Research Article

Biogeography-Based Optimization with Orthogonal Crossover

Quanxi Feng, 1,2 Sanyang Liu, 1 Guoqiang Tang, 2 Longquan Yong, 1 and Jianke Zhang 3

1 Department of Applied Mathematics, Xidian University, Xi’an 710071, China
2 School of Science, Guilin University of Technology, Guilin 541004, China
3 School of Science, Xi’an University of Posts and Telecommunications, Xi’an 710121, China

Correspondence should be addressed to Quanxi Feng; fqx9904@gmail.com and Sanyang Liu; liusanyang@126.com

Received 6 January 2013; Revised 15 April 2013; Accepted 15 April 2013

Academic Editor: Alexander P. Seyranian

Copyright © 2013 Quanxi Feng et al. This is an open access article distributed under the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

Biogeography-based optimization (BBO) is a new biogeography inspired, population-based algorithm, which mainly uses migration operator to share information among solutions. Similar to crossover operator in genetic algorithm, migration operator is a probabilistic operator and only generates the vertex of a hyperrectangle defined by the emigration and immigration vectors. Therefore, the exploration ability of BBO may be limited. Orthogonal crossover operator with quantization technique (QOX) is based on orthogonal design and can generate representative solution in solution space. In this paper, a BBO variant is presented through embedding the QOX operator in BBO algorithm. Additionally, a modified migration equation is used to improve the population diversity. Several experiments are conducted on 23 benchmark functions. Experimental results show that the proposed algorithm is capable of locating the optimal or closed-to-optimal solution. Comparisons with other variants of BBO algorithms and state-of-the-art orthogonal-based evolutionary algorithms demonstrate that our proposed algorithm possesses faster global convergence rate, high-precision solution, and stronger robustness. Finally, the analysis result of the performance of QOX indicates that QOX plays a key role in the proposed algorithm.

1. Introduction

Many problems in both industrial application field and scientific research world can be regarded as optimization problems. During the past decades, several kinds of classical methods [1, 2] have been proposed to handle optimization problem, which have made enormous progress, but these methods demand to know the property of optimization problem, such as continuity or differentiability. In recent decades, many meta heuristic algorithms sprung up, for example, genetic algorithm (GA) [3], particle swarm optimization (PSO) [4], simulated annealing algorithm (SA) [5], differential evolution (DE) [6], and biogeography-based optimization (BBO) [7]. These algorithms can solve optimization problems without using some information such as differentiability.

Biogeography-based optimization (BBO) [7], developed by Simon, is a new emerging population-based evolutionary algorithm motivated through mimic migration of species in natural biogeography. Just as species migrate back and forth between islands in biogeography, migration operator in BBO shares information between habitats in the population. That is to say, a good solution can share their features with poor ones through migration operator, and a poor solution can improve their quality by accepting new features from good ones. Thus, BBO possesses powerful exploitation ability. However, migration operator does not produce new SIV which may lead to weak exploration ability and poor population diversity.

In evolution algorithm, the step of generating some solutions can be considered as “experiment.” For example, the operation of using crossover operator to sample genes from their parents and produce new offspring, this process can be considered as sample experiment. Orthogonal experiment design is a method that is used to generate multifactor experiment uniformly, which has been developed to sample a small, but representative set of combinations for experimentation. Zhang and Leung [8] introduced orthogonal experimental design into genetic algorithm and proposed orthogonal crossover (OX). Leung and Wang [9] used quantization technique in orthogonal experimental design and proposed quantization orthogonal crossover operator
(QOX). Experimental studies show that QOX is an effective and efficient operator for numerical optimization.

Since BBO has properties of powerful exploitation ability, weak exploration ability, poor population diversity, and slow convergence rate simultaneous. QOX operator is a global search operator with systematic and rational search ability. In order to enhance the exploration ability and improve the population diversity, this paper, an improved BBO variant, namely, biogeography-based optimization with orthogonal crossover (denoted as OXBBBO), is presented for solving global optimization problems. In the proposed algorithm, QOX operator is embedded into BBO to enhance its exploration ability and accelerate its convergence rate, and a modified migration operator is used to improve the population diversity.

The rest of the paper is organized as follows. The basic BBO is introduced in Section 2, while Section 3 briefly reviews orthogonal experiment design. The proposed algorithm is introduced in Section 4 in detail. In Section 5, extensive experiments have been carried out to test OXBBBO. Finally, some conclusions and future works are summarized in Section 6.

2. Biogeography-Based Optimization

BBO is a new population-based biogeography inspired global optimization algorithm, which has some features in common with other biology-based algorithms. Migration operator is the main operator of BBO which shares information among solutions. PSO and BBO solutions survive forever, while GA solutions “die” at the end of each generation. PSO solutions are more likely to clump together in similar groups, while BBO and GA solutions do not have any built-in trend to cluster.

BBO is a method motivated by geographical distribution of biological organisms. In BBO, each individual is considered as a “habitat” with a habitat suitability index (HSI). Habitats with a high HSI tend to have a large number of species, while those with a low HSI have a small number of species. Habitats with a high HIS migrate to the nearby habitats, while those with a low HSI accept features from neighbor habitats with high HISI.

Migration strategy in BBO is similar to the global recombination approach in the breeder GA and evolutionary strategy. Migration operator is a probabilistic operator that adjusts habitat based on the immigration rate and emigration rate. The probability that \( H \) is modified is proportional to its immigration rate \( \lambda \), and the probability that the modification source comes from \( H \) is proportional to the emigration rate \( \mu \). Migration operator [7] can be described in Algorithm 1.

A habitat’s HSI can be changed suddenly because of random events which are modeled as mutation operator in BBO. Mutation operator randomly modifies SIV based on the habitat’s priori probability. The mutation rate \( m \) is expressed as follows:

\[
m = m_{\text{max}} \left( \frac{1 - p_k}{p_{\text{max}}} \right),
\]

where \( m_{\text{max}} \) is a user-defined parameter. In BBO, highly probable solutions will tend to be more dominant in the population without this modification. This mutation scheme tends to increase diversity among population. The mutation operator [7] can be loosely described in Algorithm 2.

As some scholars has pointed out that BBO may have some deficiencies. For example, BBO is good at exploiting the solution space, but BBO is weak at exploring the solution space. Recently, many researchers have been working on the improvement of BBO, and many variants of BBO are presented.

Improving the operator in BBO is one direction for improvement. Gong et al. [11] used a real code to represent the solution and extend BBO to the continuous domain optimization and then presented a real-coded BBO (RCBBO). Ma and Simon [16] modified the migration operator by giving a new migration formula and then proposed blend BBO (B-BBO). Li et al. [10] presented perturbing biogeography-based optimization (PBBO) through a design of perturbing migration based on the sinusoidal migration curve. Li and Yin [12] proposed another migration operation based on multiparent crossover called multiparent migration model and then presented multioperator biogeography-based optimization (MOBBO). Ma [17] generalized linear migration model to six different migration models and then used experimental study and theoretical analysis to investigate their performance.

Moreover, theory analysis of BBO and application has also been developing. Simon [23] introduced several simplified versions of BBO and used probability theory to perform an approximate analysis. Simon et al. [24] showed that BBO is a generalization of a genetic algorithm with global uniform recombination (GA/GUR) and compared BBO and GA/GUR algorithms by using analytical Markov models. Simon [25] also introduced dynamic system models for BBO and GA/GUR.

Ma et al. [26] incorporated resampling in BBO to solve optimization problems in noisy environments. Ergezer et al. [27] employed opposition-based learning alongside migration rate of BBO and created oppositional BBO (OBBO).

It is necessary to emphasize that our work is to add a new global search operator into BBO algorithm. Meanwhile, we also aim at not increasing too many control parameters in the algorithm.

3. Orthogonal Crossover

In practical application, it is impossible to consider all combinations in large sample space, and then we can use a small part of representative samples to represent the sample space. Orthogonal design method [9], with both orthogonal array (OA) and factor analysis (FA), is used to execute multifactor experiment uniformly and sample well-distributed in solution space. OA is a fractional factorial array of numbers arranged in rows and columns which can assure a balanced comparison of levels of any factor. All columns in orthogonal array can be evaluated independently of one another, a number of such arrays can be found in http://www2.research.att.com/~njas/oadri/

For example, an experiment with 4 factors and 3 levels, its orthogonal array is \( L_9 (3^4) \). There are 81 combinations for all. If we apply the orthogonal array, we only consider nine combinations.

For convenience, we denote an orthogonal array for \( F \) factors with \( Q \) levels and \( M \) combinations as \( L_M (Q^F) = \{ a_{ij} \}_{M \times N} \), where \( a_{ij} \in \{1, 2, \ldots, Q \} \) represents the \( j \)th factor in the \( i \)th combination. Generally,

\[
M \leq \frac{Q^F - 1}{Q - 1}.
\]  

Orthogonal crossover operator was first developed by Zhang and Leung [8], which originated from the integration of orthogonal experiment design with crossover operator to generate several new solutions in the line with orthogonal array. As a matter of fact, each operation of generating a new offspring can be regarded as an experiment. For example, a crossover is a procedure for sampling several points from a defined region while orthogonal crossover operator (OX) uses orthogonal array to make crossover operator with more statistically. OX in [8] was used in combination problems. Leung and Wang [9] used OX with quantization technique and proposed new OX used for dealing with numerical optimization with continuous variables. Leung quantized the continuous domain into several levels firstly and then used orthogonal array to generate combinations.

Consider two parent habitats \( x = (x_1, x_2, \ldots, x_N) \) and \( y = (y_1, y_2, \ldots, y_N) \). \( x \) and \( y \) define a search space \([\min(x_i, y_i), \max(x_i, y_i)]\) of variable \( x_i \). Then, the solution space is \([l_{\text{low}}, u_{\text{up}}]\), where

\[
l_{\text{low}} = \min(x_1, y_1), \min(x_2, y_2), \ldots, \min(x_N, y_N),
\]

\[
u_{\text{up}} = \max(x_1, y_1), \max(x_2, y_2), \ldots, \max(x_N, y_N).
\]

QOX quantizes the solution space \([l_{\text{low}}, u_{\text{up}}]\) into \( Q \) levels:

\[
\beta_{ij} = \begin{cases} 
\min(x_i, y_i), & j = 1, \\
\min(x_i, y_i) + (j - 1) \left( \frac{|x_i - x_j|}{Q - 1} \right), & 2 \leq j \leq Q - 1, \\
\max(x_N, y_N), & j = Q.
\end{cases}
\]

In practical application, dimension \( N \) is often much larger than orthogonal factor \( F \), and \( L_M (Q^F) \) cannot be used directly. So, we first randomly generate \( F - 1 \) different integers between 1 and \( N \) and then divide \( N \)-dimension solution vector into \( F \) factors:

\[
f_1 = (x_1, \ldots, x_k),
\]

\[
f_2 = (x_{k+1}, \ldots, x_{k+l}),
\]

\[
\vdots
\]

\[
f_F - 1 = (x_{k_F - 1 + 1}, \ldots, x_{k_F}),
\]

\[
f_F = (x_{k_F}, \ldots, x_N).
\]

Finally, \( M \) combinations are generated according to orthogonal array \( L_M (Q^F) \) whose habitat suitability index is evaluated, and the two best habitats are selected from them to replace their parent habitats.

QOX has been incorporated into particle swarm optimization [4], differential evolution [28], and genetic algorithm [9] to enhance the exploration ability.
4. The Proposed Algorithm: OXBBO

4.1. Orthogonal Crossover Operator. Migration operator plays a key role in BBO. Simon et al. [24] has pointed out that BBO migration strategy is conceptually similar to the GA global uniform recombination operator, which can be considered as a special GA crossover operator [24]. Similar to GA crossover operator, migration operator only generates and evaluates the habitat which is located in the vertex of hyperrectangle defined by the emigration habitats and immigration habitats. The inner and other vertexes of the hyperrectangle are not considered in this process, which might be a promising region in the solution space. This process may lead the optimal solution to fall into local optimal solution, especially, in the initial optimizing stage. That is to say, migration operator cannot perform a systematic search in the hyperrectangle. Therefore, the exploration ability of BBO could be limited to some extent.

Orthogonal crossover operator with quantization technique (QOX) samples a small but representative set of combinations for experimentation. In the searching process, QOX not only searches the vertexes of hyperrectangle defined by parents but also searches the inner of the hyperrectangle. So QOX could execute systematic search in the hyperrectangle, which can locate the global optimal solution easy and overcome the limitation of migration operator.

As shown in Figure 1, $P_1(x_1, y_1)$ and $P_2(x_2, y_2)$ are the parents (supposing that one is emigration habitat and the other one is immigration habitat in two-dimensional solution space). We can see in Figure 1 that GA crossover operator generates two offspring randomly. Migration operator absorbs better SIVs and generates one offspring, determined by immigration and emigration rate, in the vertex consciously. And orthogonal crossover operator with $L_9(3^4)$ generates nine potential offspring which lie in both the vertexes and the inner of hyperrectangle uniformly. Obviously, orthogonal crossover operator searches the solution space more thoroughly than GA crossover operator and migration operator do.

In order to enhance the exploration ability of BBO, we embedded the QOX operator with $L_9(3^4)$ into BBO. For the sake of randomness, two random selected habitats are used as the parent of QOX, which will be replaced by two best habitats chosen from offspring after performing QOX. Note that QOX operator is more costly than migration operator and GA operator, since it needs to evaluate 9 offspring if $L_9(3^4)$ is used. Considering the limitation of the number of fitness function evaluations, we apply QOX with $L_9(3^4)$ only once at each generation to save the computation cost and to simplify the implementation procedure. The brief description of QOX is shown in Algorithm 3.

**Algorithm 3: Procedure of QOX.**

1. **Begin**
2. **Input:** parameters $Q$ and $F$
3. Select $M$ fulfilling formula (2)
4. Randomly select two habitats from the population
5. Quantize the domain defined by the two habitats into $Q$ levels
6. Randomly generate $F-1$ different integers between 1 and dimension $N$
7. Divide $N$-dimension habitat vector into $F$ factors
8. Apply $L_M(Q^F)$ to generate $M$ potential offspring
9. Evaluate $M$ potential offspring
10. Select two best habitats to replace their parents
11. **Output:** two best habitats
12. **End**

4.2. Modified Migration Operator. As analyzed before, in migration operator, the SIV of new habitat migrates from other habitats. This procedure only uses the SIV of better
(1) Begin
(2) Generate initial population Pop randomly
(3) Evaluate each habitat $X_i$
(4) While the halting criteria are not satisfied do
(5) Sort habitats from best to worst based on the fitness values
(6) Map the fitness value to the number of spices
(7) Calculate the emigration rate and immigration rate of each habitat
(8) For $i = 1$ to NP /* migration stage */
(9) Select $X_i$ based on immigration rate $\lambda_i$
(10) Generate two different integers $r1 \neq r2$ randomly between 1 and NP
(11) If $X_i$ is selected
(12) For $j = 1$ to NP
(13) Select $X_j$ based on immigration rate $\mu_j$
(14) If rand(0, 1) < $\mu_j$
(15) $H_i (\text{SIV}) = H_j (\text{SIV}) + \alpha (H_{r1} (\text{SIV}) - H_{r2} (\text{SIV}))$
(16) End If
(17) EndFor
(18) EndIf
(19) EndFor /* end of migration stage */
(20) For $i = 1$ to NP /* mutation stage */
(21) Compute the probability $p_i$
(22) Select $X_i$ based on $p_i$
(23) If rand(0, 1) < $m_i$
(24) $H_i (\text{SIV}) = H_j (\text{SIV}) + \delta(1)$
(25) End If
(26) EndFor /* end of mutation stage */
(27) Make sure each habitat legal based on boundary constraints
(28) Evaluate each newly generated habitat $M_i$
(29) For $i = 1$ to NP /* selection stage */
(30) If $f(X_i) < f(M_i)$
(31) Reserve $X_i$ in the next population
(32) Else
(33) Reserve $M_i$ in the next population
(34) EndIf
(35) EndFor /* end of selection stage */
(36) Input: parameters $Q$ and $F$ /* QOX search stage */
(37) Select $M$ fulfilling formula (3)
(38) Randomly select two habitats $r3 \neq r4$ from the population
(39) Quantize the domain defined by the two habitats into $Q$ levels
(40) Randomly generate $F-1$ different integers between 1 and dimension $N$
(41) Divide $N$-dimension habitat vector into $F$ factors
(42) Apply $L_{Q2}(Q^F)$ to generate $M$ potential offspring
(43) Evaluate $M$ potential offspring
(44) Select two best habitats to replace habitat $r3,r4$
(45) Output: two best habitats /* end of QOX search stage */
(46) EndWhile
(47) End

Algorithm 4: Main Procedure of OXBBO.

habitats to replace the SIV of the worse habitats but does not produce new SIV which results in poor population diversity, especially in the later stage. For the sake of this, a new modified migration is used to improve the population diversity. In Algorithm 1, formula (**) is replaced by a new formula:

$$H_i (\text{SIV}) = H_j (\text{SIV}) + \alpha (H_{r1} (\text{SIV}) - H_{r2} (\text{SIV})),$$  \hspace{1cm} (6)

where $r1, r2$ are two different integers generated randomly between 1 and NP (NP is population size). We can see in formula (6), the difference generated by two randomly selected habitats are added to the emigration SIV of habitat, which can generate a random perturbation on the emigration SIV and then improve the diversity of new population.

4.3. Mutation with Mutation Operator. In order to enhance the exploration ability of BBO, a modified mutation operator, Cauchy mutation operator, is integrated into BBO to replace the randomly generated SIV.
The formula for the probability density function of the Cauchy distribution \([11]\) is

\[
f_t(x) = \frac{1}{\pi \left(t^2 + x^2\right)}.
\]

where \(x \in \mathbb{R}\), and \(t > 0\) is a scale parameter. A real-valued random variable \(Y\) is Cauchy distributed with \(t > 0\) being written as

\[
Y \sim \delta(t).
\]

The Cauchy mutation \((t = 1)\) in OXBBO can be described as

\[
X_i(\text{SIV}) = X_i(\text{SIV}) + \delta(1).
\]

4.4. Boundary Constraints. The OXBBO algorithm assumes that all the habitats in the population should be limited in an isolated and finite solution space. There are some habitats moving out of the solution space, in the optimizing process, which should be prevented. In order to keep the solution of bound-constrained problems feasible, those habitats that violate boundary constraints should be replaced by a new generate habitat. The formula is

\[
X_{ij}(\text{SIV}) = l_j + \text{rand} \times (u_j - l_j),
\]

where \(l_j, u_j\) are the lower and upper bound, of the solution space, respectively.

4.5. Selection Operator. Selection operator can impel the metabolism of population and retain good habitats. Greedy selection operator is used in this paper. In other words, the population habitat will be replaced by its corresponding trial habitat if the HSI of the trial habitat is better than that of its population habitat:

\[
X_i = \begin{cases} 
X_i & \text{if } f(X_i) < f(M_i) \\
M_i & \text{else},
\end{cases}
\]

where \(X_i, M_i\) are the population habitat and trial habitat, respectively.

4.6. Main Procedure of OXBBO. As the analysis above, migration operator is good at exploiting the solution space and weak at exploring the solution space. QOX is good at exploring the solution space, which can offset the deficiency of BBO. Modified migration operator can increase the difference of population and improve the population diversity. Therefore, we introduce biogeography-based optimization with orthogonal crossover (viz., OXBBO) by incorporating the above-mentioned QOX and modified migration operator into BBO in this section. The pseudocode of OXBBO approach is described in Algorithm 4.

5. Experimental Study

In order to verify the performance of the proposed algorithm, several experiments are implemented on 23 benchmark functions. These benchmark functions are chosen from \([29]\) and briefly summarized in Table 1, detailed descriptions about these functions can be found in \([29]\).

Functions \(f_1-f_5\) are unimodal functions. Where function \(f_5\) has a very sharp narrow ridge running around a parabola. Function \(f_6\) is a discontinuous step function with one minimum. Function \(f_7\) is a low-dimensional function with a noisy perturbation. Functions \(f_8-f_{13}\) are multimodal functions where the number of local minima increases exponentially with dimension. Functions \(f_{14}-f_{22}\) are low-dimensional functions with only a few local minima.

5.1. Experimental Setting. For OXBBO, we have chosen a reasonable set of values. For all experimental tests, we use the following parameters setting unless a change is mentioned:

1. population size: \(NP = 100\);
2. maximum immigration rate: \(I = 1\);
3. maximum emigration rate: \(E = 1\);
4. mutation probability: \(m_{\text{max}} = 0.005\);
5. value to reach \((\text{VTR}) = 10^{-8}\), except for \(f_7\) of VTR = \(10^{-2}\);
6. maximum number of fitness function evaluations \((\text{Max.NFES}) = 150,000 \text{ for } f_1, f_6, f_{10}, f_{12}, \text{ and } f_{13}; 200,000 \text{ for } f_2 \text{ and } f_{11}; 300,000 \text{ for } f_7, f_8, \text{ and } f_9; 500,000 \text{ for } f_3, f_4, \text{ and } f_{15}; 10,000,000 \text{ for } f_14, f_{16}-f_{19}, f_{21}, \text{ and } f_{22}; 40,000 \text{ for } f_5; 20,000 \text{ for } f_{20}\).

All the algorithms are evaluated on performance criteria \([14, 15]\) or similar to those. (1) Successful rate (SR) is The ratio of the number of successful runs to total runs. (2) NEFFs the number of fitness function evaluations is needed when reach VTR. (3) Acceleration rate (AR): AR = \(\text{NEFFs}_{\text{other}}/\text{NEFFs}_{\text{OXBBO}}\). AR is used to compare the convergence speed between OXBBO and other algorithms.

Note. For convenience's sake of comparison, the best function values and their standard variation are compared in Section 5.2, and the best function error values and their corresponding standard variation are compared in other sections. Moreover, high-dimensional functions \(f_1-f_{13}\) with dimension 10, 50, 100, and 200 are optimized in Section 5.5. Functions \(f_1-f_{13}\) with 30 dimensions are optimized in other sections.

All experimental tests in this paper are implemented on a computer with 1.86 GHz Inter-core Processor, 1GB of RAM, and Windows XP operating system in Matlab software 7.6.

5.2. Comparison with Improved BBO Algorithms. In order to validate the performance of OXBBO, we first compare OXBBO with BBO \([11]\) and improved BBO algorithms: Perturb BBO with Gaussian mutation (PBB-G) \([10]\), real code BBO with Gaussian mutation (RCBBO-G) \([11]\), and Multioperator BBO (MOBBO) \([12]\). OXBBO executed 50 independent runs on 23 benchmark functions; mean fitness values and their standard variations are recorded. Results of other algorithms are taken from corresponding papers directly. Comparison results are summarized in Table 2.
Table 1: Benchmark functions used in our experimental tests.

<table>
<thead>
<tr>
<th>Function</th>
<th>Name</th>
<th>Dim</th>
<th>Search space</th>
<th>Optima</th>
</tr>
</thead>
<tbody>
<tr>
<td>$f_1(x) = \sum_{i=1}^{n} x_i^2$</td>
<td>Sphere</td>
<td>$n$</td>
<td>$[-100, 100]$</td>
<td>0</td>
</tr>
<tr>
<td>$f_2(x) = \sum_{i=1}^{n}</td>
<td>x_i</td>
<td>+ \prod_{i=1}^{n}</td>
<td>x_i</td>
<td>$</td>
</tr>
<tr>
<td>$f_3(x) = \sum_{i=1}^{n} \left( \sum_{j=1}^{i} x_j \right)^2$</td>
<td>Schwefel's problem 1.2</td>
<td>$n$</td>
<td>$[-100, 100]$</td>
<td>0</td>
</tr>
<tr>
<td>$f_4(x) = \max\left{</td>
<td>x_i</td>
<td></td>
<td>, 1 \leq i \leq n\right}$</td>
<td>Schwefel's problem 2.21</td>
</tr>
<tr>
<td>$f_5(x) = \sum_{i=1}^{n-1} [100(x_{i+1} - x_i)^2 + (x_i - 1)^2]$</td>
<td>Rosenbrock function</td>
<td>$n$</td>
<td>$[-30, 30]$</td>
<td>0</td>
</tr>
<tr>
<td>$f_6(x) = \sum_{i=1}^{n} (\max\left{</td>
<td>x_i</td>
<td>\right} + 0.5)^3$</td>
<td>Step function</td>
<td>$n$</td>
</tr>
<tr>
<td>$f_7(x) = \sum_{i=1}^{n} 4x_i^4 + \text{random}(0, 1)$</td>
<td>Quartic function</td>
<td>$n$</td>
<td>$[-1.28, 1.28]$</td>
<td>0</td>
</tr>
<tr>
<td>$f_8(x) = -\sum_{i=1}^{n} x_i \sin(\sqrt{</td>
<td>x_i</td>
<td>})$</td>
<td>Schwefel function</td>
<td>$n$</td>
</tr>
<tr>
<td>$f_9(x) = \sum_{i=1}^{n} x_i^2 - 10 \cos(2\pi x_i) + 10$</td>
<td>Rastrigin function</td>
<td>$n$</td>
<td>$[-5.12, 5.12]$</td>
<td>0</td>
</tr>
<tr>
<td>$f_{10}(x) = -20 \exp\left{ -0.2\sqrt{(1/n) \sum_{i=1}^{n} x_i^2} - \exp\left( (1/n) \sum_{i=1}^{n} \cos(2\pi x_i) \right) + 20 + e \right}$</td>
<td>Ackley function</td>
<td>$n$</td>
<td>$[-32, 32]$</td>
<td>0</td>
</tr>
<tr>
<td>$f_{11}(x) = (1/4000) \sum_{i=1}^{n} x_i^4 - \prod_{i=1}^{n} \cos(x_i/\sqrt{i}) + 1$</td>
<td>Griewank function</td>
<td>$n$</td>
<td>$[-600, 600]$</td>
<td>0</td>
</tr>
<tr>
<td>$f_{12}(x) = (\pi/n) \left[ 10 \sin^2(\pi y_1) + \sum_{i=1}^{n-1} (y_{i+1} - 1)^2 \left( 1 + 10 \sin^2(\pi y_{i+1}) \right) \right]$</td>
<td>Penalized function 1</td>
<td>$n$</td>
<td>$[-50, 50]$</td>
<td>0</td>
</tr>
<tr>
<td>$y_i = 1 + (x_i - 1)/4$</td>
<td>Penalized function 2</td>
<td>$n$</td>
<td>$[-50, 50]$</td>
<td>0</td>
</tr>
<tr>
<td>$u(x, 0) = \left{ \begin{array}{ll} a_k x_k^2, &amp; x_k &gt; a \ 0, &amp; -a \leq x_k \leq a \ l_k(x_k - a_k)^2, &amp; x_k &lt; -a \end{array} \right.$</td>
<td>Shekel function 1</td>
<td>$n$</td>
<td>$[-65.536, 65.536]$</td>
<td>1</td>
</tr>
<tr>
<td>$f_{13}(x) = 0.1 \left[ \sin(3\pi x_1) + \sum_{i=1}^{n-1} (x_{i+1} - 1)^2\left( 1 + \sin^2(3\pi x_{i+1}) \right) \right]$</td>
<td>Kowalik’s function</td>
<td>$n$</td>
<td>$[-5, 5]$</td>
<td>0.003075</td>
</tr>
<tr>
<td>$f_{14}(x) = \left[ 1/500 + \sum_{i=1}^{n} 1/\left( j + \sum_{j=1}^{n} (x_i - a_j)^6 \right)^{-1} \right]^{-1}$</td>
<td>Six-Hump camel-Back function</td>
<td>$n$</td>
<td>$[-5, 5]$</td>
<td>$-1.0316$</td>
</tr>
<tr>
<td>$f_{15}(x) = (x_1 - 0.5/4 \pi^2)^2 + (5/\pi) x_1 - 6)^2 + 10(1 - 1/8\pi) \cos(x_1) + 10$</td>
<td>Branin function</td>
<td>$n$</td>
<td>$[-5, 10] \times [0, 15]$</td>
<td>0.398</td>
</tr>
<tr>
<td>$f_{16}(x) = [1 + (x_1 + x_2 + 1)^2(19 - 14x_1 + 3x_2^2 - 14x_2 + 6x_1x_2 + 3x_2^2)] \times [30 + (2x_1 - 3x_2)^2(18 - 32x_1 + 12x_2^2 + 48x_2 - 36x_1x_2 + 27x_2^2)]$</td>
<td>Goldstein-Price function</td>
<td>$n$</td>
<td>$[-2, 2]$</td>
<td>3</td>
</tr>
<tr>
<td>$f_{17}(x) = -\sum_{i=1}^{n} \xi_i \exp\left( -\sum_{i=1}^{n} a_i(x_i - p_i)^2 \right)$</td>
<td>Hartman’s function 1</td>
<td>$n$</td>
<td>$[0, 1]$</td>
<td>$-3.86$</td>
</tr>
<tr>
<td>$f_{18}(x) = -\sum_{i=1}^{n} \xi_i \exp\left( -\sum_{i=1}^{n} a_i(x_i - p_i)^2 \right)$</td>
<td>Hartman’s function 2</td>
<td>$n$</td>
<td>$[0, 1]$</td>
<td>$-3.32$</td>
</tr>
<tr>
<td>$f_{19}(x) = -\sum_{i=1}^{n} \xi_i \exp\left( -\sum_{i=1}^{n} a_i(x_i - p_i)^2 \right)$</td>
<td>Shekel's function 1</td>
<td>$n$</td>
<td>$[0, 10]$</td>
<td>$-10.1532$</td>
</tr>
<tr>
<td>$f_{20}(x) = -\sum_{i=1}^{n} \xi_i \exp\left( -\sum_{i=1}^{n} a_i(x_i - p_i)^2 \right)$</td>
<td>Shekel's function 2</td>
<td>$n$</td>
<td>$[0, 10]$</td>
<td>$-10.4029$</td>
</tr>
<tr>
<td>$f_{21}(x) = -\sum_{i=1}^{n} \xi_i \exp\left( -\sum_{i=1}^{n} a_i(x_i - p_i)^2 \right)$</td>
<td>Shekel's function 3</td>
<td>$n$</td>
<td>$[0, 10]$</td>
<td>$-10.5364$</td>
</tr>
</tbody>
</table>

Note: Dim $n$ means these functions with changeable dimension.
We make a comparison of OXBBO with original BBO firstly. In Table 2, the results of OXBBO are better than those of BBO in 22 out of 23 benchmark functions except \(f_9\). Either higher-dimensional functions or lower-dimensional functions, OXBBO outperforms BBO. This means that the integration of QOX and modified operator in BBO can improve the performance greatly.

Compared with PBBO-G, from Table 2, we can see that OXBBO is better than PBBO-G in 21 out of 23 functions. For unimodal function \(f_1\)–\(f_6\), it is obvious that OXBBO performs better than or similar to PBBO-G in 5 out of 6 functions, and PBBO-G is better than OXBBO only on function \(f_5\). For higher-dimensional multimodal functions, OXBBO performs better than PBBO-G for all the multimodal functions. For lower dimension functions, OXBBO outperforms PBBO-G except function \(f_{17}\), and both algorithms obtain similar results on function \(f_{17}\). Moreover, the results of OXBBO are better than those of PBBO-G by several orders of magnitude on functions \(f_1\), \(f_2\), \(f_3\), \(f_5\), \(f_{10}\), \(f_{11}\), \(f_{12}\), \(f_{13}\), \(f_{14}\), \(f_{16}\), \(f_{18}\), and \(f_{23}\). The analysis shows that the combination of QOX and modified migration operator into BBO is much better than combination of perturbing migration and sinusoidal migration curve in BBO.

What is more is that similar results are obtained when we compare OXBBO with RCBBO-G. OXBBO outperforms RCBBO-G in 22 test functions, and OXBBO, RCBBO-G obtain the same results on test function \(f_6\). When compared with MOBBO, OXBBO is better than or similar to MOBBO in 12 functions out of 23 functions. According to the parameters setting, MOBBO and OXBBO are tested with population size \(65 \times 3 = 195\) and 100; both of the two algorithms are tested with the same maximum generations. That is to say, the maximum number of fitness function evaluations of MOBBO is almost twice as that of OXBBO. Meanwhile, MOBBO is better than OXBBO in precision in some functions that excellent results are also obtained by OXBBO. The above analysis indicates that OXBBO not only has better exploitation ability but also has better exploration ability.

### 5.3. Comparison with Hybrid BBO Algorithms

In order to verify the performance of OXBBO further, comparisons of OXBBO with hybrid BBO algorithms, such as BBO-EP [11], DBBO [13], eBBO [14], and DE/BBO [15], are made. The algorithms we adopt in this section to conduct the comparison are to directly cite their numerical results available in the according papers. OXBBO are executed 50 independent runs on 13 high-dimensional functions. The numerical results of 13 high-dimensional benchmark functions are listed in Table 3.

From Table 3, OXBBO outperforms BBO-EP, DBBO, eBBO, and DE/BBO in 9, 9, 8, and 6 out of 13 test functions, respectively. In addition, BBO-EP, DBBO, eBBO, and DE/BBO surpass OXBBO in 3, 3, 3, and 3 out of 13 test

---

**Note:** according to the setting, these algorithms use the same maximum generations for each function, but population size of MOBBO is almost twice as that of other algorithms is 100, so the MAX.NFFEis of MOBBO are twice those of the others.
functions, respectively. Thus, we can conclude that, overall, OXBBO is better than BBO-EP, DBBO, and eBBO and is very competitive with DE/BBO. The above discussion means that OXBBO is an effective global function optimizer.

5.4. Convergence Speed When Compared with BBO Algorithms. From Section 5.3, we can see that, though OXBBO outperforms four algorithms, the results of eBBO, DE/BBO, OXBBO, and vice versa.

Table 4 summarizes the mean NFFEs and SR of eBBO, DE/BBO, and OXBBO for solving 13 test functions, and ARs between algorithm OXBBO and eBBO, OXBBO, and DE/BBO.

OXBBO requires fewer NFFEs to reach the VTR than DE/BBO does on 10 functions out of 11 functions with a successful run, which can indicate that our algorithm is faster than DE/BBO. Similar results can be obtained when compared with eBBO. For example, tests on function $f_1$ show that the average numbers of NFFEs of 56110, 59926, and 44398 are needed by the eBBO, DE/BBO, and OXBBO, respectively, to reach the VTR. However, OXBBO only needs 44398 NFFEs, which means that its CPU time is the shortest among the three algorithms. Moreover, it can be seen from Table 4 that the average AR between algorithms OXBBO and eBBO, OXBBO, and DE/BBO in the successful run are 1.26 and 1.37 (the total AR divided by 11); that is to say, the total

Table 3: Comparison of hybrid BBO algorithms and OXBBO for high-dimension functions.

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Mean (Std)</td>
<td>Mean (Std)</td>
<td>Mean (Std)</td>
<td>Mean (Std)</td>
<td>Mean (Std)</td>
</tr>
<tr>
<td>$f_1$</td>
<td>7.76E - 36 (3.80E - 35)</td>
<td>6.22E - 27 (2.18E - 26)</td>
<td>8.66E - 28 (5.21E - 28)</td>
<td>3.96E - 28 (2.47E - 28)</td>
</tr>
<tr>
<td>$f_2$</td>
<td>6.14E - 04 (2.64E - 03)</td>
<td>4.46E - 23 (1.28E - 22)</td>
<td>0.00E + 00 (0.00E + 00)</td>
<td>1.12E - 19 (4.01E - 20)</td>
</tr>
<tr>
<td>$f_3$</td>
<td>1.52E - 11 (1.86E - 11)</td>
<td>4.98E + 01 (8.04E + 01)</td>
<td>2.26E - 03 (1.58E - 03)</td>
<td>3.21E - 03 (1.96E - 03)</td>
</tr>
<tr>
<td>$f_4$</td>
<td>7.68E - 08 (1.37E - 07)</td>
<td>1.89E - 15 (8.85E - 16)</td>
<td>2.61E - 15 (8.83E - 15)</td>
<td>1.89E - 15 (8.83E - 15)</td>
</tr>
<tr>
<td>$f_5$</td>
<td>5.16E + 01 (4.37E + 01)</td>
<td>1.90E + 01 (7.52E + 00)</td>
<td>1.89E + 01 (7.52E + 00)</td>
<td>1.89E + 01 (7.52E + 00)</td>
</tr>
<tr>
<td>$f_6$</td>
<td>0.00E + 00 (0.00E + 00)</td>
<td>0.00E + 00 (0.00E + 00)</td>
<td>0.00E + 00 (0.00E + 00)</td>
<td>0.00E + 00 (0.00E + 00)</td>
</tr>
<tr>
<td>$f_7$</td>
<td>1.54E - 03 (3.41E - 04)</td>
<td>1.88E - 03 (3.60E - 03)</td>
<td>3.44E - 03 (8.27E - 04)</td>
<td>2.01E - 03 (6.45E - 04)</td>
</tr>
<tr>
<td>$f_8$</td>
<td>8.13E - 09 (2.21E - 09)</td>
<td>0.00E + 00 (0.00E + 00)</td>
<td>0.00E + 00 (0.00E + 00)</td>
<td>0.00E + 00 (0.00E + 00)</td>
</tr>
<tr>
<td>$f_9$</td>
<td>8.05E - 03 (3.75E - 03)</td>
<td>1.45E - 14 (9.93E - 15)</td>
<td>1.07E - 14 (1.90E - 15)</td>
<td>6.79E - 15 (1.50E - 15)</td>
</tr>
<tr>
<td>$f_{10}$</td>
<td>6.57E - 03 (9.00E - 03)</td>
<td>6.79E - 15 (1.50E - 15)</td>
<td>6.79E - 15 (1.50E - 15)</td>
<td>6.79E - 15 (1.50E - 15)</td>
</tr>
<tr>
<td>$f_{11}$</td>
<td>9.64E - 06 (3.68E - 05)</td>
<td>7.35E - 32 (6.58E - 32)</td>
<td>7.16E - 29 (6.30E - 29)</td>
<td>7.06E - 30 (3.95E - 30)</td>
</tr>
<tr>
<td>$f_{12}$</td>
<td>4.51E - 04 (2.15E - 03)</td>
<td>1.08E - 26 (2.76E - 26)</td>
<td>9.81E - 27 (7.10E - 27)</td>
<td>7.45E - 29 (3.00E - 29)</td>
</tr>
</tbody>
</table>

Note: the result of $f_8$ obtained by eBBO is the fitness value but is fitness value error by the others.

Table 4: NFFEs required to obtain accuracy levels less than VTR.

<table>
<thead>
<tr>
<th>Mean (std)</th>
<th>SR</th>
<th>Mean (std)</th>
<th>SR</th>
<th>Mean (std)</th>
<th>SR</th>
<th>Mean (std)</th>
<th>SR</th>
</tr>
</thead>
<tbody>
<tr>
<td>eBBO</td>
<td>DE/BBO</td>
<td>OXBBO</td>
<td>3 versus 1</td>
<td>AR</td>
<td>AR</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$f_1$</td>
<td>56110 (3559.251)</td>
<td>100%</td>
<td>59926 (745.5)</td>
<td>100%</td>
<td>44398 (438.77)</td>
<td>100%</td>
<td>1.26</td>
</tr>
<tr>
<td>$f_2$</td>
<td>73670 (4640.076)</td>
<td>100%</td>
<td>82004 (983.9)</td>
<td>100%</td>
<td>84490 (1058.61)</td>
<td>100%</td>
<td>0.87</td>
</tr>
<tr>
<td>$f_3$</td>
<td>NA (NA)</td>
<td>0</td>
<td>NA (NA)</td>
<td>0</td>
<td>NA (NA)</td>
<td>0</td>
<td>NA</td>
</tr>
<tr>
<td>$f_4$</td>
<td>491498 (14486.144)</td>
<td>44%</td>
<td>296572 (4969.9)</td>
<td>100%</td>
<td>214889 (4626.80)</td>
<td>100%</td>
<td>2.29</td>
</tr>
<tr>
<td>$f_5$</td>
<td>NA (NA)</td>
<td>0</td>
<td>NA (NA)</td>
<td>0</td>
<td>NA (NA)</td>
<td>0</td>
<td>NA</td>
</tr>
<tr>
<td>$f_6$</td>
<td>17138 (2141.132)</td>
<td>100%</td>
<td>21590 (573.3)</td>
<td>100%</td>
<td>16055 (545.62)</td>
<td>100%</td>
<td>1.07</td>
</tr>
<tr>
<td>$f_7$</td>
<td>43864 (39947.48)</td>
<td>100%</td>
<td>109574 (2105.8)</td>
<td>100%</td>
<td>58173 (19577.82)</td>
<td>100%</td>
<td>0.75</td>
</tr>
<tr>
<td>$f_8$</td>
<td>59948 (2831.762)</td>
<td>100%</td>
<td>95952 (3126.7)</td>
<td>100%</td>
<td>63213 (1639.74)</td>
<td>100%</td>
<td>0.95</td>
</tr>
<tr>
<td>$f_9$</td>
<td>72220 (16458.023)</td>
<td>100%</td>
<td>170226 (8379.0)</td>
<td>100%</td>
<td>124639 (5195.17)</td>
<td>100%</td>
<td>0.58</td>
</tr>
<tr>
<td>$f_{10}$</td>
<td>84870 (4800.818)</td>
<td>100%</td>
<td>91308 (922.7)</td>
<td>100%</td>
<td>78839 (853.99)</td>
<td>100%</td>
<td>1.08</td>
</tr>
<tr>
<td>$f_{11}$</td>
<td>120422 (71312.35)</td>
<td>54%</td>
<td>62042 (1219.6)</td>
<td>100%</td>
<td>46312 (798.55)</td>
<td>100%</td>
<td>2.6</td>
</tr>
<tr>
<td>$f_{12}$</td>
<td>45416 (2502.314)</td>
<td>100%</td>
<td>54482 (873.3)</td>
<td>100%</td>
<td>40646 (641.83)</td>
<td>100%</td>
<td>1.19</td>
</tr>
<tr>
<td>$f_{13}$</td>
<td>54302 (2228.94)</td>
<td>100%</td>
<td>64772 (1134.3)</td>
<td>100%</td>
<td>43970 (605.58)</td>
<td>100%</td>
<td>1.23</td>
</tr>
</tbody>
</table>

Average 1.02E + 05 (1.50E + 04) 77% 1.01E + 05 (3.99E + 03) 85% 7.41E + 04 (3.27E + 03) 85% 1.26 | 1.37
Table 5: Comparison of DE/BBO and OXBBO with differential dimensions.

<table>
<thead>
<tr>
<th>Dim = 10</th>
<th>Dim = 50</th>
<th>Dim = 100</th>
<th>Dim = 200</th>
</tr>
</thead>
<tbody>
<tr>
<td>f_1</td>
<td>0.00E+00 (0.00E+00)</td>
<td>100%</td>
<td>0.00E+00 (0.00E+00)</td>
</tr>
<tr>
<td>f_2</td>
<td>0.00E+00 (0.00E+00)</td>
<td>100%</td>
<td>0.00E+00 (0.00E+00)</td>
</tr>
<tr>
<td>f_3</td>
<td>6.07E−14 (9.60E−14)</td>
<td>100%</td>
<td>3.12E−12 (3.65E−12)</td>
</tr>
<tr>
<td>f_4</td>
<td>1.58E−16 (9.88E−17)</td>
<td>100%</td>
<td>3.62E−13 (1.48E−13)</td>
</tr>
<tr>
<td>f_5</td>
<td>3.40E+00 (8.03E−01)</td>
<td>0</td>
<td>5.85E+00 (6.70E−01)</td>
</tr>
<tr>
<td>f_6</td>
<td>0.00E+00 (0.00E+00)</td>
<td>100%</td>
<td>0.00E+00 (0.00E+00)</td>
</tr>
<tr>
<td>f_7</td>
<td>1.11E−03 (3.96E−04)</td>
<td>100%</td>
<td>1.31E−03 (6.90E−04)</td>
</tr>
<tr>
<td>f_8</td>
<td>0.00E+00 (0.00E+00)</td>
<td>100%</td>
<td>0.00E+00 (0.00E+00)</td>
</tr>
<tr>
<td>f_9</td>
<td>5.89E−16 (0.00E+00)</td>
<td>100%</td>
<td>2.66E−15 (0.00E+00)</td>
</tr>
<tr>
<td>f_10</td>
<td>0.00E+00 (0.00E+00)</td>
<td>100%</td>
<td>0.00E+00 (0.00E+00)</td>
</tr>
<tr>
<td>f_11</td>
<td>0.00E+00 (0.00E+00)</td>
<td>100%</td>
<td>0.00E+00 (0.00E+00)</td>
</tr>
<tr>
<td>f_12</td>
<td>4.71E−32 (0.00E+00)</td>
<td>100%</td>
<td>4.71E−32 (1.66E−47)</td>
</tr>
<tr>
<td>f_13</td>
<td>1.35E−32 (0.00E+00)</td>
<td>100%</td>
<td>1.35E−32 (1.11E−47)</td>
</tr>
</tbody>
</table>

DE/BBO | OXBBO | DE/BBO | OXBBO | DE/BBO | OXBBO | DE/BBO | OXBBO | DE/BBO | OXBBO | DE/BBO | OXBBO |
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Mean (Std)</td>
<td>SR</td>
<td>Mean (Std)</td>
<td>SR</td>
<td>Mean (Std)</td>
<td>SR</td>
<td>Mean (Std)</td>
<td>SR</td>
<td>Mean (Std)</td>
<td>SR</td>
<td>Mean (Std)</td>
<td>SR</td>
</tr>
</tbody>
</table>
Table 6: Comparison of OX algorithms.

<table>
<thead>
<tr>
<th></th>
<th>Mean (Std)</th>
<th>SR</th>
<th>Mean (Std)</th>
<th>SR</th>
<th>Mean (Std)</th>
<th>SR</th>
</tr>
</thead>
<tbody>
<tr>
<td>f₁</td>
<td>1.58E−16</td>
<td>100%</td>
<td>1.55E−16</td>
<td>100%</td>
<td>8.84E−6</td>
<td>100%</td>
</tr>
<tr>
<td>f₂</td>
<td>4.38E−12</td>
<td>100%</td>
<td>5.76E−31</td>
<td>100%</td>
<td>5.25E−46</td>
<td>100%</td>
</tr>
<tr>
<td>f₃</td>
<td>6.41E−07</td>
<td>0%</td>
<td>2.17E−04</td>
<td>0%</td>
<td>1.81E−11</td>
<td>100%</td>
</tr>
<tr>
<td>f₄</td>
<td>1.49E+00</td>
<td>0%</td>
<td>2.81E+00</td>
<td>0%</td>
<td>2.39E−01</td>
<td>0%</td>
</tr>
<tr>
<td>f₅</td>
<td>1.59E−01</td>
<td>96%</td>
<td>4.78E−01</td>
<td>88%</td>
<td>2.61E−01</td>
<td>0%</td>
</tr>
<tr>
<td>f₆</td>
<td>0.00E+00</td>
<td>100%</td>
<td>0.00E+00</td>
<td>100%</td>
<td>0.00E+00</td>
<td>100%</td>
</tr>
<tr>
<td>f₇</td>
<td>2.95E−03</td>
<td>100%</td>
<td>1.35E−03</td>
<td>100%</td>
<td>9.41E−04</td>
<td>100%</td>
</tr>
<tr>
<td>f₈</td>
<td>0.00E+00</td>
<td>100%</td>
<td>2.84E+01</td>
<td>80%</td>
<td>0.00E+00</td>
<td>100%</td>
</tr>
<tr>
<td>f₉</td>
<td>4.06E+00</td>
<td>0%</td>
<td>1.99E+01</td>
<td>80%</td>
<td>1.23E−01</td>
<td>84%</td>
</tr>
<tr>
<td>f₁₀</td>
<td>2.99E−09</td>
<td>100%</td>
<td>3.52E−15</td>
<td>100%</td>
<td>2.66E−15</td>
<td>100%</td>
</tr>
<tr>
<td>f₁₁</td>
<td>1.48E−03</td>
<td>80%</td>
<td>0.00E+00</td>
<td>100%</td>
<td>0.00E+00</td>
<td>100%</td>
</tr>
<tr>
<td>f₁₂</td>
<td>6.94E−18</td>
<td>100%</td>
<td>1.57E−32</td>
<td>100%</td>
<td>1.57E−32</td>
<td>100%</td>
</tr>
<tr>
<td>f₁₃</td>
<td>4.07E−17</td>
<td>100%</td>
<td>1.35E−32</td>
<td>100%</td>
<td>1.35E−32</td>
<td>100%</td>
</tr>
<tr>
<td>f₁₄</td>
<td>1.24E−16</td>
<td>100%</td>
<td>3.64E−16</td>
<td>100%</td>
<td>8.88E−18</td>
<td>100%</td>
</tr>
<tr>
<td>f₁₅</td>
<td>5.12E−19</td>
<td>100%</td>
<td>1.56E−04</td>
<td>0%</td>
<td>3.35E−04</td>
<td>0%</td>
</tr>
<tr>
<td>f₁₆</td>
<td>1.81E−15</td>
<td>100%</td>
<td>4.44E−16</td>
<td>100%</td>
<td>3.46E−16</td>
<td>100%</td>
</tr>
<tr>
<td>f₁₇</td>
<td>1.62E−11</td>
<td>100%</td>
<td>1.67E−16</td>
<td>100%</td>
<td>1.67E−16</td>
<td>100%</td>
</tr>
<tr>
<td>f₁₈</td>
<td>4.97E−15</td>
<td>100%</td>
<td>9.02E−15</td>
<td>100%</td>
<td>2.91E−15</td>
<td>100%</td>
</tr>
<tr>
<td>f₁₉</td>
<td>4.44E−16</td>
<td>100%</td>
<td>3.59E−14</td>
<td>100%</td>
<td>0.00E+00</td>
<td>100%</td>
</tr>
<tr>
<td>f₂₀</td>
<td>9.51E−03</td>
<td>92%</td>
<td>6.47E−08</td>
<td>96%</td>
<td>9.51E−03</td>
<td>92%</td>
</tr>
<tr>
<td>f₂₁</td>
<td>2.03E−01</td>
<td>0%</td>
<td>7.79E−02</td>
<td>0%</td>
<td>2.34E+00</td>
<td>24%</td>
</tr>
<tr>
<td>f₂₂</td>
<td>1.68E−07</td>
<td>12%</td>
<td>2.71E−01</td>
<td>0%</td>
<td>2.67E−01</td>
<td>92%</td>
</tr>
<tr>
<td>f₂₃</td>
<td>2.78E−07</td>
<td>16%</td>
<td>3.67E−03</td>
<td>0%</td>
<td>2.94E−01</td>
<td>92%</td>
</tr>
</tbody>
</table>

* a: The value of t with 48 degrees of freedom is significant at 0.05 confidence level by two-tailed test.
* b: The corresponding algorithm is better than our proposed OXBBO method.

The convergence rate of OXBBO is faster than that of DE/BBO and eBBO.

From Table 4, we can also find that the average SR of eBBO, DE/BBO, and OXBO are 76.8%, 84.6%, and 84.6%, respectively. OXBO and DE/BBO reach the VTR with a successful ratio of 100% on 11 functions except function f₃ and f₅. While eBBO on 9 functions. So, OXBO and DE/BBO are more robust than eBBO.

5.5. Effect of Dimensionality on the Performance. From the analysis above, OXBO is a BBO algorithm with robust effective performance. In order to investigate the influence of scalability on the performance of OXBO, we carry out a scalability study comparison with DE/BBO for the scalable functions. For functions f₁−f₁₅, Dim = 10, 50, 100, and 200. The results of 50 independent runs are recorded in Table 5 after Dim × 10000 NFFEs.

We can find, from Table 5, that the overall SR is decreasing for both DE/BBO and OXBO upon increasing the dimension. For OXBO, the average SR for solving function with dimension 10, 50, 100, and 200 is 92%, 77%, 69%, and 67%, respectively, whereas the SR of DE/BBO for solving these functions with the same dimension is 92%, 77%, 72%, and 57%, respectively. Obviously, the average SR of DE/BBO decreases more than that of OXBO does upon increasing dimension, which can also be proved when we carefully compare the mean fitness values accordingly. By carefully looking at the results, we can recognize that for unimodal function f₁, DE/BBO is better than OXBO at dimensions 10 and 50. However, DE/BBO is outperformed by OXBO at high dimensions (100 and 200). Moreover, the precision of f₁ obtained by DE/BBO is decreased very much upon increasing dimension, while that obtained by OXBO is increased upon increasing dimension. Another example, for multimodal function f₈, which is very difficult to locate its global optima position, OXBO can find its global solution at different dimensions in Table 5, but DE/BBO cannot do it. The experimental test shows that OXBO can also solve higher dimensionality function effectively. Therefore, we can know that the operators used in OXBO can enhance the exploration ability but do not offset its exploitation ability.

5.6. Comparison with Other OX-Based Algorithms. In this section, we compare the performance of OXBO with two excellent OX-based algorithms: OXDE [28], which is proposed by embedding QOX into DE, and OLPSO [4], which is presented through using an orthogonal learning strategy for PSO to overcome the “oscillation” phenomenon of traditional PSO.

Now, we apply OX-based algorithms to solve 13 test functions adopted in this paper under 25 independent runs.
In this paper, a new improved BBO algorithm (OXBBO) is presented through the combination of the QOX operator. The convergence curves and boxplot figures are listed in Figures 2 and 3 separately. From Table 6, we can find that the results of OXBBO are significantly better than those of OXDE, OLPSO in 12, and 10 out of 23 functions, while this number for OXDE, OLBBO is 3, 3 out of 23 functions. In terms of the success rate, the average SR of OXDE, OLPSO, and OXBBO is 74%, 71%, and 82%, respectively. By carefully looking at the results in Table 6, one can note that OXDE, OLPSO, and OXBBO achieve a 100% success rate for \( f_1, f_2, f_3, f_4, f_5, f_6, f_7, f_8, f_9, f_{11}, f_{12}, f_{13}, f_{16}, f_{17}, f_{18}, f_{19}, f_{20}, f_{21}, f_{22}, f_{23} \). Additionally, for function \( f_4, f_5, f_6, f_7, f_8, f_9, f_{10}, f_{11}, f_{12}, f_{13}, f_{16}, f_{17}, f_{18}, f_{19}, f_{20}, f_{21}, f_{22}, f_{23} \), OXBBO surpass OXDE and OLPSO. For \( f_4, f_6, f_7, f_8, f_9, f_{10}, f_{11}, f_{12}, f_{13}, f_{16}, f_{17}, f_{18}, f_{19}, f_{20}, f_{21}, f_{22}, f_{23} \), OXBBO and OLPSO achieve the optimal, but the successful rate of OXBBO is higher than that of OLPSO. For \( f_5 \), OXDE and OLPSO can reach the optimal, but OXBBO failed.

From Figures 2 and 3, we can see that OXBBO is not only converging faster than both OXDE and OLPSO, but also converging more robust than both two. From the analysis, we can conclude that the overall performance of OXBBO is better than OXDE and OLPSO.

5.7. Analysis the Performance of QOX. From the analysis above, we know that the proposed algorithm OXBBO possesses very good performance through comparison with improved BBO algorithms, hybrid BBO algorithms, and some state-of-the-art orthogonal-based algorithms. Does QOX really play a key role of OXBBO? For these reasons, in this section, a detailed analysis is made to analyze the performance of QOX operator in BBO. We consider two variants of OXBBO: OXBBO with QOX operator, OXBBO without QOX operator. These algorithms are denoted as OXBBO, OXBBO1. The experimental test performs on 23 test functions with 25 independent runs. The same parameter setting suggested in Section 5.1. The mean and standard deviation of the function error values and \( t \)-test values of 25 independent runs for each algorithm have been listed in Table 7. The convergence curves and boxplot figures are listed in Figures 4 and 5.

According to the \( t \)-test, the results of OXBBO are significantly better than those of OXBBO1 in 11 out of 23 test functions, and the results of OXBBO are similar to those of OXBBO1 in 11 out of 23 functions. OXBBO1 surpasses OXBBO on only one function. From this, we know that QOX has played a very important role in enhancing the performance of OXBBO, which not only improves the solution precision but also enhances the reliability. This can also be demonstrated in Figures 4 and 5.

6. Conclusions and Future Work

In this paper, a new improved BBO algorithm (OXBBO) is present through the combination of the QOX operator
Figure 2: Continued.
Figure 2: (a) Convergence curves of mean fitness error values of OXBBO, OXDE, and OLPSO for functions $f_1, f_2, f_3, f_4, f_5, f_7, f_9, f_{10}$, and $f_{11}$. (b) Convergence curves of mean fitness error values of OXBBO, OXDE, and OLPSO for functions $f_{12}, f_{13}, f_{14}, f_{15}, f_{16}, f_{17}, f_{18}, f_{19}, f_{21}$, and $f_{22}$.
Figure 3: Continued.
Figure 3: (a) ANOVA tests of fitness error values of functions $f_1, f_2, f_3, f_4, f_5, f_6, f_7, f_8, f_9, f_{10}, f_{12}$, and $f_{13}$ for OXBBO, OXDE, and OLPSO. (b) ANOVA tests of fitness error values of functions $f_{14}, f_{15}, f_{16}, f_{17}, f_{18}, f_{19}, f_{20}, f_{21}, f_{22}$, and $f_{23}$ for OXBBO, OXDE, and OLPSO.
Figure 4: Convergence curves of mean fitness error values of OXBBO and OXBBO1 for functions \( f_1, f_3, f_5, f_7, f_9, f_{11}, f_{10}, f_{18}, \) and \( f_{22}. \)
Figure 5: ANOVA tests of fitness error values of functions $f_1, f_3, f_5, f_9, f_{12}, f_{14}, f_{15}, f_{16}, f_{20},$ and $f_{21}$ for OXBBO and OXBBO1.
into BBO algorithm, which is designed to overcome the shortcoming that migration operator only visits one vertex of the hyperrectangle defined by the immigration and emigration vectors. In OXBBBO, QOX is able to make a systematic and rational search in the region and enhance the exploration ability. Moreover, a modified migration operator is used in OXBBBO to improve the population diversity. In this paper, extensive experiments have been implemented to compare the performance of OXBBBO with state-of-the-art BBO algorithms and orthogonal crossover-based evolutionary algorithms. We have also experimentally studied the effect of QOX on the performance of OXBBBO.

We have observed that the results obtained by OXBBBO differ very much with different population sizes, which maybe due to the search frequency of QOX, similar results in [28]. In the future, we will study the effect of the search frequency of QOX on the performance in BBO framework for large scale dimension test functions. Furthermore, we would like to point out that experimental tests are based on the no-rotation functions; studying the performance on rotation functions is also our future work.

Acknowledgments

The authors would like to thank Doctor Zhan for providing the OLPSO code. The authors wish to make the acknowledgement of the area editor and the anonymous reviewers for their valuable comments and suggestions on this paper. This work is proudly supported in part by the National Natural Science Foundation of China (no. 60974082, 11010101, 11226219, and 51174263), Foundation of National Natural Science Foundation of Guangxi Region (no. 2013GXNSFBA019008, 2013GXNSFAA019003), and Project supported by Program to Sponsor Teams for Innovation in the Construction of Talent Highlands in Guangxi Institutions of Higher Learning.

References


Submit your manuscripts at
http://www.hindawi.com