Hindawi Publishing Corporation
Mathematical Problems in Engineering
Volume 2015, Article ID 474805, 15 pages
http://dx.doi.org/10.1155/2015/474805

Research Article

The Algorithm for Algorithms: An Evolutionary Algorithm Based on Automatic Designing of Genetic Operators

Dazhi Jiang[1,2] and Zhun Fan[2,3]

[1] *Department of Computer Science, Shantou University, Shantou 515063, China*
[2] *Guangdong Provincial Key Laboratory of Digital Signal and Image Processing, Shantou University, Shantou 515063, China*
[3] *Department of Electronic and Information Engineering, Shantou University, Shantou 515063, China*

Correspondence should be addressed to Zhun Fan; zfan@stu.edu.cn

Received 11 May 2014; Accepted 2 August 2014

Academic Editor: Hailin Liu

At present there is a wide range of evolutionary algorithms available to researchers and practitioners. Despite the great diversity of these algorithms, virtually all of the algorithms share one feature: they have been manually designed. A fundamental question is "are there any algorithms that can design evolutionary algorithms automatically?" A more complete definition of the question is "can computer construct an algorithm which will generate algorithms according to the requirement of a problem?" In this paper, a novel evolutionary algorithm based on automatic designing of genetic operators is presented to address these questions. The resulting algorithm not only explores solutions in the problem space like most traditional evolutionary algorithms do, but also automatically generates genetic operators in the operator space. In order to verify the performance of the proposed algorithm, comprehensive experiments on 23 well-known benchmark optimization problems are conducted. The results show that the proposed algorithm can outperform standard differential evolution algorithm in terms of convergence speed and solution accuracy which shows that the algorithm designed automatically by computers can compete with the algorithms designed by human beings.

1. Introduction

At present there is a wide range of evolutionary algorithms available to researchers and practitioners. Despite the great diversity of these algorithms, virtually all of the algorithms share one feature: they have been manually designed. As a result, inevitably, current evolutionary algorithms in general incorporate human preconceptions in their designs. This situation encourages us to ask the following questions: are there any algorithms that can design evolutionary algorithms automatically? A more complete definition of the question is "can computer construct an algorithm which will generate algorithms according to the requirement of a problem?" In the 13th century, a French scientist, Villand de Honnecourt proposed a perpetual motion machine for the first time. However, the truth shows that it is impossible to make such kind of entity in the real world. But in the world of computer for the machine to automatically design algorithms, several automatic algorithm design techniques have been proposed in recent years to overcome this limitation. For example, hyperheuristics include search methods that automatically select and combine simpler heuristics, creating a generic heuristic that is used to solve more general instances of a given type of optimization problem. Hence, hyperheuristics search in the space of heuristics, instead of in the problem solution space [1], raising the level of generality of the solutions produced by the hyperheuristics. Ant Colony algorithms are population-based methods widely used in combinatorial optimization problems. Taveres and Pereira [2] proposed a grammatical evolution [3] approach to automatically design ant colony optimization algorithms. The grammar adopted by this framework has the ability to guide the learning of novel architectures, by rearranging components regularly found on human designed variants. Furthermore, Taveres and Pereira [4] proposed a strongly typed genetic programming [5] approach to automatically evolve the communication mechanism that allows ants to cooperatively solve a given problem. For these two applications, results obtained with several TSP instances show that the evolved pheromone update strategies are effective and

exhibit a strong generalization capability and are competitive with human designed variants. For rule induction algorithms, Pappa and Freitas [6] proposed the use of grammar-based genetic programming (GGP) to automatically evolve rule induction algorithms. The experiments involving 11 data sets show that novel rule induction algorithms can be automatically generated using GGP. Oltean and Grosan [7] used multi expression programming (MEP) [8] technique to evolve evolutionary algorithm, in which each MEP chromosome encodes multiple EAs.

Although the aforementioned automatic algorithms have different emphases on research objectives and contents, one thing in common is that they use automatic method to design algorithms, which shows that automatic programming method can build algorithms to solve problems automatically.

As the core components of the evolutionary algorithms, the genetic operators, such as mutation and combination, are more variable and complicated compared with other components, such as initialization and selection, in the algorithm framework. Furthermore, in terms of the design of the new algorithms, the most difficult and important part of the previous work is focused on the design of genetic operators. In our work, we also focus on designing genetic operators. This paper proposes a novel approach to design genetic operators in evolutionary algorithm, namely, the evolutionary algorithm based on automatic designing of genetic operators (EA^{2}DGO), which uses MEP with a new encoding scheme [9] to automatically generate genetic operators in the evolutionary algorithm to solve simulated problems.

Organization of this paper is as follows. In Section 2, the commonality of three classical evolutionary algorithms is introduced and discussed and then in Section 3, the general scheme of designing genetic operators is presented. In Section 4, the framework of EA^{2}DGO is described, explaining the mechanism of automatically designing genetic operators. Experimental verifications are presented in Section 5. Section 6 gives conclusions and discussions.

2. Three Classical Evolutionary Algorithms

It is important to investigate what expressions of genetic operators are amenable to automatic design, for which we can get inspirations from analyzing the standard genetic algorithm (SGA) [10], particle swarm optimization (PSO) [11] and differential evolution (DE) [12, 13].

In classical GA's crossover arithmetic operator, the new vectors are generated by linear combination of two different individuals. PSO and DE can be considered extended algorithms of the SGA. In the operator of combination of PSO, the particle's personal experience *pbest* and population's best experience *gbest* influence the movement of each particle. In the common operator of mutation in DE algorithm, the new vector is generated by the difference of two individuals in population and sum with another individual according to certain rules. These three algorithms are different but share some common characteristics. In the following subsections, the equations describing the operations of the operators of SGA, PSO, and DE algorithms are analyzed in detail.

2.1. Genetic Algorithm. GA with the real coding usually adopts arithmetic crossover as one of the genetic operators. Take total arithmetic crossover, for example, assume N is a constant number which presents the size of population, and D is the dimension of parameter vectors. The population P is then expressed as $P = \{X_i(t),\ i = 1, 2, \ldots, N\}$ and t is the generation. Select two individuals $X_j(t)$, $X_k(t)$ from population according to certain rule, where $j, k \in \{1, 2, \ldots, N\}$, and $j \neq k$, the child vector $X^{(1)}(t)$ and $X^{(2)}(t)$ could be generated and expressed, respectively, as the following:

$$\begin{aligned} X^{(1)}(t) &= \alpha X_j(t) + (1-\alpha) X_k(t) \\ &= X_k(t) + \alpha\left(X_j(t) - X_k(t)\right), \end{aligned} \tag{1}$$

$$\begin{aligned} X^{(2)}(t) &= \alpha X_k(t) + (1-\alpha) X_j(t) \\ &= X_j(t) + \alpha\left(X_k(t) - X_j(t)\right), \end{aligned} \tag{2}$$

where $\alpha = \{\alpha_1, \alpha_2, \ldots, \alpha_D\}$ and $\alpha_l \in [0, 1]$, $l = 1, 2, \ldots, D$.

2.2. Particle Swarm Optimization. PSO, like other evolutionary algorithms, is also a population-based search algorithm and starts with an initial population of randomly generated solutions called particles. Each particle in PSO has a velocity and a position. PSO remembers both the best position found by all particles and the best positions found by each particle in the search process. For a search problem in a D dimensional space, a particle represents a potential solution. The velocity V_i^d and position X_i^d of the dth dimension of the ith particle are updated according to the following equations:

$$\begin{aligned} V_i^d(t+1) = V_i^d(t) &+ c_1 * \text{rand1}_i^d * \left(pbest_i^d(t) - X_i^d(t)\right) \\ &+ c_2 * \text{rand2}_i^d * \left(gbest^d - X_i^d(t)\right), \end{aligned} \tag{3}$$

$$X_i^d(t+1) = X_i^d(t) + V_i^d(t+1), \tag{4}$$

where $i = 1, 2, \ldots$, is the particle's index, $X_i = (X_i^1, X_i^1, \ldots, X_i^D)$ is the position of the ith particle, and $V_i = (V_i^1, V_i^1, \ldots, V_i^D)$ represents the velocity of ith particle. $pbest_i = (pbest_i^1, pbest_i^2, \ldots, pbest_i^D)$ represents the best previous position yielding the best fitness value for the ith particle. $gbest = (gbest^1, gbest^2, \ldots, gbest^D)$ is the best position discovered by the whole population. rand1_i^d and rand2_i^d are two random numbers independently generated within the range of $[0, 1]$, c_1 and c_2 are two learning factors reflecting the weights of stochastic acceleration terms that pull each particle toward *pbest* and *gbest* positions, respectively. $t = 1, 2, \ldots$, indicates the iterations.

2.3. Differential Evolution. Differential evolution (DE) is a population-based, direct, robust, and efficient search method.

Like other evolutionary algorithms, DE starts with an initial population vector randomly generated in the solution space. Assume that N is a constant number which presents the size of population, and D is the dimension of parameter vectors, and the population is expressed as $X_i(t)$, where $i = 1, 2, \ldots, N$, and t is the generation. The main difference between DE and other evolutionary algorithms, such as GA and PSO, is its new generating method to generate new population vectors. In order to generate a new population vector, three vectors in population are randomly selected and weighted difference of two of them is added to the third one. After crossover, the new vector is compared with a predetermined vector in the population. If the new vector is better than the predetermined one, it replaces it; otherwise, the predetermined vector is copied to the next generation's population. For a traditional DE, the mutation procedure is illustrated as the following.

For the ith vector from generation t, a mutant vector $X_i(t+1)$ is defined by

$$X_i(t+1) = X_{r_1}(t) + F\left(X_{r_2}(t) - X_{r_3}(t)\right), \tag{5}$$

where $\in \{1, 2, \ldots, N\}$, $r_1, r_2, r_3 \in [0, N]$, i, and r_1, r_2 and r_3 are different. The differential mutation parameter F, known as scale factor, is a positive real normally between 0 and 1 but can also take values greater than 1. Generally speaking, larger values for F result in higher diversity in the generated population and the lower values lead to faster convergence.

3. General Scheme of Designing Genetic Operators

3.1. The General Characteristics of Genetic Operators. Through the analysis above, the following observations are made.

(i) A genetic operator is a formula which is composed of a group of objects (such as $X_j(t)$ and $X_k(t)$), arithmetic operators (such as $+, -, *$), and parameters (such as α, c_1, c_2, F).

(ii) A formula representing a genetic operator can have many variants. For example, DE and PSO have similar but different formulas, which is again different from the formula of SGA.

(iii) While existing evolutionary algorithms (including SGA, PSO, and DE) have different formulas, their genetic operators actually share characteristics (i) and (ii).

According to the observations (i), (ii), and (iii), we could design a scheme to represent the genetic operator for automatic design.

3.2. The Scheme of Genetic Operators

3.2.1. The Encoding Scheme of Genetic Operator Chromosome. Firstly, we need an entity to express the genetic operators which can be distinguished and manipulated by a computer. Genetic programming (GP), gene expression programming (GEP), and multiexpression programming (MEP) are three kinds of methods with focus of generating computer programs automatically for given problems. According to this, a similar chromosome structure $O(i, C, t)$, like MEP, is presented to express genetic operators. Where $i = 1, 2, \ldots, N$ is the chromosome's index, t is the generation, and C is the genetic operator chromosome which is composed of a head and a tail. The head contains symbols that represent both functions (elements from the function set F) and terminals (elements from the terminal Set T), whereas the tail contains only terminals. Generally, F is composed of arithmetic operators, and T is composed of objects and parameters.

Each gene in C encodes a terminal or a function symbol. A gene that encodes a function includes pointers towards the function arguments. Function arguments always have indices of higher values than the position of the function itself in the chromosome.

There is little difference compared with chromosome represented by MEP. In MEP chromosome, the function arguments have indices of lower values than the position of function itself. However, both of them are essentially the same. MEP chromosome presented in this way is similar to the GEP chromosome where the tail is constructed with terminal symbol. Further information about relationships between chromosomes of GEP and MEP can be seen in [14].

3.2.2. The Decoding Scheme of Genetic Operator Chromosome. Chromosome translation is obtained by parsing the chromosome right-left. A terminal symbol specifies a simple expression. A function symbol specifies a complex expression obtained by connecting the operands specified by the argument positions with the current function symbol. As MEP chromosome encodes more than one problem solution, and there is neither practical nor theoretical evidence that one of these expressions is better than the others before fitness calculation. For simplicity, the expression tree expressed by the first symbol is chosen as the chromosome's final representation.

After decoding, a genotype of $O(i, C, t)$ could be translated into a phenotype $\overline{O}(i, C, t)$ which can be further processed by a computer.

3.2.3. The Characteristics of the Scheme of Designing Genetic Operators. According to the encoding and decoding scheme of genetic operators, two characteristics are essential for automatically designing genetic operators.

(i) Changeability: the most different characteristic compared with traditional genetic operators is that its structure could be reconstructed by a computer, which means that the genetic operators could be generated and changed according to the requirements of problem. For example, every gene could be changed into another terminal or function symbol; every function arguments could be changed into another function argument; when the gene or function arguments are changed, the genotype and phenotype of chromosome are transformed.

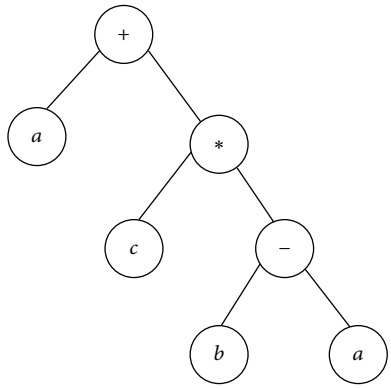

Figure 1: The expression tree of genetic operator (1) in SGA.

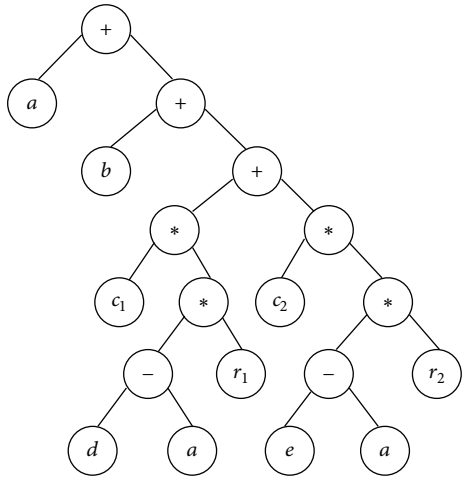

Figure 2: The expression tree of genetic operators (3) and (4) in PSO.

(ii) Adaptability: an automatic way of generating novel formula (using the existing objects, arithmetic operators, and parameters) may lead to a very novel design of evolutionary algorithm which can adapt itself to address problems with dynamic t, which means genetic operators are simultaneously searched and designed in the process of problem solving. In the following, we take a snapshot of the chromosomes used to represent the genetic operators of the three classic evolutionary algorithms.

3.3. The Chromosome for Three Classic Evolutionary Algorithms

3.3.1. SGA. If we consider (1) and take an example of $X_k(t) = a$, $X_j(t) = b$, $\alpha = c$, and $T = \{a, b, c\}$ and $O = \{+, -, *\}$, then (1) could be expressed by an expression tree (phenotype) as shown in Figure 1.

Take an example with the length of a chromosome as 7, $T = \{a, b, c\}$ and $O = \{+, -, *\}$, the expression tree in Figure 1 could be expressed as a genotype as shown in Table 1.

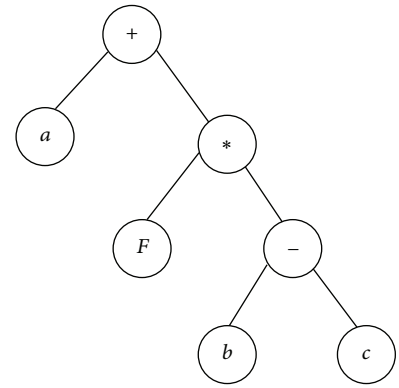

Figure 3: The expression tree of genetic operator (5) in DE.

Table 1: The equivalent genotype of (1) in SGA.

1	2	3	4	5	6	7
+	**a**	*	**c**	−	**b**	**a**
2, 3		4, 5		6, 7		

3.3.2. PSO. For PSO, an example of $X_i(t) = a$, $V_i(t) = b$, $V_i(t+1) = c$, $pbest_i(t) = d$, $gbest(t) = e$, $T = \{a, b, c, d, e, c_1, c_2, r_1, r_2\}$, and $O = \{+, -, *\}$ is used. The PSO's particle updating equation $X_i(t+1)$ can be expressed by an expression tree (phenotype) as shown in Figure 2.

Since (3) and (4) are more complicated equations compared with SGA operators, they need a longer chromosome to express the equivalent genotype. For this example, the length of chromosome is 18, $T = \{a, b, c, d, e, c_1, c_2, r_1, r_2\}$ and $O = \{+, -, *\}$, a kind of equivalent genotype for phenotype can be expressed as shown in Table 2.

3.3.3. DE. For the mutation operator of DE, if we take an example of $X_{r_1}(t) = a$, $X_{r_2}(t) = b$, $X_{r_3}(t) = c$, $T = \{a, b, c, F\}$, and $O = \{+, -, *\}$, the DE's mutation equation $X_i(t+1)$ could be expressed by an expression tree (phenotype) shown in Figure 3.

Here the length of chromosome is 7, $T = \{a, b, c, F\}$ and $O = \{+, -, *\}$. A kind of equivalent genotype for phenotype could be expressed as shown in Table 3.

By analysis above, it is evident that for SGA, PSO, and DE, their genetic operators can be expressed by a specific chromosome, respectively. But the *structures* of the chromosomes are not changed, or *static* through the whole process of evolutionary run for the three classic evolutionary algorithms. This is generally true also for many other variants of evolutionary algorithms.

Now the question is "can automatic programming method automatically construct genetic operators of evolutionary algorithms, where operators were automatically generated in the running process of problem solving, rather than predefined?" As we know, the evolutionary algorithms generally have capabilities of self-organizing, self-adapting, and self-regulating, but their genetic operators are normally predetermined. While the algorithms construed by predetermined operators are effective in certain aspects of problem

Table 2: The equivalent genotype of (3) and (4) in PSO.

1	2	3	4	5	6	7	8	9
+	+	+	*	*	*	*	−	−
18, 2	17, 3	4, 5	10, 6	11, 7	8, 12	9, 13	16, 18	14, 18
10	11	12	13	14	15	16	17	18
$\mathbf{c_1}$	$\mathbf{c_2}$	$\mathbf{r_1}$	$\mathbf{r_2}$	**e**	**c**	**d**	**b**	**a**

Table 3: The equivalent genotype of (5) in DE.

1	2	3	4	5	6	7
+	**a**	*	**F**	−	**b**	**c**
2, 3		4, 5		6, 7		

solving, their performances in addressing other issues may not be so competitive. This phenomenon can be explained to some extent by the famous "No Free Lunch" theory [15]. If there is an algorithm framework, in which the genetic operators can automatically adjust themselves during the problem-solving process with the change of the nature of the problems to be solved, the capabilities of self-organizing, self-adapting, and self-regulating of the algorithms can be further enhanced, and the limit imposed by the "NO Free Lunch" theory may be broken. Peng et al. proposed a population-based algorithm portfolio (PAP) [16], which distributes the time among multiple different algorithms, to decrease the inherent risk associated with the selection of algorithms. However, the algorithms are still predefined in PAP, which is very different compared with our method. In the following, the details of a framework that can automatically design genetic operators in the running process of problem solving will be introduced.

4. Evolutionary Algorithm Based on Automatic Designing of Genetic Operators (EA^{2}DGO)

In the framework of Evolutionary Algorithm based on Automatic designing of genetic operators (EA^{2}DGO), the genetic operators are not predefined by a designer before problem solving but are searched and designed in the process of problem solving. Thus, the framework of the EA^{2}DGO consists of two core components: one is the unit of problem solving (e.g., function optimization), which relates to operations in the problem solution space. The object of this unit is set to find global optimal solutions; and the other is the unit of automatically designing genetic operators, which relates to exploration in the space of genetic operators. The object of this unit is set to find the optimal genetic operators according to the requirement of the problem (see Figures 4(a), 4(b), and 4(c)). The unit of function optimization and the unit of automatically designing genetic operators are not isolated. Actually, they work together in a closely related way. In the unit of function optimization, the genetic operators for mutation are selected from the unit of automatically designing genetic operators in the process of problem solving. And in the unit of automatically designing genetic operators, individuals are selected from the function optimization population in the unit of function optimization for evaluating the performance of genetic operators.

The general framework of EA^{2}DGO is given in Algorithm 1.

The unit of function optimization focuses on the finding of global optimal solution, and the framework is given in Algorithm 2. According to the framework, we can see that the unit of function optimization is very similar with standard differential evolution, including the population initialization, crossover manipulation, individual fitness assessment, and individual selection. For mutation operator, an individual $\vec{o}_G^k$, $k \in [1, \mathrm{NOP}]$ is selected from the population of operator automatic generating unit according to the Roulette Wheel Selection algorithm, and the selected individual will be used as mutation operator in the unit of function optimization.

In the EA^{2}DGO, the mutation operator is a function model. The functionality of a function model is to provide a corresponding output (result) given a certain input parameters (terminals). The input parameters include $\vec{x}_{j,G}^{r_1}$, $\vec{x}_{j,G}^{r_2}$, $\vec{x}_{j,G}^{r_3}$, $\vec{x}_{j,G}^{\mathrm{best}}$, and $\vec{o}_G^k$. $\vec{x}_{j,G}^{r_1}$, $\vec{x}_{j,G}^{r_2}$, and $\vec{x}_{j,G}^{r_3}$ are three individuals selected from the population of function optimization unit randomly. $\vec{x}_{j,G}^{\mathrm{best}}$ is the best individual in the current population. Suppose all the function symbols in set O are binary operators, the calculation of the result of the function model is expressed as shown in Algorithm 3.

In the unit of function optimization, when the selected chromosome $\vec{o}_G^k$, namely, the genetic operator, can help function optimization unit generate a better candidate individual, the fitness value of chromosome $\vec{o}_G^k$ will be increased by one. In other words, the fitness of a chromosome in the unit of automatically designing genetic operators is measured by the times it has made positive effect for the unit of function optimization. With this measurement, in the next generation, the chromosomes with higher fitness will with a higher probability be selected as the genetic operators according to the Roulette Wheel Selection method (see Algorithm 4).

The evolution of chromosome in the unit of automatically designing genetic operators presents challenges in the general framework of the automation of EA. The proposed method is for the individual $\vec{o}_G^k$ as well as its offspring $\vec{o}_G'^k$ in the unit of automatically designing genetic operators, within a certain time (which is a parameter that can be set by the user); we repeatedly select individuals for mutation manipulation from the population of the unit of function optimization then count the times that the fitness of child becomes better than its parent, denoted by $\mathrm{ST}(\vec{o}_G^k)$ and $\mathrm{ST}(\vec{o}_G'^k)$, respectively. A

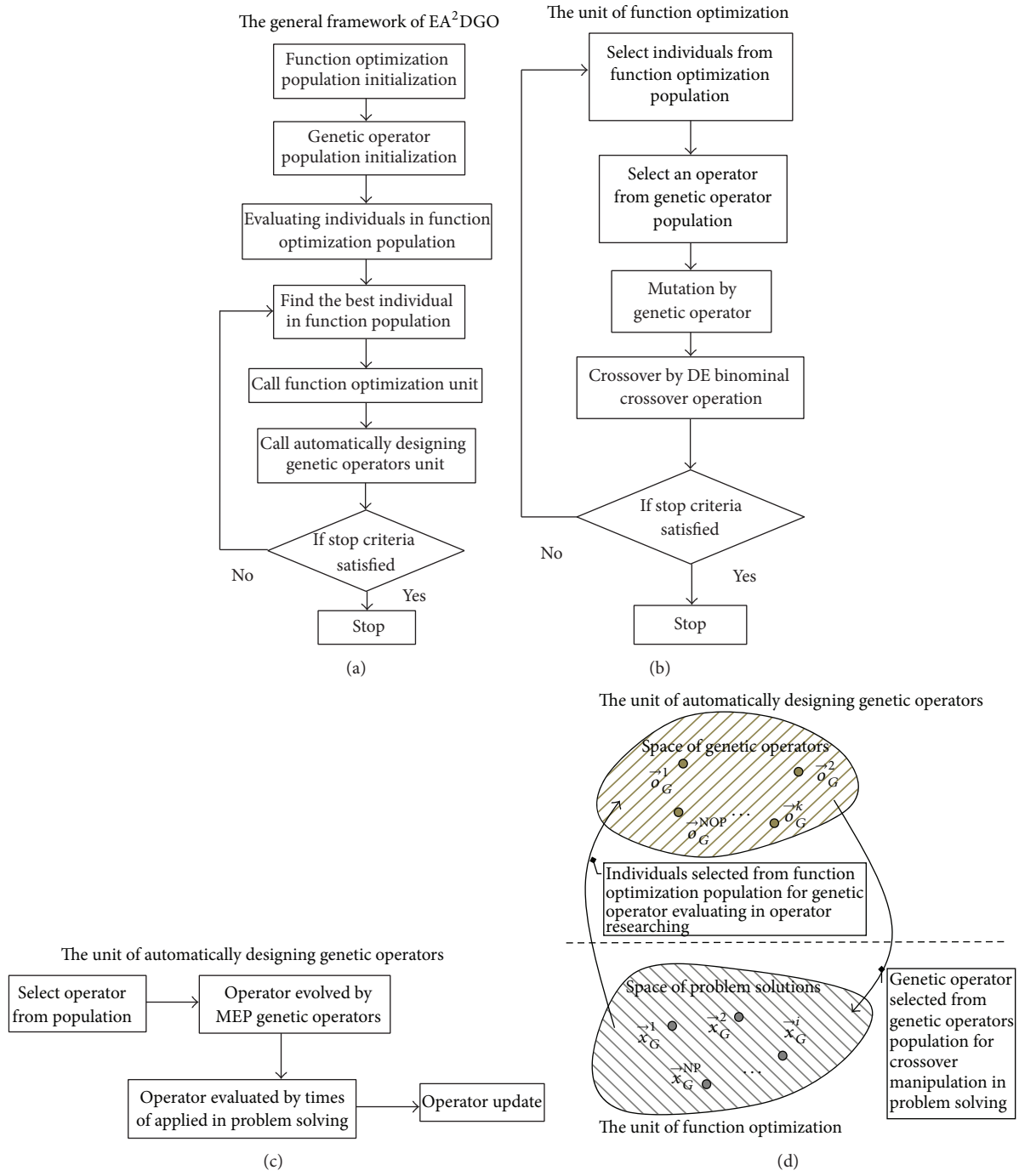

FIGURE 4: The components of EA²DGO. (a) The general framework of EA²DGO. (b) The flowchart of the function optimization unit. (c) The flowchart of the unit of automatically designing genetic operators. (d) The interaction relationship between two units.

Boolean function better is defined as the fitness for evolution of genetic operators:

$$\text{Better}\left(\vec{o}_G^{\,k}, \vec{o}_G^{\,\prime k}\right) = \begin{cases} \text{true}, & \text{if } \left(\text{ST}\left(\vec{o}_G^{\,k}\right) < \text{ST}\left(\vec{o}_G^{\,\prime k}\right)\right), \\ \text{false}, & \text{else.} \end{cases} \quad (6)$$

If Better $(\vec{o}_G^{\,k}, \vec{o}_G^{\,\prime k})$ is true, we think that the candidate $\vec{o}_G^{\,\prime k}$ is better than $\vec{o}_G^{\,k}$, and $\vec{o}_G^{\,k}$ is replaced by $\vec{o}_G^{\,\prime k}$.

The offspring $\vec{o}_G^{\,\prime k}$ is generated by MEP genetic manipulations [8]. While the traditional MEP has both crossover and mutation operators, only the mutation operator is used in EA²DGO for simplicity and saving computing time. Each symbol (terminal, function, and function pointer) in the

(1) **Begin**
(2) **Input**: NP, NOP, F, CR, OMR, Max_Fes, hl, tl, times, T, and O; where NP denotes the size of function optimization population; NOP denotes the size of operator generating population; F denotes scaling factor; CR denotes the probability of crossover; OMR denotes the probability of Mutation for operators in operator generating population; Max_FEs denotes the max number of function calls; hl is the of head length of chromosome in new encoding scheme MEP; tl is the tail length; times denotes the number of repeat times which randomly select individuals for mutation manipulation from population in unit of function optimization; T is the terminal symbol set; O is the function symbol set.
(3) $G = 0$
(4) Create the function optimization population $\vec{x}_G^i, \forall i, i = 1, \ldots, \mathrm{NP}$
(5) Create the genetic operators population $\vec{o}_G^k, \forall k, k = 1, \ldots, \mathrm{NOP}$
(6) $f\left(\vec{o}_G^k\right) = 0, \forall k, k = 1, \ldots, \mathrm{NOP}$
(7) Evaluate $f(\vec{x}_G^i), \forall i, i = 1, \ldots, \mathrm{NP}$
(8) **For** $G = 1$ to Max_FES **Do**
(9) Find the best $\vec{x}_G^{\mathrm{best}}$ in the function optimization population
(10) Call *the unit of function optimization*
(11) **If** (rand$[0, 1) <$ OMR) **Then**
(12) Call *the unit of automatically designing genetic operators*
(13) **End If**
(14) $G = G + 1$
(15) **End For**
(16) Output $\vec{x}_G^{\mathrm{best}}$
(17) **End**

Algorithm 1: The general framework of EA^{2}DGO.

(1) **Begin**
(2) Suppose the function optimization population is $\vec{x}_G^i, \forall i, i = 1, \ldots, \mathrm{NP}$
(3) Suppose the operator generating population is $\vec{o}_G^k, \forall k, k = 1, \ldots, \mathrm{NOP}$
(4) **For** $i = 1$ to NP **Do**
(5) Select randomly $r_1 \neq r_2 \neq r_3 \neq i, r_1, r_2, r_3 \in [1, \mathrm{NP}]$:
(6) Select $\vec{o}_G^k, k \in [1, \mathrm{NOP}]$ by Roulette Wheel Selection algorithm
(7) $j_{\mathrm{rand}} = \mathrm{randint}(1, D)$
(8) **For** $j = 1$ to D **Do**
(9) **If** (rand$_j[0, 1) <$ CR or $j = j_{\mathrm{rand}}$) **Then**
(10) $u_{j,G+1}^i = \mathrm{GeneCalculate}\left(\vec{x}_{j,G}^{r_1}, \vec{x}_{j,G}^{r_2}, \vec{x}_{j,G}^{r_3}, \vec{x}_{j,G}^{\mathrm{best}}, \vec{o}_G^k\right)$
(11) **Else**
(12) $\vec{u}_{j,G+1}^i = \vec{x}_{j,G}^i$
(13) **End If**
(14) **End For**
(15) **If** (better$(\vec{u}_{j,G+1}^i \vec{x}_{j,G}^i)$) **Then**
(16) $\vec{x}_{G+1}^i = \vec{u}_{G+1}^i$
(17) $f\left(\vec{o}_G^k\right) + +$
(18) **Else**
(19) $\vec{x}_{G+1}^i = \vec{x}_G^i$
(20) **End If**
(21) **End For**
(22) **End**

Algorithm 2: The framework of the unit function optimization.

chromosome may be a target of the mutation operator. When a symbol is changed, a new offspring is generated.

Although the evolution method motioned above can get access to the genetic operators before and after MEP genetic manipulation, an unavoidable disadvantage of this method is that it costs more computing resources compared with traditional optimization algorithms. The reason is that the genetic operators need to be evolved in the process of problem

(1) **Begin**
(2) Input: $\vec{x}_{j,G}^{r_1}, \vec{x}_{j,G}^{r_2}, \vec{x}_{j,G}^{r_3}, \vec{x}_{j,G}^{\text{best}}, \vec{o}_G^k = \{\vec{o}_{1,G}^k, \ldots, \vec{o}_{m,G}^k\}$, m is the length of Chromosome, $k = 1, \ldots,$ NOP.
(3) EL = 1
(4) **For** $i = 1$ to m **Do**
(5) **If** $(\vec{o}_{i,G}^k \in O)$ **Then**
(6) EL = EL + 2
(7) **End If**
(8) **End For**
(9) **For** $i =$ EL to 1 **Do**
(10)
$$\text{fitness}_i\left(\vec{o}_{i,G}^k\right) = \begin{cases} \vec{x}_{j,G}^{r_1}, & \text{if } \left(\vec{o}_{i,G}^k == \vec{x}^{r_1}\right) \\ \vec{x}_{j,G}^{r_2}, & \text{if } \left(\vec{o}_{i,G}^k == \vec{x}^{r_2}\right) \\ \vec{x}_{j,G}^{r_3}, & \text{if } \left(\vec{o}_{i,G}^k == \vec{x}^{r_3}\right) \\ \vec{x}_{j,G}^{r_{\text{best}}}, & \text{if } \left(\vec{o}_{i,G}^k == \vec{x}^{\text{best}}\right) \\ \text{rand}[0,1), & \text{if } \left(\vec{o}_{i,G}^k == F\right) \end{cases}$$
(11) **End For**
(12) Calculate the last $\vec{o}_{i,G}^l \in O, l \in [1, \text{EL}]$
(13) **For** $i =$ EL to 1 **Do**
(14) **If** $(\vec{o}_{i,G}^k \in O)$ **Then**
(15) $\text{fitness}_i\left(\vec{o}_{i,G}^k\right) = \text{fitness}_i\left(\vec{o}_{i,G}^{\text{EL}-1}\right) \Theta_{\vec{o}_{i,G}^k} \text{fitness}_i\left(\vec{o}_{i,G}^{\text{EL}}\right), \Theta_{\vec{o}_{i,G}^k} \in O$
(16) EL = EL − 2
(17) **End If**
(18) **End For**
(19) Return $\text{fitness}_1(\vec{o}_{1,G}^k)$
(20) **End**

Algorithm 3: The framework of gene calculation.

(1) **Begin**
(2) Calculate probability for each chromosome $p(\vec{o}_G^k) = f(\vec{o}_G^k) / \sum_{k=1}^{\text{NOP}} f(\vec{o}_G^k), k \in [1, \text{NOP}]$
(3) **For** $k = 1$ to NOP − 1 **Do**
(4) $P\left(\vec{o}_G^{k+1}\right) = p\left(\vec{o}_G^k\right) + p\left(\vec{o}_G^{k-1}\right)$
(5) **End For**
(6) **If** $(P(\vec{o}_G^k) \le \text{rand}[0,1) \le P(\vec{o}_G^{k+1}))$ **Then**
(6) $\vec{o}_G^k$ is selected for FunctionOptimization
(8) **End If**
(9) **Return** $\vec{o}_G^k$
(10) **End**

Algorithm 4: Roulette Wheel selection algorithm.

solving. The cost of time is influenced by two parameters times and OMR. Setting proper parameter values will reduce the cost of time and will be researched in the future work.

It is worthwhile to point outthat even though there is a mutation operator, respectively, in both the unit of automatically designing genetic operator and the unit of function optimization, their operations are very different. In the unit of automatically designing genetic operator, the operand/chromosome of the mutation operator is literally an expression tree. In contrast, the operand/chromosome of the mutation operator in the unit of function optimization is a vector of real values used as variables for the function optimization.

The general framework of the unit of automatically designing genetic operators is given in Algorithm 5.

5. Experimental Verification

Single-objective optimization problems are adopted to verify the validity of the EA^{2}DGO algorithm. This means that the genetic operators represented by the above scheme are

Table 4: The 23 test functions used in the experiments, X is the definition domain and $f_{\min}$ is the minimum values of the function.

Number	Test functions	n	X	$f_{\min}$
F1	$f(x) = \sum_{i=1}^{D} x_i^2$	30	$[-100, 100]$	0
F2	$f(x) = \sum_{i=1}^{D} \lvert x_i \rvert + \prod_{i=1}^{D} \lvert x_i \rvert$	30	$[-10, 10]$	0
F3	$f(x) = \sum_{i=1}^{D} \left(\sum_{j=1}^{i} x_j \right)^2$	30	$[-100, 100]$	0
F4	$f(x) = \max \{ \lvert x_i \rvert, 1 \le i \le D \}$	30	$[-100, 100]$	0
F5	$f(x) = \sum_{i=1}^{D-1} \left[100 \left(x_{i+1} - x_i^2 \right)^2 + (1 - x_i)^2 \right]$	30	$[-30, 30]$	0
F6	$f(x) = \sum_{i=1}^{D} \left(\lfloor x_i + 0.5 \rfloor \right)^2$	30	$[-100, 100]$	0
F7	$f(x) = \sum_{i=1}^{D} i x_i^4 + \text{random}[0, 1)$	30	$[-1.28, 1.28]$	0
F8	$f(x) = -\sum_{i=1}^{D} \left(x_i \sin \sqrt{\lvert x_i \rvert} \right)$	30	$[-500, 500]$	−12569.5
F9	$f(x) = \sum_{i=1}^{D} \left[x_i^2 - 10 \cos (2\pi x_i) + 10 \right]$	30	$[-5.12, 5.12]$	0
F10	$f(x) = -20 \exp \left(-0.2 \sqrt{\frac{1}{n} \sum_{i=1}^{D} x_i^2} \right) - \exp \left(\frac{1}{n} \sum_{i=1}^{D} \cos (2\pi x_i) \right) + 20 + e$	30	$[-32, 32]$	0
F11	$f(x) = \sum_{i=1}^{D} \frac{x_i^2}{4000} - \prod_{i=1}^{D} \cos \left(\frac{x_i}{\sqrt{i}} \right) + 1$	30	$[-600, 600]$	0
F12	$f(x) = \frac{\pi}{n} \left\{ 10 \sin^2 (\pi y_i) + \sum_{i=1}^{D-1} (y_i - 1)^2 \left[1 + 10 \sin^2 (\pi y_{i+1}) \right] + (y_n - 1)^2 \right\} + \sum_{i=1}^{D} u (x_i, 10, 100, 4), \quad y_i = 1 + \frac{1}{4} (x_i + 1)$ $u (x_i, a, k, m) = \begin{cases} k(x_i - a)^m, & x_i > a \\ 0, & -a \le x_i \le a, \\ k(-x_i - a)^m, & x_i < -a \end{cases}$	30	$[-50, 50]$	0
F13	$f(x) = 0.1 \left\{ \sin^2 (3\pi x_1) + \sum_{i=1}^{D-1} (x_i - 1)^2 \left[1 + \sin^2 (3\pi x_{i+1}) \right] + (x_n - 1)^2 \left[1 + \sin^2 (2\pi x_D) \right] \right\} + \sum_{i=1}^{D} u (x_i, 5, 100, 4)$	30	$[-50, 50]$	0
F14	$f(x) = \left[\frac{1}{500} + \sum_{j=1}^{25} \frac{1}{j + \sum_{i=1}^{2} (x_i - a_{ij})^6} \right]$	2	$[-65.536, 65.536]$	1
F15	$f(x) = \sum_{i=1}^{11} \left[a_i - \frac{x_1 (b_i^2 + b_i x_2)}{b_i^2 + b_i x_3 + x_4} \right]^2$	4	$[-5, 5]$	0.0003075
F16	$f(x) = 4x_1^2 - 2.1x_1^4 + \frac{1}{3} x_1^6 + x_1 x_2 - 4x_2^2 + 4_2^4$	2	$[-5, 5]$	−1.0316285
F17	$f(x) = \left(x_2 - \frac{5.1}{4\pi^2} x_1^2 + \frac{5}{\pi} x_1 - 6 \right)^2 + 10 \left(1 - \frac{1}{8\pi} \right) \cos x_1 + 10$	2	$[-5, 10] \times [0, 15]$	0.398
F18	$f(x) = \left[1 + (x_1 + x_2 + 1)^2 \left(19 - 14x_1 + 3x_1^2 - 14x_2 + 6x_1 x_2 + 3x_2^2 \right) \right] \times \left[30 + (2x_1 - 3x_2)^2 \left(18 - 32x_1 + 12x_1^2 + 48x_2 - 36x_1 x_2 + 27x_2^2 \right) \right]$	2	$[-2, 2]$	3
F19	$f(x) = -\sum_{i=1}^{4} c_i \exp \left[-\sum_{j=1}^{4} a_{ij} \left(x_j - p_{ij} \right)^2 \right]$	4	$[0, 1]$	−3.86

Table 4: Continued.

Number	Test functions	n	X	$f_{\min}$
F20	$f(x) = -\sum_{i=1}^{4} c_i \exp\left[-\sum_{j=1}^{6} a_{ij}\left(x_j - p_{ij}\right)^2\right]$	6	$[0, 1]$	−3.32
F21	$f(x) = -\sum_{i=1}^{5} \left[\left(x - a_i\right)\left(x - a_i\right)^T + c_i\right]^{-1}$	4	$[0, 10]$	−10
F22	$f(x) = -\sum_{i=1}^{7} \left[\left(x - a_i\right)\left(x - a_i\right)^T + c_i\right]^{-1}$	4	$[0, 10]$	−10
F23	$f(x) = -\sum_{i=1}^{10} \left[\left(x - a_i\right)\left(x - a_i\right)^T + c_i\right]^{-1}$	4	$[0, 10]$	−10

(1) **Begin**
(2) Input: $\vec{o}_G^k = \{\vec{o}_{1,G}^k, \ldots, \vec{o}_{m,G}^k\}$, $k \in [1, \text{NOP}]$;
(3) Select randomly $t, t \in [1, m]$:
(4) Select a new operator to $\vec{o}_{t,G}^k$ which will generate a new chromosome $\vec{o}_G'^k$ by MEP genetic operators.
(5) **For** $i = 1$ to times **Do**
(6) Select randomly $r_1 \neq r_2 \neq r_3 \neq r, r, r_1, r_2, r_3 \in [1, \text{NP}]$:
(7) $j_{\text{rand}} = \text{randint}(1, D)$
(8) **For** $j = 1$ to D **Do**
(9) **If** $(\text{rand}_j[0, 1) < \text{CR}$ or $j = j_{\text{rand}})$ **Then**
(10) $\vec{u}_{j,G}^r = \text{GeneCalculate}\left(\vec{x}_{j,G}^{r_1}, \vec{x}_{j,G}^{r_2}, \vec{x}_{j,G}^{r_3}, \vec{x}_{j,G}^{\text{best}}, \vec{o}_G^k\right)$
(11) $\vec{v}_{j,G}^r = \text{GeneCalculate}\left(\vec{x}_{j,G}^{r_1}, \vec{x}_{j,G}^{r_2}, \vec{x}_{j,G}^{r_3}, \vec{x}_{j,G}^{\text{best}}, \vec{o}_G'^k\right)$
(12) **Else**
(13) $\vec{u}_{j,G}^r = \vec{v}_{j,G}^r = \vec{x}_{j,G}^r$
(14) **End If**
(15) **End For**
(16) **If** $\vec{u}_{j,G}^r$ is better than $\vec{x}_{j,G}^r$ **Then**
(17) $\text{ST}\left(\vec{o}_G^k\right) + +$
(18) **End If**
(19) **If** $\vec{v}_{j,G}^r$ is better than $\vec{x}_{j,G}^r$ **Then**
(20) $\text{ST}\left(\vec{o}_G'^k\right) + +$
(21) **End If**
(22) **End For**
(22) **If** $(\text{Better}(\vec{o}_G^k, \vec{o}_G'^k))$ **Then**
(24) $\vec{o}_{G+1}^k = \vec{o}_G'^k$
(25) **Else**
(26) $\vec{o}_{G+1}^k = \vec{o}_G^k$
(27) **End If**
(28) **End**

Algorithm 5: The framework of the unit of automatically designing genetic operators.

used to manipulate the individuals in the population in the problem space, and the goal is to find the global optimal solutions of the problems. All the following functions listed in Table 4 are well-known benchmark functions which have been frequently used in literature [17]. All of these functions used in this paper are minimization problems. In order to verify the effectiveness and efficiency of EA^{2}DGO algorithm, we carried out experiments based on the 23 benchmark functions list in Table 4.

In our experiment, the parameters are set as in Table 5. Especially in T, a is $\vec{x}_{j,G}^{r_1}$, b is $\vec{x}_{j,G}^{r_2}$, c is $\vec{x}_{j,G}^{r_3}$, d is $\vec{x}_{j,G}^{\text{best}}$, and $F = \text{Random}(0, 1)$. In order to verify the performance of the proposed algorithm, standard DE algorithm is conducted. In DE algorithm, NP = 100, F = 0.7, CR = 0.8, and Max_FEs = 250000. The obtained results are presented in Table 5. Simulation was carried out in Eclipse and run on an AMD laptop with 2 G RAM under Windows XP platform. For each test problem, 25 independent runs were conducted with different random initializations.

The results on functions F1 to F23 are summarized in Table 6. For functions F1–F5, F7, and F9–F11, the EA^{2}DGO achieved better mean results than DE. For functions F6, F12–F14, and F16–F23, both algorithms obtain exactly the same mean results. DE achieved better mean results in function

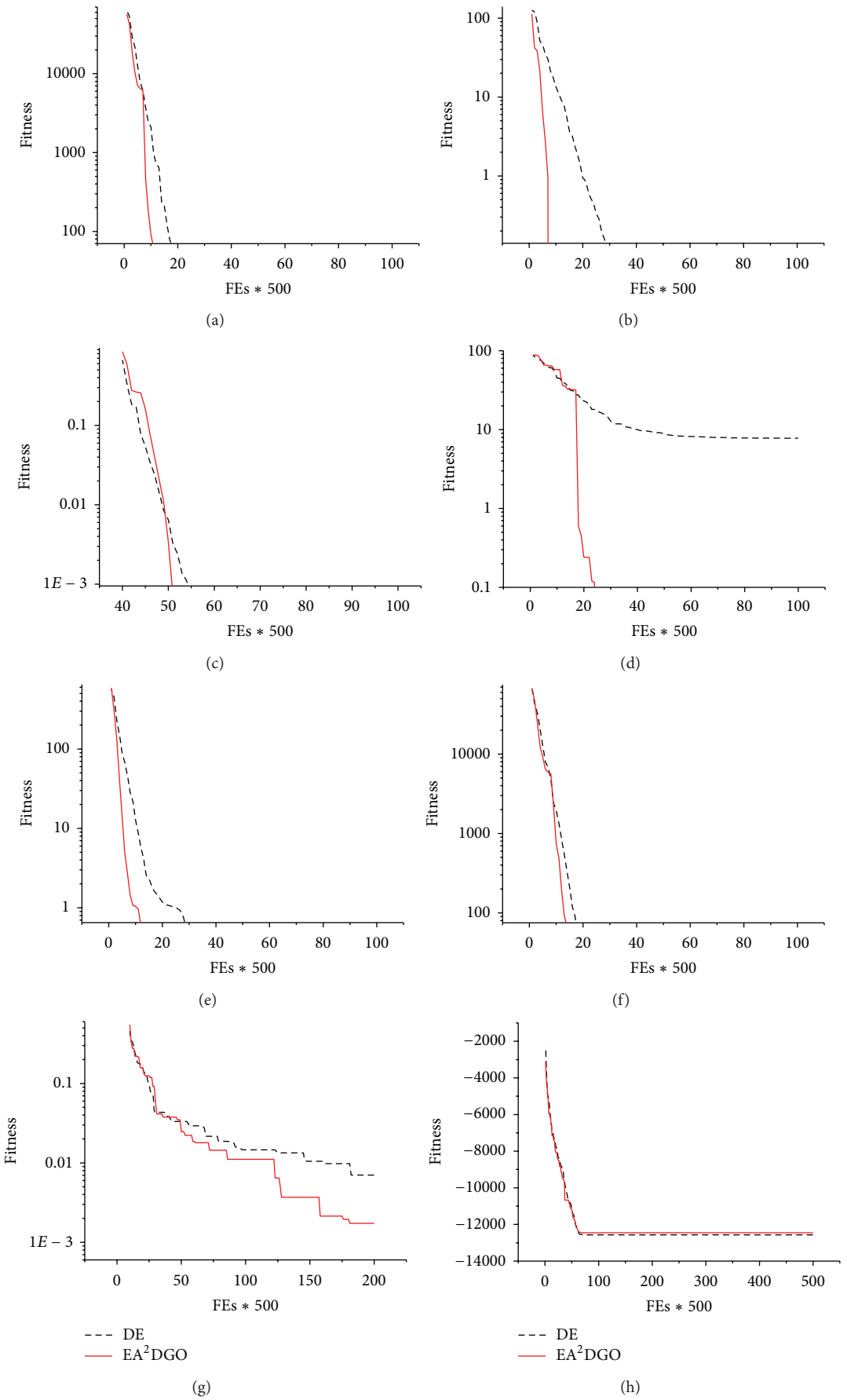

Figure 5: Continued.

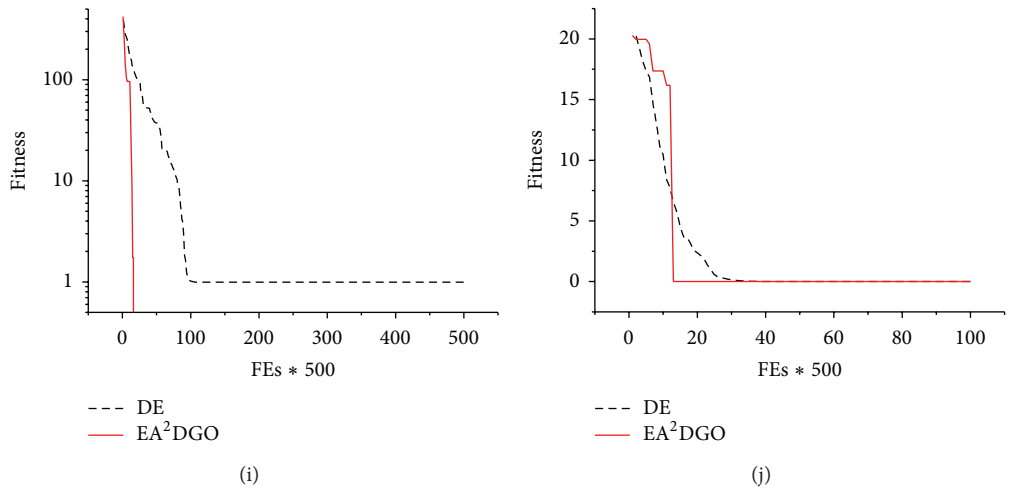

Figure 5: Convergence curves of the EA^{2}DGO and DE for partial test functions. x-axis represents number of function calls and y-axis represents the best fitness. (a) F1. (b) F2. (c) F3. (d) F4. (e) F5. (f) F6. (g) F7. (h) F8. (i) F9. (j) F10.

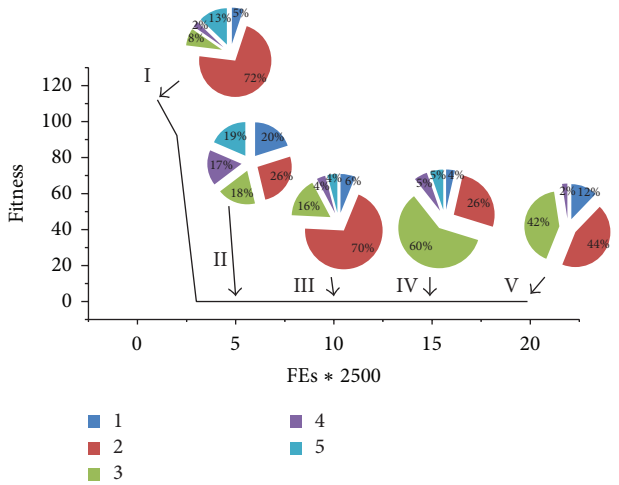

Figure 6: Phases I~V in one run for F7.

Table 5: Parameters setting used in EA^{2}DGO algorithm.

The unit of function optimization	NP	100
	CR	0.8
	Max_FEs	250000
The unit of automatically designing genetic operators	NOP	5
	h	5
	t	6
	L	11
	Times	50
	T	$\{a, b, c, d, F\}$
	O	$\{+, -, *\}$

F8 and F15. For F15, while the mean result of the EA^{2}DGO is close to that of DE, both algorithms have the ability to achieve the best solution. The biggest difficulty of EA^{2}DGO is from the function F8, for which DE can easily obtain the global minima but the new algorithm presented in this paper frequently falls into the local minima. The reason for this needs further investigation.

For the best results, the EA^{2}DGO achieved better results than DE in 8 functions and obtained the same best results in the reminded 15 functions. For the median results, the EA^{2}DGO achieved better results than DE in 7 functions and obtained the same median results in 14 functions. DE achieved better median results in functions F5 and F8. For the worst results, the EA^{2}DGO achieved better results than DE in 9 functions and obtained the same median results in 12 functions. DE achieved better median results in 2 functions, which are F8 and F15.

The convergence comparisons between the EA^{2}DGO and DE are shown in Algorithm 5. For simplicity, each Figure shows the result of a random trial. Because of space limitation, just some samples (F1~F10) are selected and presented. According to Figure 5, we can find that EA^{2}DGO converges more quickly than DE in every trail, except for F8 when the two algorithms converge almost through the same curve.

In order to better understand the internal operating mechanism of EA^{2}DGO, the following experiment is designed. Take F7 for example, the 50,000 function calls are shown in Figure 6, wherein the horizontal axis represents the number of calls, and the vertical axis represents the optimal solutions found so far. The 50,000 calls are divided into five stages of assessment (Phase I~V). Phase I represents the 0~50 calls, Phase II represents the 51~12,500 calls, Phase III represents 12,501~25,000 calls, Phase IV represents 25,001~37,500 calls, and Phase V is the last 37,501~50,000 calls. After 50,000 evaluations, the best optimal solution we found for F7 is $1.50E-6$. In Phase I, totally 39 effective evaluations are obtained, which means 39 new child candidates replace the parents.

Table 6: The results achieved for F1 to F23 using the EA^{2}DGO algorithm and DE algorithm.

	EA^{2}DGO					DE				
	Best	Median	Worst	Mean	Standard	Best	Median	Worst	Mean	Standard
F1	0.0	0.0	0.0	**0.0**	0.0	$2.38E-90$	$7.58E-89$	$3.90E-88$	$1.07E-88$	$9.22E-89$
F2	0.0	0.0	0.0	**0.0**	0.0	$3.07E-52$	$1.0E-51$	$6.34E-51$	$1.36E-51$	$1.17E-51$
F3	0.0	0.0	0.0	**0.0**	0.0	$9.84E-87$	$1.8E-85$	$6.8E-85$	$2.23E-85$	$1.7E-85$
F4	0.0	0.0	0.0	**0.0**	0.0	1.349	9.494	13.075	8.293	2.836
F5	0.0	22.117	29.0	**13.001**	10.839	1.810	18.356	71.343	23.886	14.486
F6	0.0	0.0	0.0	**0.0**	0.0	0.0	0.0	0.0	**0.0**	0.0
F7	$3.74E-5$	$8.61E-5$	$1.78E-3$	$\mathbf{8.82E-4}$	$4.26E-4$	$2.1E-3$	$3.3E-3$	$4.76E-3$	$3.3E-3$	$5.81E-4$
F8	−12569.487	−11858.856	−5897.496	−10864.79	1755.355	−12569.487	−12451.048	−12095.733	**−12419.463**	118.28
F9	0.0	0.0	0.0	**0.0**	0.0	0.0	1.99	6.96	2.12	1.32
F10	$4.441E-16$	$4.441E-16$	$4.441E-16$	$\mathbf{4.441E-16}$	0.0	$4.0E-15$	$7.55E-15$	$7.55E-15$	$5.89E-15$	$1.37E-15$
F11	0.0	0.0	0.0	**0.0**	0.0	0.0	0.0	$7.4E-3$	$4.93E-4$	$1.43E-3$
F12	$1.571E-32$	$1.571E-32$	$1.571E-32$	$\mathbf{1.571E-32}$	0.0	$1.571E-32$	$1.571E-32$	$1.571E-32$	$\mathbf{1.571E-32}$	0.0
F13	$1.35E-32$	$1.35E-32$	$1.35E-32$	$\mathbf{1.35E-32}$	0.0	$1.35E-32$	$1.35E-32$	$1.35E-32$	$\mathbf{1.35E-32}$	0.0
F14	0.998	0.998	0.998	**0.998**	$3.2E-8$	0.998	0.998	0.998	**0.998**	$8.6E-17$
F15	$3.075E-4$	$3.075E-4$	$1.791E-3$	$4.496E-4$	$1.024E-4$	$3.075E-4$	$3.075E-4$	$1.22E-3$	$\mathbf{3.69E-4}$	$1.77E-4$
F16	−1.03163	−1.03163	−1.03163	**−1.03163**	0.0	−1.03163	−1.03163	−1.03163	**−1.03163**	0.0
F17	0.398	0.398	0.398	**0.398**	$1.715E-4$	0.398	0.398	0.398	**0.398**	0.0
F18	3	3	3	**3**	$2.045E-14$	3	3	3	**3**	$6.88E-16$
F19	−3.86	−3.86	−3.86	**−3.86**	$2.66E-9$	−3.86	−3.86	−3.86	**−3.86**	$9.25E-13$
F20	−3.32	−3.32	−3.32	**−3.32**	$4.79E-7$	−3.32	−3.32	−3.32	**−3.32**	$6.27E-12$
F21	−10.153	−10.153	−10.153	**−10.153**	$1.01E-9$	−10.153	−10.153	−10.153	**−10.153**	$1.38E-15$
F22	−10.403	−10.403	−10.403	**−10.403**	$2.59E-9$	−10.403	−10.403	−10.403	**−10.403**	$1.38E-15$
F23	−10.536	−10.536	−10.536	**−10.536**	$1.03E-9$	−10.536	−10.536	−10.536	**−10.536**	$1.85E-15$

For each stage we use a pie chart to represent the success rate of every genetic operator in the population of genetic operators in the unit of automatically designing genetic operators (such as in Table 7, for first genetic operator in Phase I, the success rate is 5% and the totally effective evaluations are 39, which means that there are two useful candidates are generated by first genetic operator and then go into the population in the unit of function optimization).

In Phase I, it is obvious that the second genetic operator, which is $U^i_{G+1} = \vec{X}^{r_2}_G(F + \vec{X}^{r_2}_G)$ in operator generating population, produces more new candidates. In the beginning of evolution, the algorithm generates the new candidates easily and quickly, so we set a small number of function calls in Phase I. In Phase II, all the genetic operators have achieved comparable performance. In Phase III, the second operator demonstrates a stronger genetic capability; the phenotype of this genotype is $U^i_{G+1} = F\vec{X}^{r_2}_G + \vec{X}^{\text{best}}_G$ (see Table 7). In Phase VI, the best performance genotype is $U^i_{G+1} = \vec{X}^{\text{best}}_G - \vec{X}^{\text{best}}_G\vec{X}^{\text{best}}_G$. The common characteristics of these two formulas are that they both apply the current best individual in the population $\vec{X}^{\text{best}}_G$, which indicates that searching in vicinity of the best individual can improve the convergence efficiency. In Phase V, only 41 effective evaluations are achieved in 12,500 function calls, and the 4th and 5th genetic operators only produce one and zero effective candidate, respectively. This means that the searching of operators in the unit of automatically designing genetic operators gradually converges to those genetic operators that are hard to be further improved. The reason for that is, as mentioned above, the unit of function optimization and the unit of automatically designing genetic operators work together. When the function optimization population gradually converges to the global optimal solutions, the evolution of the genetic operator population is also led to a convergence.

According to the experiment, the genetic operator that produces more effective individuals does not necessarily produce the best individual. In this experiment, the first genetic operator in Phase V obtains the best performance in the 50,000 calls, whose phenotype is $U^i_{G+1} = \vec{X}^{r_1}_G - \vec{X}^{r_2}_G\vec{X}^{r_2}_G$.

6. Conclusions

A novel evolutionary algorithm based on automatic designing of genetic operators is presented to tentatively solve algorithm designed problem automatically. EA^{2}DGO does not only search in the problem solution space, but also explore in the space of genetic operators, which means that the genetic operators are not predefined by a designer before problem solving but searched and designed automatically in the process of problem solving.

Experimental results have shown that the EA^{2}DGO can achieve comparable performance with the state-of-art algorithm such as DE. However, many more issues deserve to be investigated for this potential algorithm. For example,

Table 7: The detail information in Phases I~V.

Phase	Formula	Success rate	Totally effective evaluations
I	$U_{G+1}^{i} = \vec{X}_G^{\text{best}} \vec{X}_G^{r_1} - \vec{X}_G^{r_1}$	5%	39
	$U_{G+1}^{i} = \vec{X}_G^{r_2} \left(F + \vec{X}_G^{r_2}\right)$	72%	
	$U_{G+1}^{i} = \vec{X}_G^{r_2}$	8%	
	$U_{G+1}^{i} = \left(\vec{X}_G^{r_1} \vec{X}_G^{r_1} - 2\vec{X}_G^{\text{best}}\right)\left(\vec{X}_G^{\text{best}} - \vec{X}_G^{r_1}\right)$	2%	
	$U_{G+1}^{i} = \vec{X}_G^{r_2} \left(\vec{X}_G^{r_1} - \vec{X}_G^{r_2}\right)$	13%	
II	$U_{G+1}^{i} = \vec{X}_G^{r_1} \vec{X}_G^{r_1} + \vec{X}_G^{\text{best}} \vec{X}_G^{\text{best}} \vec{X}_G^{r_1} - \vec{X}_G^{r_1} \vec{X}_G^{\text{best}}$	20%	434
	$U_{G+1}^{i} = \vec{X}_G^{r_1} + \vec{X}_G^{r_2} \vec{X}_G^{r_2}$	26%	
	$U_{G+1}^{i} = \vec{X}_G^{r_2} \vec{X}_G^{r_2} \vec{X}_G^{r_2} \vec{X}_G^{r_2}$	18%	
	$U_{G+1}^{i} = 6\vec{X}_G^{r_2} \vec{X}_G^{r_2} \vec{X}_G^{r_2}$	17%	
	$U_{G+1}^{i} = F\vec{X}_G^{r_2} + \vec{X}_G^{r_1} - \vec{X}_G^{r_1} \vec{X}_G^{r_1}$	19%	
III	$U_{G+1}^{i} = \vec{X}_G^{r_1} - \vec{X}_G^{r_2} \vec{X}_G^{r_2}$	6%	194
	$U_{G+1}^{i} = F\vec{X}_G^{r_2} + \vec{X}_G^{\text{best}}$	70%	
	$U_{G+1}^{i} = \vec{X}_G^{\text{best}} - \vec{X}_G^{r_1}$	16%	
	$U_{G+1}^{i} = 6\vec{X}_G^{r_2} \vec{X}_G^{r_2} \vec{X}_G^{r_2}$	4%	
	$U_{G+1}^{i} = \vec{X}_G^{\text{best}} + \vec{X}_G^{r_1} - \vec{X}_G^{r_1} \vec{X}_G^{r_1}$	4%	
IV	$U_{G+1}^{i} = \vec{X}_G^{r_1} - \vec{X}_G^{r_2} \vec{X}_G^{r_2}$	4%	57
	$U_{G+1}^{i} = -\vec{X}_G^{r_2} \vec{X}_G^{r_2}$	26%	
	$U_{G+1}^{i} = \vec{X}_G^{\text{best}} - \vec{X}_G^{\text{best}} \vec{X}_G^{\text{best}}$	60%	
	$U_{G+1}^{i} = F\vec{X}_G^{\text{best}}$	5%	
	$U_{G+1}^{i} = \vec{X}_G^{r_1} \vec{X}_G^{r_1}$	5%	
V	$U_{G+1}^{i} = \vec{X}_G^{r_1} - \vec{X}_G^{r_2} \vec{X}_G^{r_2}$	12%	41
	$U_{G+1}^{i} = \vec{X}_G^{\text{best}} \vec{X}_G^{r_2}$	44%	
	$U_{G+1}^{i} = \vec{X}_G^{r_3} - \vec{X}_G^{\text{best}}$	42%	
	$U_{G+1}^{i} = F\vec{X}_G^{\text{best}}$	2%	
	$U_{G+1}^{i} = \vec{X}_G^{r_1} \vec{X}_G^{r_1}$	0%	

at this stage, there are still quite a few parameters involved in the algorithm framework. As a result, parameter analysis becomes naturally a next step of study. Meanwhile, how to set up these parameters optimally (or even reduce some parameters) is an issue worth to be further studied. In addition, it is very interesting and desirable to investigate methods to define some important metrics for this new algorithm, such as evolvability, adaptability, and computational complexity.

Conflict of Interests

The authors declare that there is no conflict of interests regarding the publication of this paper.

Acknowledgments

The authors would like to thank the anonymous reviewers for their very detailed and helpful comments that help us to increase the quality of this work. This work was supported by Natural Science Foundation of Guangdong Province (no. S2013010013974), in part by the Shantou University National Foundation Cultivation Project (no. NFC13003), in part by the National Natural Science Foundation of China (no. 61175073) and in part by the Leading Talent Project of Guangdong Province.

References

[1] M. Dorigo and G. di Caro, "The ant colony optimization meta-heuristic," in *New Ideas in Optimization*, D. Corne, M. Dorigo, and F. Glover, Eds., pp. 11–32, McGraw-Hil, 1999.

[2] J. Taveres and F. B. Pereira, "Automatic design of ant algorithms with grammatical evolution," in *Proceedings of the 15th European Conference on Genetic Programming*, pp. 206–217, 2012.

[3] C. Ryan, J. J. Collins, and M. O'Neill, "Grammatical evolution: evolving programs for an arbitrary language," in *Genetic Programming*, vol. 1391 of *Lecture Notes in Computer Science*, pp. 83–96, 1998.

[4] J. Taveres and F. B. Pereira, "Designing pheromone update strategies strongly typed genetic programming," in *Proceedings of the 14th European Conference on Genetic Programming (EuroGP '11)*, pp. 85–96, 2011.

[5] J. R. Koza, *Genetic Programming: On the Programming of Computers by Means of Natural Selection*, MIT Press, Cambridge, Mass, USA, 1992.

[6] G. L. Pappa and A. A. Freitas, "Towards a genetic programming algorithm for automatically evolving rule induction algorithms," in *Proceedings of the ECML/PKDD 2004—Workshop on Advances in Inductive Learning*, pp. 93–108, 2004.

[7] M. Oltean and G. Grosan, "Evolving evolutionary algorithms using multi expression programming," in *Advances in Artificial Life*, vol. 2801 of *Lecture Notes in Computer Science*, pp. 651–658, 2003.

[8] M. Oltean and D. Dumitrescu, "Multi expression programming," Tech. Rep. UBB-01-2002, Babes-Bolyai University, Cluj-Napoca, Romania, 2002, http://www.mep.cs.ubbcluj.ro/.

[9] C. Ferreira, "Gene expression programming: a new adaptive algorithm for solving problems," *Complex Systems*, vol. 13, no. 2, pp. 87–129, 2001.

[10] Z. J. Pan, L. S. Kang, and Y. P. Chen, *Evolutionary Computation*, Tsinghua University, Beijing, China, 1998.

[11] J. Kennedy and R. Eberhart, "Particle swarm optimization," in *Proceedings of the IEEE International Conference on Neural Networks*, pp. 1942–1948, IEEE Press, December 1995.

[12] R. Storn and K. Price, "Differential evolution—a simple and efficient heuristic for global optimization over continuous spaces," *Journal of Global Optimization*, vol. 11, no. 4, pp. 341–359, 1997.

[13] K. V. Price, R. M. Storn, and J. A. Lampinen, *Differential Evolution-A Practical Approach to Global Optimization*, Springer, Berlin, Germany, 2005.

[14] D. Jiang, Z. Wang, H. Sun, and Y. Du, "A unified fitness calculation method for automatic modeling algorithms," in *Proceeding of the 8th World Congress on Intelligent Control and Automation (WCICA '10)*, pp. 1569–1573, Jinan, China, July 2010.

[15] D. H. Wolpert and W. G. Macready, "No free lunch theorems for optimization," *IEEE Transactions on Evolutionary Computation*, vol. 1, no. 1, pp. 67–82, 1997.

[16] F. Peng, K. Tang, G. Chen, and X. Yao, "Population-based algorithm portfolios for numerical optimization," *IEEE Transactions on Evolutionary Computation*, vol. 14, no. 5, pp. 782–800, 2010.

[17] X. Yao, Y. Liu, and G. Lin, "Evolutionary programming made faster," *IEEE Transactions on Evolutionary Computation*, vol. 3, no. 2, pp. 82–102, 1999.