - o_{s_k}\right) \text{ if } r_d = 0 \\ o_{s_k} - \Delta\left(\tau, o_{s_k} - para^k_{\min}\right) \text{ if } r_d = 1 \end{cases}, \tag{A.10}$$

where r_d is a random number equal to 0 or 1 only. The function $\Delta(\tau, y)$ returns a value in the range [0, *y*] such that $\Delta(\tau, y)$ approaches 0 as τ increases. It is defined as follows,

$$\Delta(\tau, y) = y\left(1 - r^{\left(1 - \frac{\tau}{T}\right)^b}\right), \tag{A.11}$$

where *r* is a random number in [0, 1], τ is the present generation number of the population, *T* is the maximum generation number of the population, and *b* is a system parameter that determines the degree of non-uniformity.