Journal of Computers Vol. 28, No. 3, 2017, pp. 93-107
doi:10.3966/199115592017062803008

A Hybrid Genetic Algorithm Based on Variable Grouping and Uniform Design for Global Optimization

Xuyan Liu, Yuping Wang, and Haiyan Liu

School of Computer Science and Technology, Xidian University
Xi'an, 710071, Shaanxi, China
lucyxu107@163.com, ywang@xidian.edu.cn, hyliu83@126.com

Received 29 May 2016; Revised 05 December 2016; Accepted 05 December 2016

Abstract. In this paper, we propose a hybrid genetic algorithm based on variable grouping and uniform design for global optimization problems, a function formula based grouping (FBG) strategy is adopted to classify the separable variables into different groups and put the interactive variables into the same group. In this way, the problem considered can be changed into several lower dimension sub-problems. The solution can be more easily obtained by simultaneously solving these sub-problems. Then, an efficient crossover operator is designed by using a specific uniform design method. When we have no prior knowledge on global optimal solution, this crossover operator has more possibility to find high quality solutions. Furthermore, in order to enhance the diversity and efficient explore the search space, an adapted mutation operator is design to adaptively adjust the search scope, and a local search scheme is proposed to speed up the search. By integrating all these schemes, a hybrid genetic algorithm is proposed for global optimization problems. Finally, the experiments are conducted on widely used benchmarks and the results indicate the proposed algorithm is efficient and effective.

Keywords: genetic algorithm, hybrid algorithm, uniform design, variable grouping

1 Introduction

Optimization problems in science and engineering are getting more and more attention in past years, and many real-world problems in different domains can be formulated as the optimization of a continuous function. The Genetic Algorithm (GA) is very popular in solving optimization problems, and it has been applied in many domains, such as pattern recognition, neural network, timetabling problem [22, 24, 28] and so on. GA develops very fast since it was first proposed by J. Holland for global optimization in real-world applications. The widely used evolutionary algorithms such as Particle Swarm Optimization (PSO) [10], Bee Colony Algorithm (ABC) [6] and Differential Evolution (DE) [26] can all be seen as variants of GA.

Like other evolutionary algorithms (EAs), GA is a very simple but competitive search technique for solving global optimization problems. GA repeats crossover, mutation, and selection operators by generation to evolve its solution toward the global optimum. With the development of various modern intelligent algorithms, an increasing tendency in merging GA and other heuristic methods is upward. As a result, much more attentions have been focused on the design of various hybrid genetic algorithms.

However, there are two major challenges for GA in solving global optimization problems. One is the premature convergence and the other is the slow convergence speed. Premature means the algorithm can be easily trapped in local minima and the slowly convergent speed means that GA needs lots of function evaluations (FEs) to find the global optimal solution. Many works focus on these two problems have been done to improve the performance of GA [11, 22, 28, 30, 31], such as combining with other schemas [12], adding other search method into GA [17], and so on. Despite many success stories, classical GA often lose their efficacy and advantages when they are applied to large and complex problems such as those with high dimension and search space, and the reason for this phenomenon is not only there are lots of local optimal solutions, but also the population might lapse into the local space s. In this paper we focus on the following three points.

Firstly, how to generate a good initial population and design efficient genetic operators? There are a variety of population initialization techniques such as random population generation by pseudo-random number generators, the population generated by uniform experimental design such as orthogonal design [11, 33], and by hybrid and multi-step technique [7] etc. For more information about the variety of population initialization techniques and the analysis of its effect, please refer to [4, 8-9]. Among these methods, the uniform design is a powerful tool to generate the initial population, which can make the EAs more robust and statistically sound, and orthogonal design and Latin square [5, 11-12, 30] are two famous uniform design methods in EAs. In literature [11], the authors applied the quantization technique and orthogonal design to enhance the crossover operator and generate the initial population, and they got good results. Compared to orthogonal design, Latin square is more easily and has the similar property as orthogonal design. It can help the algorithm to generate points scattering uniformly over a specific domain. In literature [12], the authors used Latin square to improve the evolutionary algorithm. Wang and Dang proposed an efficient evolutionary algorithm based on level set and Latin square [30]. Except for the initial population generation, the uniform design method can also be used to design efficient mutation and crossover operators. In this paper, we improve the population initialization method by combining permutation Latin square design and feasible solution division technique. The diversity of the population will increase by dividing feasible solution and using permutation Latin square design method. Also, we use the permutation Latin square method to design a crossover operator. Furthermore, an adaptive mutation operator is designed.

Secondly, how to design an efficient local search mechanism? It has been proven that the local search method can greatly improve the performance of GA. In literature [29, 37], the authors integrated local search method into GA to further improve the search performance. Other authors [17, 21, 36] also combined the specific local search or self-adaptive local search methods into GAs to enhance their performances. DFP Quasi-Newton method [3] is an efficient local search method to get the local optimal solution, but it requires that the function is differentiable and has to compute the derivatives of the function. In this paper, we use a modified Quasi-newton method as the local search method, in which the derivatives are not required.

Thirdly, to further enhance the performance of the algorithm, the variable decomposition method is used. When the dimension of the problem increases, the performance of GA will drop dramatically [15]. This is often referred to the curse of dimensionality. Potter and De Jong made the first attempt to solve this problem by divide and conquer strategy and proposed the cooperative co-evolutionary (CC) framework [20]. The main idea of CC is to divide the n dimensional variables into m sub-groups, and thus decompose the original problem into several lower dimensional sub problems. Then these sub-problems will be evolved separately. This procedure is called grouping or decomposition. However, CC is very sensitive to grouping strategies. If related (interacted) variables are not grouped together, the performance of CC deteriorates dramatically [25]. So the key point is how to decompose the problem. Many grouping strategies have been successively proposed such as those in [2, 14, 16, 18-19, 31-32, 34-35]. The formula based grouping (FBG) strategy [31-32] does the decomposition by analysing the formula expression, so it can correctly identify the variable interaction and decompose the problem into several sub groups. We adopt FBG strategy to decompose the problem before optimization to make the optimization more effective.

In this paper, we propose a hybrid genetic algorithm based on variable grouping and uniform design. More specifically, we have the following research objectives:

(1) We use the uniform design method called permutation Latin square to generate the initial population and design the crossover operator, so that the initial population and the offspring can evenly distribute in the search space.

(2) To improve the efficiency of the proposed algorithm, we introduce a local search algorithm to the genetic algorithm.

(3) We design a new selection operator and adaptive mutation operator to enhance the diversity of population.

(4) A formula based grouping (FBG) strategy is adopted to divide the problem into several lower dimensional sub-problems to improve the performance of GA.

The remainder of this paper is organized as follows: Section 2 gives a review of variable interaction problems and describes the adopted variable grouping strategy; The genetic operators and the local search scheme are introduced in Section 3; Section 4 gives the framework of the proposed algorithm for global optimization problems with continuous variables; The experiments are conducted in Section 5; Finally, Section 6 concludes this paper and makes the discussions.

2 Variable Grouping Strategy

This section defines the notion of non-separable variables and provides various grouping methods and a description of CC and formula based grouping (FBG) strategy.

2.1 Cooperative Co-evolutionary

We consider the following global optimization problem:

$$\begin{cases} \min f(\mathbf{x}) \\ s.t\ \mathbf{l} \le \mathbf{x} \le \mathbf{u} \end{cases} \tag{1}$$

Where $\mathbf{x} = (x_1, x_2, ..., x_N)$ is a vector in R^N, $f(\mathbf{x})$ is the objective function, $\mathbf{l} = (l_1, l_2, ..., l_N)$, and $\mathbf{u} = (u_1, u_2, ..., u_N)$ with $x_i = [l_i, u_i]$ for $i = 1 \sim N$.

As the dimension of the problem increases, the problem solving will become more difficult and the function evaluations of a GA will increase dramatically. To reduce the difficulty of the problem solving and the function evaluations, we can use a grouping method to divide all the variables into several groups and evolve them one by one with an optimization algorithm. The CC is proposed for this purpose [20]. The critical steps of CC can be summarized as follows: 1) Problem decomposition into several subcomponents: a decomposition method is used to divide the *N*-dimensional decision vector into groups $G_1 ... G_m$ of variables. Each group is optimized with a separate subpopulation of the corresponding dimension $|G_i| < N$. 2) Each subcomponent optimization independently: an optimizer is applied to the population for optimizing the decision variables in the current group. 3) Solution combination of the subcomponents: in order to evaluate the fitness of the individuals from a certain subpopulation, a representative element from each of the other subpopulations is selected. In this cooperation step, a population completed with *N*-dimensional candidate solutions is constructed by concatenating the representatives to each element of the current subpopulation. In the conventional CC framework, optimizing a group with the corresponding subpopulation is called a phase.

2.2 Decision Variable Interactions

CC is an effective method for solving large-scale optimization problems. This effectiveness is attributed to the decomposition of a large-scale problem into a set of smaller subproblems. However, one drawback of CC is that its performance is sensitive to the choice of decomposition strategy. Here, we give the notion of variable interactions and review various decomposition strategies. The formal definition of separability and non-separability according to [27] is as follows:

Definition 1 A function $f(x_1, x_2, ..., x_N)$ is separable iff

$$arg\ min_{(x_1, x_2, ..., x_N)} f(x_1, x_2, ..., x_N) = \left(arg\ min_{x_1} f(x_1, ...), arg\ min_{x_2} f(x_2, ...), ..., arg\ min_{x_2} f(..., x_N)\right) \tag{2}$$

The decomposition strategy that identifies interacting decision variables and divides the search space into subspaces of lower dimensionality is the most important component of CC algorithms. A function *f* is separable if Equation 2 holds, and its global optimum can be reached by successive line search along the axes. Therefore, if a certain function is not separable, there must be interactions between at least two variables in the decision vector. In other words, if it is possible to find the global optimum of a function by optimizing one dimension at a time regardless of the values taken by other dimensions, then the function is said to be separable. It is nonseparable otherwise.

2.3 Formula Based Grouping

The ideal goal of problem decomposition is that the decision variables in one subcomponent are non-separable, and the decision variables in any two subcomponents are separable. Recently, grouping strategies have been proposed extensively, such as random grouping [34], MLCC [35], variable interaction learning grouping (CCVIL [2]), route distance grouping [16], differential grouping [18], delta grouping [19], and so on. However, these strategies still cannot accurately group the variables. For example, random grouping method randomly shuffles the indices of the variables to obtain a different grouping at the beginning of each cycle and it uses a constant group size, and randomly decomposes the high-dimensional variable vector into several sub groups. These are then optimized with a certain EA. The problem of random grouping is that the best group size is not known in advance. MLCC adopts a multilevel strategy for decomposition. It maintains a decomposer pool from which decomposers with different group sizes are selected depending on the problem under investigation and the stage of the evolution. For nonseparable problems, MLCC tends to select the decomposers with large group sizes. However, determining a good pool of decomposers is hard in practice since the interaction between variables is usually not known beforehand. CCVIL needs lots of fitness evolutions and so on.

To overcome this shortcoming, the formula based grouping (FBG) strategy [31-32] is adopted to decompose the problem in this paper. FBG can identify the separable variables and the interacting variables, and can correctly group the variables into the several independent groups. Each group can be optimized independently. The FBG framework is as follows: First, building a set *N-Sep* whose elements are the factors determining variables interaction; Second, matching these elements to the test problem; finally, if an element is matched, putting the variables connected by the element into a group. Note that a general objective function consists of the finite number of four arithmetic operations '$+$', '$-$','$\times$' and '$\div$' and composite operations of basic elementary functions (e.g. exponential functions a^x, logarithmic functions $\log ax$). Based on these, a set *N-Sep* is constructed as follows:

(1) Variables separability in four arithmetic operations. If function $f(x) = x_1 + x_2 + ... + x_m$ then each x_i; where i =1 ~m, it is obvious that the operators'$+$'and '$-$' do not affect the variables separability. So in this function the variables can be optimized independently, and thus the variables $x_{1,}, x_2, ..., x_m$ in this function are separable. While if a function contains '$\times$' and '$\div$' of two variables, these two variables can not be optimized independently and thus they are non-separable. We put '$\times$' and '$\div$' into a set *N -Sep*.

(2) Variables separability in a composite function. For a basic elementary function $g(h(x))$, the variables in x are separable if $g(y)$ is monotone and the variables in $h(x)$ are separable, for example, $g(y) = e^y$; $h(x) = x_1 + x_2$; $g(h(x))$, it's obvious that x_1 and x_2 is separate ; otherwise, variables in $g(h(x))$ are non-separable. Therefore, put the non-monotonic functions (e.g. trigonometric functions, inverse trigonometric functions, etc.) of the basic elementary functions into *N -Sep*.

(3) Some special circumstances about the variables separability. From item (1), we know that the variables linked by operations '$\times$' or '$\div$'are non-separable, nevertheless, if two functions are exponential functions, and the variables in exponential function are separable, then the variables in '$\times$' or '$\div$'of two exponential functions are separable. Put '$\times$' and '$\div$' into a set *N -Sep*. (except that both composite functions are exponential functions).

From these three cases, the set *N -Sep* can be constructed, and the variables can be classified into different groups by the following Algorithm 1.

Algorithm 1 The flow of the formula based variable grouping strategy FBG

```
Step1. The elements in N -Sep. can be seen as strings (such as $\cos$ and $\sin$ );
Step2. Using regular expressions to match these strings in N-Sep for each problem;
Step3. If a string is matched from the problem, then these variables contained in this string will be
       classified into a group, and so on;
Step4. If some variables are not matched, then they are separable, and each variable is put into a group.
```

According to Algorithm 1, the variables can be classified into several independent groups, and each group can be optimized independently. We adopt FBG strategy in our paper to divide the original problem into several sub problems to make the optimization easier and more efficient. Thus the original problem can be divided into several independent sub-problems. For detailed information, please refer to literature [31, 32].

3 Genetic Operators and Local Search Scheme

The genetic algorithm follows the general procedure of an evolutionary algorithm [1]. The initial population P_G which consists of NP individuals $X_{i,G}$, $i=1,2,...,NP$ where G is the current generation. Then, crossover, mutation, and selection operations are employed, and the above process is repeated until a termination criterion is reached.

3.1 Crossover Operator Based on Permutation Latin Square

After FBG strategy is used to divide the *N*-dimensional vectors into several groups, the crossover operator will work on every independent group and get several potential offspring. To design the crossover, we first introduce the concept of permutation Latin square. For any $(x_1,x_2,...,x_q)\in R^q$, a shift mapping $S:R^q\to R^q$ is defined as

$$S(x_1,x_2,...,x_q)=(x_2,x_3,...,x_q,x_1) \tag{3}$$

Definition 2 A $q\times q$ matrix is called a permutation Latin square of order q if the matrix satisfies the following conditions:

(I).Its first row $a=(x_1,x_2,...,x_q)$, denoted as V_1, is a permutation of $(1,2,...,q)$.

(II).Its $i-th$ row, denoted as V_i, is given by $V_i=S(V_{i-1})$ for $i=2\sim q$.

A permutation Latin square of order q with a being its first row is denoted as $Ls(q.a)$, and its $i-th$ row and the $j-th$ column element is denoted as $v_{i,j}$. Permutation Latin-square design is one of the uniform design methods ([5]) which have been widely used in many application problems. It can generate points uniformly scattered in a domain. We want to make use of the permutation Latin square to design an efficient crossover operator. Note that the crossover operator is mainly focused on exploration, i.e., searching for the good solution near their parents. Keep this in mind, when two parents are given, it is better for us to find a way which has the more possibility to generate better offspring near their parents by using crossover operator. How to realize it? Since we do not know initially where the optimal solution is, it is reasonable to generate the offspring which scatter near their parents as uniformly as possible. We use permutation Latin-square to design a crossover operator as follows. Suppose that the domain considered is given by

$$[L,U]=\{x\in R^q \mid l_j\le x_j\le u_j, j=1\sim q\} \tag{4}$$

Where x_j, l_j and u_j are the $j-th$ component of x, L and U, respectively. Choose a row vector $a\in R^q$ and generate a permutation Latin square of order q, $Ls(q.a)=(v_{ij})_{q\times q}$. Then permutation Latin-square generates a set of uniformly scattered points in $[L,U]$ as follows:

$$\{W^i \mid W^i=(w_{i1},w_{i2},...,w_{in}), i=1\sim q\} \tag{5}$$

Where $w_{ij}=l_j+\dfrac{2v_{ij}-1}{2n}(u_j-l_j)$, i, $j=1\sim q$. For any two parents $X^i=(x_{i1},x_{i2},...,x_{in})$, $x=1,2$, define

$$\overline{L}=(l_1,l_2,...,l_n),\ \overline{U}=(u_1,u_2,...,u_n) \tag{6}$$

With

$$l_j=\min\{x_{1j},x_{2j}\},\ u_j=\max\{x_{1j},x_{2j}\},\ j=1\sim n \tag{7}$$

and define a domain

$$\left[\overline{L},\overline{U}\right]=\left\{x\in R^n \left|_j \le x_j \le u_j, j=1\sim n\right.\right\} \tag{8}$$

Choose a proper Latin square $Ls\left(q.a\right)=v_{ij}$ of order q . If $q\ge n$, the *i-th* offspring

$$O^i=\left(o_{i1},o_{i2},...,o_{in}\right) \tag{9}$$

is generated by

$$o_{ij}=l_j+\frac{2v_{ij}-1}{2q}\left(u_j-l_j\right), j=1\sim n \tag{10}$$

for $i=1\sim q$. If $q<n$, divide the components of X^1 , X^2 , $\overline{L}$ and $\overline{U}$ into q sub-vectors in the same way as follows:

$$X^1=\left(A^1,A^2,...,A^q\right),\ X^2=\left(B^1,B^2,...,B^q\right) \tag{11}$$

$$\overline{L}=\left(L^1,L^2,...,L^q\right),\ \overline{U}=\left(U^1,U^2,...,U^q\right) \tag{12}$$

Where A^j , B^j , L^j and U^j are sub-vectors of the same dimension and the dimensions of q sub-vectors are randomly determined. The $i-th$ offspring

$$O^i=\left(O^i{}_1,O^i{}_2,...,O^i{}_q\right) \tag{13}$$

is generated by

$$O^i{}_j=L_j+\frac{2v_{ij}-1}{2q}\left(U_j-L_j\right), j=1\sim q \tag{14}$$

for $\mathrm{i}=1\sim \mathrm{q}$.

Example 1. We choose $\mathrm{q}=4$ and the first row of Latin square $\mathrm{a}=\left(1,2,3,4\right)$. Then the Latin square of order 4 is

$$Ls\left(4,a\right)=\begin{bmatrix}1&2&3&4\\2&3&4&1\\3&4&1&3\\4&1&3&2\end{bmatrix}$$

Suppose that $n=4$ and two parents are $X^1=\left(0,0,0,0\right)$ and $X^2=\left(1,1,1,1\right)$, respectively. Then we have $q\ge n$. In this case, we obtain from Eq. (6) that

$$\overline{L}=\left(0,0,0,0\right),\overline{U}=\left(1.1.1.1\right)$$

and we obtain from Eq. (9) and Eq. (10) that four offsprings are

$$O^1=\frac{1}{8}\left(1,3,5,7\right)$$

$$O^2=\frac{1}{8}\left(3,5,7,1\right)$$

$$O^3=\frac{1}{8}\left(5,7,1,3\right)$$

$$O^4=\frac{1}{8}\left(7,1,3,5\right)$$

Suppose that $n=6$ and two parents are $X^1=(0,0,0,0,0,0)$ and $X^2=(1,1,1,1,1,1)$, respectively. Then we have
$q<n$. In this case, we obtain from Eq. (6) that

$$\overline{L}=(0,0,0,0.0,0),\overline{U}=(1.1.1.1,1,1)$$

Suppose that X^1, X^2, $\overline{L}$ and $\overline{U}$ are randomly divided into 4 sub-vectors as follows:

$$X^1=(A^1,A^2,A^3,A^4),X^2=(B^1,B^2,B^3,B^4)$$
$$\overline{L}=(\mathrm{L}^1,\mathrm{L}^2,,\mathrm{L}^3,\mathrm{L}^4),\overline{U}=(U^1,U^2,U^3,U^4)$$

Where

$$\mathrm{L}^1=A^1=(0,0,0),\mathrm{L}^2=A^2=0,\mathrm{L}^3=A^3=0,,\mathrm{L}^4=A^4=0,$$
$$U^1=B^1=(1,1,1),U^2=B^2=1,U^3=B^3=1,,U^4=B^4=1.$$

Then we obtain from Eq. (13) and Eq. (14) that four offsprings are

$$O^1=\frac{1}{8}(1,1,1,3,5,7)$$
$$O^2=\frac{1}{8}(3,3,3,5,7,1)$$
$$O^3=\frac{1}{8}(5,5,5,7,1,3)$$
$$O^4=\frac{1}{8}(7,7,7,1,3,5)$$

3.2 Adaptive Mutation Operator

The purpose of mutation is to increase the diversity of population and avoid the premature. In conventional mutation process [11, 30], the GA chooses the mutation offspring with an given possibility p_m, and they didn't consider the diversity of the population. In this paper, we propose an improved adaptive mutation operator based on the diversity on every dimension.

The main idea is that when the numerical variance of some dimensions is relatively small, it may mean the diversity at these dimensions is very low and it is necessary to increase the diversity of these dimensions. The algorithm selects several individuals from the current population $P(k)$ and crossover offspring set $O(k)$ with mutation rate p_m for mutation.

The algorithm framework is shown in Algorithm 2, and the meaning of the notations used in the Algorithm 2 is as follows:

M The total number of individuals selected to take part in mutation;

Q The number of individuals in $P(k)\cup O(k)$;

N The dimension of the problem;

D : A parameter of positive integer which means D dimensions will undergo the mutation, where $D<N$.

$d=(d_1,d_2,...,d_D)$: determine the dimensions $d_1,d_2,...,d_D$ will undergo the mutation. For $N>30$ we choose $D=5$ and for $N>30$ we choose $D=10$;

Algorithm 2 Adaptive Mutation Operator

```
Choose individuals for mutation;
for $i=1$ to $Q$ do
```

```
    if $rand < p_m$ then
        The $i^{th}$ individual is selected for mutation;
    end if
end for Suppose $M$ individuals have been chosen to take part in mutation.
for $j = 1$ to $N$ do
    Calculate the j-th dimensional mean value $m(j)$ and standard deviation $std(j)$ of $P(k) \cup O(k)$;
end for
Let $d = (d_1, d_2, ..., d_D)$ denote the indices of the $D$ dimensions with the smallest $std$ values;
For each chosen individual, its $d_j - th$ dimension value $Z(d_j)$ will be changed by mutation to
```

$$\begin{cases} Z(d_j) = Z(d_j) - \left(Z(d_j) - l(d_j)\right) * r, & if\ Z(d_j) < m(j) \\ Z(d_j) = Z(d_j) + \left(u(d_j) - Z(d_j)\right) * r, & else \end{cases} \tag{15}$$

for $j = 1 \sim D$.

3.3 Local Search Scheme

To speed up the convergence, we introduce a local search scheme DFP Quasi-newton method into our algorithm. DFP Quasi-newton method is an efficient local search method to get the local optimal solutions, but it requires that the function is differentiable and has to compute the derivatives of the function. In this paper, we propose a modified the Quasi-newton method in which the derivatives are not required. We use difference quotient to estimate the partial derivative as follows:

$$\frac{\partial f(x)}{\partial x_i} \approx \frac{f(x + H_i) - f(x - H_i)}{2h} \tag{16}$$

where $H_I = (0, ..., 0, \text{H}, ..., 0)$ with h being the $i - th$ component. We used the modified DFP Quasi-Newton method as the local search method to improve the convergence speed of the algorithm. The modified DFP Quasi-Newton method is shown in Algorithm 3.

Algorithm 3 The local search method

```
Step1. Choose an initial point $x^0$ and an initial positive definite matrix $H^0 = I^n$; $H = H^0$ and searching
        precision remarked as $eps$;
Step2. If $\left\|g\left(x^k\right)\right\| \le eps$ eps then algorithm stops, the optimal solution is set to $x^k$, otherwise, go to
       step3.
Step3. Set the search direction $P = -Hg\left(x^k\right)$;
Step4. Calculate the step length l by
                        $\min_{\lambda > 0} f\left(x^k + \lambda P(k)\right)$
       and set $x^{k+1} = x^k + \lambda P$;
Step5. If $k + 1 = n$, let $H = H^0$; $k = 0$, otherwise, set
```

$$H = H + \frac{\left(x^{k+1} - x^k\right)\left(x^{k+1}\right)^T}{\left(x^{k+1} - x^k\right)^T\left(g^{k+1} - g^k\right)} - \frac{H^k\left(g^{k+1} - g^k\right)\left(g^{k+1} - g^k\right)^T H^k}{\left(g^{k+1} - g^k\right)^T H^k \left(g^{k+1} - g^k\right)} \tag{17}$$

```
       Go to step3.
```

4 A Hybrid Genetic Algorithm

In this section we propose a hybrid genetic algorithm based on variable grouping and uniform design for global optimization problems. The permutation Latin square method is used in the initialization of the population and crossover operator to enhance the search capability of the algorithm. Also, we adopt FBG grouping strategy in the crossover process.

4.1 Space Division and Population Initialization

In this paper, we use the Latin square $L_s(N,a)$ to generate evenly scattered points in the solution space. When the feasible solution space is large, it may be desirable to generate more potential chromosomes for a better coverage, so we divide the solution space into S subspaces, where S is the design parameter. We choose the $s-th$ dimension such that

$$u_s - l_s = max\{u_i - l_i\} \qquad \textbf{(18)}$$

Where $i = 1,2,...,N$; Then in each subspace, N points are generated evenly. Thus the total $N \times S$ points will be generated. The best NP points among these $N \times S$ points form the initial population.

The following steps of Algorithm 4 and Algorithm 5 show the space division process and initialization.

Algorithm 4 Divide Solution Space

```
Step1. Select the $s-th$ dimension, which satisfies us $u_s - l_s = max\{u_i - l_i\}$, where $i = 1,2,...,N$ ;
Step2. Cut the feasible region into $S$ domains along dimensions $[L(1),U(1)],[L(2),U(2)],...,[L(S),U(S)]$
       where the lower and upper bounds of the i-th domain along dimension s are
```

$$\begin{cases} l_s(i) = l_s + (i-1)\left(\dfrac{u_s - l_s}{S}\right) \\ u_s(i) = u_s + i\left(\dfrac{u_s - l_s}{S}\right) \end{cases} \qquad i = 1,2,...,\mathrm{N} \qquad \textbf{(19)}$$

The lower and upper bounds in any domain along any other dimension j are still l_j and u_j

Algorithm 5 Population initialization

```
Step1. Use algorithm 4 to divide the solution space to $S$ sub spaces such that $NP < N \times S$, where $NP$ is
       the population size;
Step2. Use Latin square $L_s(N,a)$ of order $N$ to generate $N$ points in each sub-space according to the
       method in crossover operator;
Step3. Among the $N * S$ points, select $NP$ best points as the initial population.
```

4.2 The Proposed Hybrid Genetic Algorithm

In this section, we present the proposed hybrid genetic algorithm. We generate a good initial population with chromosomes and use permutation Latin square method to generate good initial population, and then evolve and improve the population iteratively. Besides, we apply the proposed crossover and the adaptive mutation operator to generate a set of potential offspring in iteration. Then we select the point with the smallest function values from these potential offspring and the parents to form the next generation. The overall algorithm is as Algorithm 6.

Algorithm 6 A hybrid genetic algorithm based on FBG and Latin square

```
Set the initial iteration number k=0 and the maximal number of generations $Max\_gen$ ;
Use algorithm 5 to generate the initial population $P(k)$ ;
**While** $k < Max\_gen$ **do**
```

```
    Randomly select some individuals from current population with probability $p_c$. These points together
with the best point of the population will take part in crossover and mutation procedure;
    Use the crossover described in section 3.1 to generate the set of offspring $O(k)$;
    Generate the set of offspring $M(k)$ by using mutation algorithm 2;
    Select NP chromosomes from $P(k) \cup O(k) \cup M(k)$ with the smallest function value to form the
next generation.
    Identify the best point $new\_best$;
    Execute the local search algorithm 3 by using the current best point as the initial point to get a new
point $x'$;
    Set $new\_best = x'$;
    $k = k + 1$;
end While
When the stop criterion is satisfied, output the current best individual;
```

5 Experiments

5.1 Test Functions

We execute the proposed algorithm to solve the following test functions:

$$f_1 = \sum_{i=1}^{N} \left(-x_i \sin\sqrt{|x_i|} \right)$$

$$f_2 = \sum_{i=1}^{N} \left(x_i^2 - 10\cos(2\pi x_i) + 10 \right)$$

$$f_3 = -20\exp\left(-0.2\sqrt{\frac{1}{N}\sum_{i=1}^{N} x_i^2} \right) - \exp\left(\frac{1}{N}\sum_{i=1}^{N} \cos(2\pi x_i) \right)$$

$$f_4 = \frac{1}{4000}\sum_{i=1}^{N} x_i^2 - \prod_{i=1}^{N} \cos\left(\frac{x_i}{\sqrt{i}} \right) + 1$$

$$f_5 = \sum_{i=1}^{N} i x_i^2$$

$$f_6 = \sum_{i=1}^{N} \left(\sum_{j=1}^{i} x_j \right)^2$$

$$f_7 = \sum_{i=1}^{N} x_i^2$$

$$f_8 = \sum_{i=1}^{N} x_i^4 + Random\left[0,1\right)$$

$$f_9 = \sum_{i=1}^{N} |x_i| + \prod_{i=1}^{N} |x_i|$$

Table 1 lists the basic characteristics of these test functions.

Table 1. Basic characteristics of these test functions

Test function	Feasible space	Global minima	Dimension
f_1	$[-500,500]^N$	-12569.5	30
f_2	$[-5.12,5.12]^N$	0	30
f_3	$[-32,32]^N$	0	30
f_4	$[-600,600]^N$	0	100
f_5	$[-10,10]^N$	0	30
f_6	$[-100,100]^N$	0	30
f_7	$[-100,100]^N$	0	30
f_8	$[-1.28,1.28]^N$	0	30
f_9	$[-10,10]^N$	0	30

5.2 Control Experiment

We adopt the following parameter values:

(1) We want to take more points distributed evenly in the feasible region at the beginning, and divide the search space into S sub-domain, where $S=100$. Use the Latin square $L_s(N,a)$ with $a=(1,2,...,N)$ to generate N points in each sub-domain;

(2) The population size: $NP=50$;

(3) Crossover probability $p_c=0.2$; mutation probability $p_m=0.12$; the order $q=10$ Latin square is used in crossover process. The number of dimensions chosen to take part in mutation is

$$D=\begin{cases}10 & \text{N=100}\\ 5 & \text{N=30}\end{cases}$$

(4) Stop criterion: When the best solution cannot be improved further in 20 successive generations after 100 generations.

5.3 Results and Comparison

We performed 30 independent runs for each algorithm on each test function and recorded: (1) the mean number of function evaluations (FEs), (2) the mean function value (Mean), (3) the best value (Best), (4) the worst value (Worst) and (5) the standard deviation of the function values (Std).

Table 2 shows the performance of the proposed hybrid genetic algorithm based on formula based grouping method (FBG-GA). We see that the mean function values are equal or close to the optimal ones, and the standard deviations of the function values are relatively small.

In Table 3, we compare the performance of the proposed algorithm FBG-GA with orthogonal learning cuckoo search algorithm (OLCS) [13] and social learning particle swarm optimization algorithm (SL-PSO) [23]. It can be seen from Tables 3, that the difference between the mean solution of our algorithm and the mean solution of each of the compared algorithms is significant (especially for f1and f5). The algorithm we proposed and the compared algorithms reach a similar level of performance for the most functions, but our algorithm uses less number of function evaluations. The above comparisons and analysis indicate that proposed algorithm can yield higher mean solution quality than both of the compared algorithms. In addition, our algorithm requires a smaller mean number of function evaluations than the compared algorithms. Furthermore, it gives small standard deviations of function values, and hence has more stable solution quality.

Table 2. The results

Test function	Function evaluation	Mean	Std	Worst	Best	Global minima
f_1	25349	-12569.49	$5.3065e-02$	-12569.48	-12569.49	-12569.5
f_2	28051	0	0	0	0	0
f_3	31206	$8.8818e-16$	0	$8.8818e-16$	$8.8818e-16$	0
f_4	26699	0	0	0	0	0
f_5	29609	0	0	0	0	0
f_6	27012	$-0.2499e-01$	$1.2748e-04$	$-0.2497e-01$	-0.2500	0
f_7	26708	0	0	0	0	0
f_8	25502	$7.9095e-03$	$3.3672e-03$	$1.4443e-02$	$4.4149e-03$	0
f_9	28389	0	0	0	0	0

Table 3. Comparison of SL-PSO, OLCS and the proposed algorithm FBG-GA, where the results for SL-PSO and OLCS are from [13, 23] personally

Test function	Algorithm	SL-PSO	OLCS	FBG-GA
f_1	Mean	-	-12514	-12569.49
	Std	-	75.7934	$5.3065e-02$
	FEs	-	900000	25349
f_2	Mean	$1.55e+01$	0	0
	Std	$3.19e+00$	0	0
	FEs	200000	300000	28051
f_3	Mean	$5.51e-15$	$8.8817e-16$	$8.8818e-16$
	Std	$5.51e-15$	0	0
	FEs	200000	150000	31206
f_4	Mean	0	0	0
	Std	0	0	0
	FEs	200000	200000	26699
f_5	Mean	-	$5.0749e-05$	0
	Std	-	$3.3339e-05$	0
	FEs	-	300000	29609
f_6	Mean	$4.66e-07$	$5.5768e-147$	$-0.2499e-01$
	Std	$2.48e-07$	$1.4262e-146$	$1.2748e-04$
	FEs	200000	500000	27012
f_7	Mean	$4.34e-90$	$1.5686e-107$	0
	Std	$5.26e-90$	$4.5543e-107$	0
	FEs	200000	150000	26708
f_8	Mean	-	$5.0749e-05$	$7.9095e-03$
	Std	-	$3.3339e-05$	$3.3672e-03$
	FEs	-	300000	25502
f_9	Mean	$1.50e-46$	-	0
	Std	$5.34e-47$	-	0
	FEs	200000	-	28389

6 Conclusions

We have developed a hybrid genetic algorithm based on FBG grouping strategy and Latin square design for global numerical optimization with continuous variables. Our objective has been to apply the Latin square design to enhance the GA so that it can be more statistically sound and robust. In particular, we apply Latin squares and FBG grouping to design a new crossover operator, so that the problem with high dimension can be changed into several lower dimension sub-problems. Besides, we propose a new mutation operator based on variance. Furthermore, we add the local search in the GA to speed up the algorithm. The proposed algorithm is executed to solve 9 benchmark test problems, and we compare the results to SL-PSO and OLCS. The results indicate that it can find optimal or close-to-optimal solutions for these test functions.

However, there are also two main shortages in our work, one is all problems considered in this study are unconstrained optimization issues, the other is FBG can't be used in black-box problems. Future work may include the extension of the proposed method for large-scale constraint optimization, and the proposed algorithm will be tested on more test functions and higher dimensions. Besides, adding other techniques into FBG-GA and applying the proposed algorithm into real-world problems is also an interesting topic.

Acknowledgement

This work is supported by National Natural Science Foundation of China (No.61472297 and No.U1404622).

References

[1] R. B. Agrawal, K. Deb, K. Deb, R. B. Agrawal, Simulated binary crossover for continuous search space, Complex Systems 9(1995) 115-148.

[2] W. Chen, T. Weise, Z. Yang, K. Tang, Large-Scale global optimization using cooperative coevolution with variable interaction learning, in: R. Schaefer, C. Cotta, J. Kolodziej, G. Rudolph (Eds.), PPSN XI, Part II, LNCS 6239, Berlin, Springer-Verlag, 2010, pp. 300-309.

[3] W. C. Davidon, Variable metric method for minimization, SIAM Journal on Optimization 1(2)(1991) 1-17.

[4] A. de Perthuis, de Laillevault, B. Doerr, C. Doerr, Money for nothing: speeding up evolutionary algorithms through better initialization, in: Proc. the 2015 Annual Conference on Genetic and Evolutionary Computation, 2015.

[5] K. Fang, Y. Wang, Number-Theoretic Methods in Statistics, Champman & Hall, London, 1994.

[6] D. Karaboga, An idea based on honey bee swarm for numerical optimization, [Technical report] Turky: Erciyes University, 2005.

[7] B. Kazimipour, X. Li, A. Qin, Initialization methods for large scale global optimization, in: Proc. 2013 IEEE Congress on Evolutionary Computation (CEC), 2013.

[8] B. Kazimipour, X. Li, A. Qin, A review of population initialization techniques for evolutionary algorithms, in: Proc. 2014 IEEE Congress on Evolutionary Computation (CEC), 2014.

[9] B. Kazimipour, X. Li, A. Qin, Effects of population initialization on differential evolution for large scale optimization, in: Proc. 2014 IEEE Congress on Evolutionary Computation (CEC), 2014.

[10] J. Kennedy, R. Eberhart, Particle swarm optimization, IEEE International Conference in 1995 on Neural Networks 4 (1995) 1942-1948.

[11] Y.-W. Leung, Y. Wang, An orthogonal genetic algorithm with quantization for global numerical optimization, Proceedings of IEEE Transactions on Evolutionary Computation 5(1)(2001) 41-53.

[12] D.-G. Li, C. Smith, A new global optimization algorithm based on Latin Square theory, in: Proc. IEEE International Conference in 1996 on Evolutionary Computation, 1996.

[13] W. J. Y. M. Li, Xiangtao, Enhancing the performance of cuckoo search algorithm using orthogonal learning Method,

Neural Computing and Applications 24(6)(2013) 1233-1247.

[14] X. Li, X. Yao, Cooperatively coevolving particle swarms for large scale optimization, IEEE Transactions on Evolutionary Computation 16(2)(2012) 210-224.

[15] Y. Liu, X. Yao, Q. Zhao, T. Higuchi, Scaling up fast evolutionary programming with cooperative coevolution, in: Proc. the 2001 Congress on Evolutionary Computation, 2001.

[16] Y. Mei, X. Li, X. Yao, Cooperative co-evolution with route distance grouping for large-scale capacitated Arc routing problems, IEEE Transactions on Evolutionary Computation 18(3)(2014) 435-449.

[17] D. Molina, M. Lozano, F. Herrera, MA-SW-Chains: Memetic algorithm based on local search chains for large scale continuous global optimization, in: Proc. 2010 IEEE Congress on Evolutionary Computation (CEC), 2010.

[18] M. Omidvar, X. Li, Y. Mei, X. Yao, Cooperative co-evolution with differential grouping for large scale optimization, Evolutionary Computation, IEEE Transactions on 18(3)(204) 1378-393.

[19] M. Omidvar, X. Li, X. Yao, Cooperative co-evolution with delta grouping for large scale non-separable function optimization, in: Proc. 2010 IEEE Congress on Evolutionary Computation, 2010.

[20] M. A. Potter, K. A. D. Jong, Cooperative coevolution: an architecture for evolving co-adapted subcomponents, Evolutionary Computation 8(2000) 1-29.

[21] A. Qin, P. Suganthan, Self-adaptive differential evolution algorithm for numerical optimization, in: Proc. 2005 IEEE Congress on Evolutionary Computation Evolutionary Computation, 2005.

[22] R. Raghavjee, N. Pillay, A comparison of genetic algorithms and genetic programming in solving the school timetabling problem, in: Proc. 2012 Fourth World Congress on Nature and Biologically Inspired Computing (NaBIC), 2012.

[23] Y.J.R. Cheng, A social learning particle swarm optimization algorithm for scalable optimization, Information Sciences: an International Journal 291(C)(2015) 43-60.

[24] T.-S. Yan, An improved genetic algorithm and its blending application with neural network, in: Proc. 2010 2nd International Workshop on Intelligent Systems and Applications (ISA), 2010.

[25] R. Salomon, Reevaluating genetic algorithm performance under coordinate rotation of benchmark functions: a survey of some theoretical and practical aspects of genetic algorithms, Biosystems 39(3)(1995) 263-278.

[26] R. Storn, K. Price, Differential evolution- a simple and efficient heuristic for global optimization over continuous spaces, Journal of Global Optimization 11(4)(1997) 341-359.

[27] K. Tang, X. Li, P. N. Suganthan, Z. Yang, T. Weise, Benchmark functions for the CEC'2010 special session and competition on large-scale global optimization, GENE 7(33)(2009) 8.

[28] T.A. Tarique, M.A. Zamee, M.I. Khan, A new approach for pattern recognition with neuro-genetic system using microbial genetic algorithm, in: Proc. 2014 International Conference on Electrical Engineering and Information Communication Technology (ICEEICT), 2014.

[29] J.-T. Tsai, T.-K. Liu, J.-H. Chou, Hybrid Taguchi-genetic algorithm for global numerical optimization, Proceedings of IEEE Transactions on Evolutionary Computation 8(4)(2004) 365-377.

[30] Y. Wang, C. Dang, An evolutionary algorithm for global optimization based on level-set evolution and Latin squares, Proceedings of IEEE Transactions on Evolutionary Computation Evolutionary Computation 11(5)(2007) 579-595.

[31] F. Wei, Y. Wang, T. Zong, A novel cooperative coevolution for large scale global optimization, in: Proc. 2014 IEEE International Conference on Systems Man and Cybernetics (SMC), 2014.

[32] F. Wei, Y. Wang, T. Zong, Variable grouping based differential evolution using an auxiliary function for large scale global

optimization, in: Proc. 2014 IEEE Congress on Evolutionary Computation (CEC), 2014.

[33] Z. Yang, K. Tang, X. Yao, An orthogonal genetic algorithm for multimedia multicast routing, Proceedings of IEEE Transactions on Evolutionary Computation Evolutionary Computation 3(1)(1999) 53-62.

[34] Z. Yang, K. Tang, X. Yao, Large scale evolutionary optimization using cooperative coevolution, Information Sciences: an International Journal 178(15)(2008) 2985-2999.

[35] Z. Yang, K. Tang, X. Yao, Multilevel cooperative coevolution for large scale optimization, in: Proc. IEEE Congress on Evolutionary Computation, 2008.

[36] Z. Yang, K. Tang, X. Yao, Self-adaptive differential evolution with neighborhood search, in: Proc. CEC 2008 on Evolutionary Computation (IEEE World Congress on Computational Intelligence), .2008.

[37] Q. Zhang, Hybrid estimation of distribution algorithm for global optimization, Engineering Computations 21(1)(2004) 91-107.