Information Sciences 185 (2012) 153–177

Contents lists available at SciVerse ScienceDirect

Information Sciences

journal homepage: www.elsevier.com/locate/ins

Enhancing the search ability of differential evolution through orthogonal crossover

Yong Wang [a,*], Zixing Cai [a], Qingfu Zhang [b]

[a] School of Information Science and Engineering, Central South University, Changsha 410083, PR China
[b] School of Computer Science and Electronic Engineering, University of Essex, Colchester CO4 3SQ, UK

ARTICLE INFO

Article history:
Received 8 September 2010
Received in revised form 28 April 2011
Accepted 4 September 2011
Available online 8 September 2011

Keywords:
Differential evolution
Orthogonal crossover
Orthogonal design
Global numerical optimization

ABSTRACT

Differential evolution (DE) is a class of simple yet powerful evolutionary algorithms for global numerical optimization. Binomial crossover and exponential crossover are two commonly used crossover operators in current popular DE. It is noteworthy that these two operators can only generate a vertex of a hyper-rectangle defined by the mutant and target vectors. Therefore, the search ability of DE may be limited. Orthogonal crossover (OX) operators, which are based on orthogonal design, can make a systematic and rational search in a region defined by the parent solutions. In this paper, we have suggested a framework for using an OX in DE variants and proposed OXDE, a combination of *DE*/*rand*/1/*bin* and OX. Extensive experiments have been carried out to study OXDE and to demonstrate that our framework can also be used for improving the performance of other DE variants.

1. Introduction

Differential evolution (DE), proposed by Storn and Price in 1995 [36,37], is a class of simple yet efficient evolutionary algorithms (EAs) for continuous optimization problems. It has been successfully used in various applications (e.g., [2,8,21,30,46]). Like other EAs, DE is a population-based stochastic optimization method. It adopts mutation and crossover operators to search for new promising areas in the search space. The commonly used crossover operators in DE are binomial crossover and exponential crossover. Note, however, that these two crossover operators can only generate one new solution, which is a vertex of a hyper-rectangle defined by two parent solutions. This could somehow limit the search ability of DE.

Observing that the reproduction of new solutions in EAs can be considered as "experiments", Zhang and his co-workers [50,51] used experimental design methods to design genetic operators and proposed orthogonal crossover (OX). OX operators can make a systematic and statistically sound search in a region defined by parent solutions. OX operators have been successfully applied in various optimization problems. Leung and Wang [24] introduced a quantization technique to OX for dealing with numerical optimization. Experimental studies show that their quantization OX (QOX) operator is effective and efficient in a number of numerical optimization test instances.

The main purposes of this paper are twofold: (1) to reveal the limitation of the commonly used crossover operators in DE, and (2) to verify that the search ability of DE can be enhanced by effectively probing the hyper-rectangle defined by the mutant and target vectors. To achieve the second purpose, we have suggested a framework for using QOX in DE variants and presented an OX-based DE (OXDE), which is a combination of *DE*/*rand*/1/*bin* and QOX, as an instantiation of our framework. OXDE is simple and easy to implement. It uses QOX to complement binomial crossover or exponential crossover for searching some promising regions in the solutions space. Experimental results indicate that QOX significantly improves the perfor-

* Corresponding author.
E-mail address: ywang@csu.edu.cn (Y. Wang).

0020-0255/$ - see front matter © 2011 Elsevier Inc. All rights reserved.
doi:10.1016/j.ins.2011.09.001

mance of *DE*/*rand*/1/*bin*. Some effort has also been made not to increase the number of control parameters in our framework. Moreover, we also show that our framework can be used for improving the performance of several other DE variants.

The remainder of this paper is organized as follows. Section 2 introduces DE and Section 3 reviews the related work. In Section 4, the idea of OX operators is briefly explained. Our framework and an implementation, that is, OXDE, are presented in Section 5. In Section 6, extensive experiments have been carried out to test OXDE and to study the effectiveness of our framework on several other DE variants. Some discussions on OXDE have also been provided in this section. Finally, we conclude this paper in Section 7.

2. Differential evolution

DE is for solving the following continuous global optimization problem:

$$\text{minimize } f(\vec{x}), \vec{x} = (x_1, \ldots, x_D) \in S = \prod_{i=1}^{D} [a_i, b_i] \tag{1}$$

where $f(\vec{x})$ is continuous and $\forall i \in \{1, \ldots, D\}$, $-\infty < a_i < b_i < +\infty$.

DE maintains a population of *NP* individual members, where *NP* is the population size, and each member is a point in the solution space *S*. DE improves its population generation by generation. It extracts distance and direction information from the current population for generating new solutions for the next generation. Almost all the DE variants adopt the following algorithmic framework:

Step 1. Set the current generation number $G = 0$.
Step 2. Sample *NP* points $\vec{x}_{i,G}, \ldots, \vec{x}_{NP,G}$ from *S* to form an initial population.
Step 3. For $i = 1, \ldots, NP$, do.
  Step 3.1. Mutation: Generate a mutant vector $\vec{v}_{i,G}$ by using a DE mutation operator.
  Step 3.2. Repair: If $\vec{v}_{i,G}$ is not feasible (i.e., not in *S*), use a repair operator to make $\vec{v}_{i,G}$ feasible.
  Step 3.3. Crossover: Mix $\vec{x}_{i,G}$ and $\vec{v}_{i,G}$ to generate a trial vector $\vec{u}_{i,G}$ by using a DE crossover operator.
  Step 3.4. Selection and replacement: If $f(\vec{u}_{i,G}) \leqslant f(\vec{x}_{i,G})$, set $\vec{x}_{i,G+1} = \vec{u}_{i,G}$, otherwise, set $\vec{x}_{i,G+1} = \vec{x}_{i,G}$.
Step 4. If a preset stopping condition is not met, set $G = G + 1$ and go to Step 3.

In the *i*th pass of the loop in Step 3, $\vec{x}_{i,G}$ is called a target vector in the literature, $\vec{v}_{i,G}$ is its mutant vector, and $\vec{u}_{i,G}$ is its trial vector. $\vec{u}_{i,G}$ inherits some parameter values from $\vec{x}_{i,G}$ in Step 3.3 and enters the next generation if its objective function value is better than or equal to the objective function value of $\vec{x}_{i,G}$.

The characteristic feature of DE is its mutation operators. Two commonly used mutation operators are:

- ***DE*/*rand*/1 mutation**:

$$\vec{v}_{i,G} = \vec{x}_{r1,G} + F \cdot (\vec{x}_{r2,G} - \vec{x}_{r3,G}) \tag{2}$$

- ***DE*/*rand*/2 mutation**:

$$\vec{v}_{i,G} = \vec{x}_{r1,G} + F \cdot (\vec{x}_{r2,G} - \vec{x}_{r3,G}) + F \cdot (\vec{x}_{r4,G} - \vec{x}_{r5,G}) \tag{3}$$

where *r*1, *r*2, *r*3, *r*4, and *r*5 are different indexes uniformly randomly selected from $\{1, \ldots, NP\} \backslash \{i\}$, and *F* is a control parameter, often called the scaling factor in the literature.

DE performs a crossover operator on $\vec{x}_{i,G}$ and $\vec{v}_{i,G}$ to generate the trial vector $\vec{u}_{i,G}$. The following two crossover operators are widely used in the DE implementations.

- **Binomial crossover**: The trial vector $\vec{u}_{i,G} = (u_{i,1,G}, u_{i,2,G}, \ldots, u_{i,D,G})$ is generated in the following way:

$$u_{i,j,G} = \begin{cases} v_{i,j,G} & \text{if } rand_j(0,1) \leqslant CR \quad \text{or} \quad j = j_{rand} \\ x_{i,j,G} & \text{otherwise} \end{cases} \tag{4}$$

where index j_{rand} is a randomly chosen integer in the range $[1, D]$, $rand_j(0,1)$ is a uniform random number in $(0,1)$, and $CR \in (0,1]$ is the user-defined crossover control parameter. Due to the use of j_{rand}, $\vec{u}_{i,G}$ is always different from $\vec{x}_{i,G}$.

- **Exponential crossover**: The trial vector $\vec{u}_{i,G} = (u_{i,1,G}, u_{i,2,G}, \ldots, u_{i,D,G})$ is created as follows [35]:

$$u_{i,j,G} = \begin{cases} v_{i,j,G} & \text{for} \quad j = \langle l \rangle_D, \langle l+1 \rangle_D, \ldots, \langle l+L-1 \rangle_D \\ x_{i,j,G} & \text{otherwise} \end{cases} \tag{5}$$

where $i = 1, 2, \ldots, NP$, $j = 1, 2, \ldots, D$, and $\langle\rangle_D$ denotes the modulo function with modulus *D*. The starting index *l* is a randomly chosen integer in the range of $[1, D]$. The integer *L* is also drawn from the range $[1, D]$ with the probability $P_r(L \geqslant v) = CR^{v-1}$, $v > 0$. The parameters *l* and *L* are re-generated for each trial vector $\vec{u}_{i,G}$.

- **Repair operator**: A simple and popular repair operator works as follows: if the *j*th element $v_{i,j,G}$ of the mutant vector $\vec{v}_{i,G} = (v_{i,1,G}, v_{i,2,G}, \ldots, v_{i,D,G})$ is out of the search region $[a_j, b_j]$, then $v_{i,j,G}$ is reset as follows:

$$v_{i,j,G} = \begin{cases} \min\{b_j, 2a_j - v_{i,j,G}\} & \text{if } v_{i,j,G} < a_j \\ \max\{a_j, 2b_j - v_{i,j,G}\} & \text{if } v_{i,j,G} > b_j \end{cases} \quad (6)$$

The illustration of DE in the two-dimensional search space has been given in Fig. 1. From Fig. 1, it is evident that the trial vector is a vertex of the hyper-rectangle defined by the mutant and target vectors, regardless of the binomial crossover and the exponential crossover.

3. The related work

Much effort has been made to improve the performance of DE and a number of DE variants have been proposed. These improvements can be classified into four categories:

(1) Dynamic adaptation or self-adaptation of the control parameters (i.e., *NP*, *F*, and *CR*) in DE to ease the task of control parameter setting and to dynamically or self-adaptively adjust the search behavior of DE to suit different landscapes [1,6,9,23,26,32,33,41,47,52]. For example, Lee et al. [23] estimated the influence of the difference vector on the objective function and utilized such information to find a better scaling factor *F*. Liu and Lampinen [26] introduced a fuzzy adaptive DE (FADE), which adapts the control parameters of DE by fuzzy logic controller. FADE outperforms the classic DE on high-dimensional test instances. Das et al. [9] proposed two schemes to adapt the scaling factor *F*. In the first scheme, time-varying scaling factor *F* is introduced. In the second scheme, the scaling factor *F* is adapted in a random way. Ali and Torn [1] studied a method for adjusting the scaling factor *F*, in which the search is diversified at its early stage and intensified at its later stage. Brest et al. [6] presented a novel approach named jDE to self-adapt control parameters *F* and *CR*. In jDE, these two control parameters are encoded at the individual level. The implementation of jDE is simple; however, its performance is better than or at least comparable to the classic DE. In SaDE [32,33], the trial vector generation strategies and two control parameters (*F* and *CR*) are dynamically adjusted based on their performance. The experimental results suggest that SaDE [32] is significantly better than the classic DE and three adaptive DE variants in 26 test instances. An adaptive DE with optional external archive, called JADE, is proposed by Zhang and Sanderson [52]. In this method, the most recent successful *F* and *CR* are used to guide the setting of new *F* and *CR*. The performance of JADE is quite competitive in 20 test instances. Weber et al. [47] proposed four simple schemes to update the scaling factor in the distributed DE and studied the impact of the schemes on the performance of the algorithms. Teo [41] presented a first attempt to deal with the population sizing problem through self-adaptation. In this approach, the concepts of absolute encoding and relative encoding are introduced.

(2) Introduction of new mutation and crossover operators for generating new solutions [5,7,12,14,16,18,28,52]. For example, Zhang and Sanderson [52] proposed a new mutation operator named "DE/current-to-*p*best" in JADE. In "DE/current-to-*p*best", the recently explored inferior solutions are stored in an archive, with the aim of providing the information of progress direction. Due to this relatively greedy mutation operator, JADE is very efficient in solving unimodal test instances. Moreover, JADE with archive can produce promising results for high-dimensional test instances. Feoktistov and Janaqi [14] generalized the mutation operators of DE into four groups according to the use of the knowledge of the objective function. Based on the suggestion in [14], the user could design the best mutation operator for a given test instance. Mezura-Montes et al. [28] presented a new mutation operator which combines the information of both the best solution in the population and the parent solution to create new mutant vectors. Fan and Lampinen [12] proposed a trigonometric mutation operator which can be viewed as a local search operator. This operator utilizes the information of objective function to produce the mutant vectors. Moreover, Fan and Lampinen [12] used

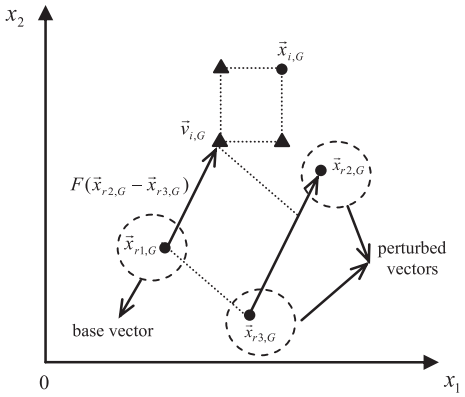

Fig. 1. Illustration of DE. $\vec{x}_{i,G}$ is the target vector, $\vec{x}_{r1,G}$, $\vec{x}_{r2,G}$, and $\vec{x}_{r3,G}$ are mutually exclusive vectors randomly chosen from the population, which are also different from $\vec{x}_{i,G}$, $\vec{v}_{i,G}$ is the mutant vector, and the triangle points represent the trial vectors.

an extra parameter M_t to control the frequency of the use of this operator. Das et al. [7] proposed a global and local neighborhood-based DE (DEGL), in which a neighborhood-based mutation operator is introduced and a local neighborhood model is combined with a global neighborhood model. It is necessary to note that this neighborhood-based mutation operator depends on a user-defined weight factor. The effectiveness of DEGL has been verified by comparing it with five DE variants and four other EAs in 24 benchmark test instances and two real-world problems. Bhowmik et al. [5] designed a new mechanism to select the vectors from the population for mutation, based on the Lagrange's mean value theorem. Gong et al. [18] used the four mutation operators proposed in JADE to construct the candidate pool and adaptively selected a more suitable mutation operator for a problem at hand during the evolution. Ghosh et al. [16] adapted the control parameters based on the objective function values, the aim of which is to achieve a better trade-off between the explorative and exploitative abilities of DE.

(3) Hybridization of DE with other operators [22,29,31,34,39]. For example, Sun et al. [39] developed DE/EDA, which combines DE with estimation of distribution algorithm (EDA). In DE/EDA, one part of the trial vector is generated by DE and the other part of the trial vector is sampled from the search space by EDA. Norman and Iba [31] incorporated a crossover-based adaptive local search operator into DE. The experimental results confirm that implementing the crossover-based adaptive local search operator on the best individual of the population can significantly enhance the performance of DE. Rahnamayan et al. [34] proposed an opposition-based DE (ODE) which employs opposition-based learning for population initialization and for generating new solutions. The experimental results indicate that the convergence speed and the solution accuracy of DE can be clearly improved by making use of the opposition-based learning. Jia et al. [22] utilized a chaotic local search to improve the optimizing performance of DE, which is capable of exploring and exploiting the search space in the early stage and in the later stage, respectively. Neri et al. [29] proposed a disturbed exploitation compact DE, which is a compact memetic computing paradigm. This method employs two mutation operators as the exploitative search mechanisms and disturbs the probability vector with a probability M_p to avoid premature convergence.

(4) Use of multiple populations [40,48]. For example, Tasgetiren and Suganthan [40] presented a multi-population DE for constrained optimization. In this method, the population is divided into several subpopulations and the subpopulations are regrouped to provide information exchange during some generations. Yang et al. [48] combined a cooperation coevolution framework with a self-adaptive neighborhood search DE. This method is very effective for solving large scale test instances (up to 1000 dimensions) based on the experimental results.

Besides the above methods, Mallipeddi et al. [27] recently proposed an ensemble framework of DE, in which a pool of mutation operators and a pool of control parameters (F and CR) are used to generate the offspring. Wang et al. [43] recently presented a composite DE (CoDE), which randomly combines three trial vector generation strategies (i.e., the mutation and crossover operators) with three control parameter settings to generate the trial vectors. Moreover, in CoDE, three trial vectors are created for each target vector. Recently, Das and Suganthan [10] carried out a detailed survey on the state-of-the-art of DE, in terms of its basic concepts, major variants, and applications.

It is necessary to emphasize that our work falls in the second category. We also aim at not increasing dramatically the number of control parameters in the algorithm.

4. Orthogonal crossover

4.1. Orthogonal design

Consider a system whose cost depends on K factors (i.e., variables), each factor can take one of Q levels (i.e., values). To find the best level for each factor to minimize the system cost, one can do one experiment for every combination of factor levels and then select the best one if K and Q are small. The number of all the combinations is Q^K. Therefore, it is not possible or efficient to test all the combinations in the case when K and Q are large. Experimental design methods can be used for sampling a small number of well representative combinations for testing [13]. The orthogonal design is one of several very popular experimental design tools. It provides a series of orthogonal arrays for accommodating different numbers of factors and different levels. The algorithm for constructing these arrays can be found in [13], and a number of such arrays can be found in http://www2.research.att.com/~njas/oadir/. An orthogonal array for K factors with Q levels and M combinations is often denoted by $L_M(Q^K)$. As an example, $L_9(3^4)$ is shown as follows:

$$L_9(3^4) = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 2 & 2 & 2 \\ 1 & 3 & 3 & 3 \\ 2 & 1 & 2 & 3 \\ 2 & 2 & 3 & 1 \\ 2 & 3 & 1 & 2 \\ 3 & 1 & 3 & 2 \\ 3 & 2 & 1 & 3 \\ 3 & 3 & 2 & 1 \end{bmatrix} \tag{7}$$

Each row in this array represents a combination of levels, that is, an experiment. For example, the last row stands for an experiment in which factor 1 is at level 3, factor 2 at level 3, factor 3 at level 2, and factor 4 at level 1. Based on this array, one can carry out 9 experiments for estimating a good combination of factor levels.

The orthogonality of an orthogonal array means that: (1) each level of the factor occurs the same number of times in each column, and (2) each possible level combination of any two given factors occurs the same number of times in the array.

4.2. Orthogonal crossover

The major idea behind the orthogonal crossover (OX) operators [50,51] is that each trial solution in a search algorithm can be regarded as an experiment, and a genetic operator (such as crossover and mutation) is a procedure for sampling several representative points (i.e., experiments) from a region defined by the parent solutions; therefore, orthogonal design or any other experimental design tools can be used to make a genetic operator more statistically sound [50,51]. Leung and Wang [24] introduced a quantization technique into OX and proposed a version of OX, which we call QOX, for dealing with numerical optimization. In this paper, we will employ QOX in our algorithm.

We now explain how QOX based on $L_M(Q^K)$ works. Given two parent solutions $\vec{e} = (e_1, \ldots, e_D)$ and $\vec{g} = (g_1, \ldots, g_D)$. $\vec{e}$ and $\vec{g}$ define a search range $[\min(e_i, g_i), \max(e_i, g_i)]$ for variable x_i. QOX first does *quantization* for this search range and defines Q levels $l_{i1}, l_{i2}, \ldots, l_{iQ}$ for x_i as follows:

$$l_{i,j} = \min(e_i, g_i) + \frac{j-1}{Q-1} \cdot (\max(e_i, g_i) - \min(e_i, g_i)), \quad j = 1, \ldots, Q \tag{8}$$

The search space defined by $\vec{e}$ and $\vec{g}$ will have Q^D points after quantization since each factor has Q possible levels. Fig. 2 shows an example of quantization. Suppose we have two parents $\vec{e} = (1.0, 3.0)$ and $\vec{g} = (3.0, 1.0)$ in the two-dimensional search space. The search space defined by these two parents is $[1.0, 3.0] \times [1.0, 3.0]$. If $Q = 3$, the search space will contain $Q^D = 3^2 = 9$ points after quantization, as shown in Fig. 2, since each factor is quantized into three levels.

Since D is often much larger than K, one cannot directly apply $L_M(Q^K)$. To overcome this difficulty, as other OX operators [50,51] do, QOX divides $(x_1, \ldots, x_D)$ into K subvectors:

$$\begin{cases} \vec{H}_1 = (x_1, \ldots, x_{t_1}) \\ \vec{H}_2 = (x_{t_1+1}, \ldots, x_{t_2}) \\ \cdots \\ \vec{H}_K = (x_{t_{K-1}+1}, \ldots, x_D) \end{cases} \tag{9}$$

where integers $t_1, t_2, \ldots, t_{K-1}$ are randomly generated such that $1 < t_1 < t_2 < \cdots < t_{K-1} < D$.

QOX treats each $\vec{H}_i$ as a factor and defines the following Q levels for $\vec{H}_i$:

$$\begin{cases} \vec{L}_{i1} = (l_{t_{i-1}+1,1}, l_{t_{i-1}+2,1}, \ldots, l_{t_i,1}) \\ \vec{L}_{i2} = (l_{t_{i-1}+1,2}, l_{t_{i-1}+2,2}, \ldots, l_{t_i,2}) \\ \cdots \\ \vec{L}_{iQ} = (l_{t_{i-1}+1,Q}, l_{t_{i-1}+2,Q}, \ldots, l_{t_i,Q}) \end{cases} \tag{10}$$

Then, QOX uses $L_M(Q^K)$ on factors $\vec{H}_1, \ldots, \vec{H}_K$ to construct M solutions (i.e., combinations of levels).

Note that if D is smaller than K, the first D columns of $L_M(Q^K)$ can be used to design OX directly. For example, when $L_9(3^4)$ is adopted, the nine offspring produced by QOX are the nine quantized points as shown in Fig. 2.

In the following, we give an example of QOX based on $L_9(3^4)$.

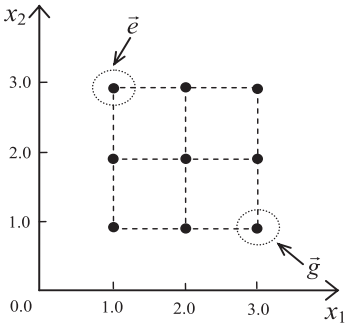

Fig. 2. Illustration of quantization in the two-dimensional search space. The square is the search region defined by $\vec{e}$ and $\vec{g}$ and the nine dots are obtained after quantization.

Suppose that $\vec{e} = (2.0, 5.0, 1.0, 0.0, 8.0, 3.0)$ and $\vec{g} = (4.0, 3.0, 2.0, 6.0, 2.0, 5.0)$. Therefore, the three levels for x_1 are 2.0, 3.0, and 4.0; the three levels for x_2 are 3.0, 4.0, and 5.0; the three levels for x_3 are 1.0, 1.5, and 2.0; the three levels for x_4 are 0.0, 3.0, and 6.0; the three levels for x_5 are 2.0, 5.0, and 8.0; and the three levels for x_6 are 3.0, 4.0, and 5.0.

Suppose that $t_1 = 2$, $t_2 = 4$, and $t_3 = 5$ are three integers randomly generated. Afterward, $(x_1, \ldots, x_6)$ is divided into:

$$\begin{cases} \vec{H}_1 = (x_1, x_2) \\ \vec{H}_2 = (x_3, x_4) \\ \vec{H}_3 = x_5 \\ \vec{H}_4 = x_6 \end{cases} \tag{11}$$

The three levels for $\vec{H}_1$ are: $\vec{L}_{11} = (2.0, 3.0)$, $\vec{L}_{12} = (3.0, 4.0)$, and $\vec{L}_{13} = (4.0, 5.0)$. The three levels for $\vec{H}_2$ are: $\vec{L}_{21} = (1.0, 0.0)$, $\vec{L}_{22} = (1.5, 3.0)$, and $\vec{L}_{23} = (2.0, 6.0)$. The three levels for $\vec{H}_3$ are: $\vec{L}_{31} = 2.0$, $\vec{L}_{32} = 5.0$, and $\vec{L}_{33} = 8.0$. The three levels for $\vec{H}_4$ are: $\vec{L}_{41} = 3.0$, $\vec{L}_{42} = 4.0$, and $\vec{L}_{43} = 5.0$.

Then, nine representative solutions generated based on $L_9(3^4)$ are:

$$\begin{cases} (2.0, 3.0, 1.0, 0.0, 2.0, 3.0) \\ (2.0, 3.0, 1.5, 3.0, 5.0, 4.0) \\ (2.0, 3.0, 2.0, 6.0, 8.0, 5.0) \\ (3.0, 4.0, 1.0, 0.0, 5.0, 5.0) \\ (3.0, 4.0, 1.5, 3.0, 8.0, 3.0) \\ (3.0, 4.0, 2.0, 6.0, 2.0, 4.0) \\ (4.0, 5.0, 1.0, 0.0, 8.0, 4.0) \\ (4.0, 5.0, 1.5, 3.0, 2.0, 5.0) \\ (4.0, 5.0, 2.0, 6.0, 5.0, 3.0) \end{cases} \tag{12}$$

During the past fifteen years, OX operators have been incorporated into EAs by some researchers to solve global optimization problems [20,24], multiobjective optimization problems [20,49], and constrained optimization problems [3,4,45].

In addition to orthogonal design, Wang and Dang [44] used Lartin square design to improve EAs, and Tsai et al. [42] combined a robust design approach, the Taguchi method, with genetic algorithm (GA).

5. Proposed approach

5.1. Basic idea

Crossover operators play a key role in DE. As shown in Fig. 1, both binomial and exponential crossover operators, the two most commonly used crossover operators in DE, only generate and evaluate one single trial vector $\vec{u}_{i,G}$, which is a vertex of the hyper-rectangle defined by the mutant vector $\vec{v}_{i,G}$ and the target vector $\vec{x}_{i,G}$. Thus, they do not carry out a systematic search in this hyper-rectangle, which might be a promising region in the solution space. Therefore, the search ability of DE could be limited in a sense. After revealing the above limitation of DE, one of the main purposes of this paper is to overcome such limitation.

We propose to use the QOX operator to probe the hyper-rectangle defined by the mutant vector and the target vector and, as a result, to improve the search ability of DE. Note, however, that the QOX operator needs to evaluate M new solutions if $L_M(Q^K)$ is used, and thus, it is unwise to apply QOX to every pair of mutant and target vectors. For example, if QOX is applied to every pair of mutant and target vectors, the number of function evaluations is $NP \cdot M$ at each generation. Whereas, the implementation of the classic DE only spends NP function evaluations at each generation since one trial vector is generated for each target vector. Hence, in order to achieve an effective design, experiments should be prepared so as to reduce the computational effort at each generation. In our framework, we apply QOX only once at each generation to save the computational cost and to keep the implementation simple. By doing this, the overhead of QOX in preparing experiments at each generation is relatively small, especially when the population size is large or the cost of the evaluation of objective function is expensive. Moreover, the M sampling solutions produced by QOX make DE more effective in probing the search space.

As pointed out, the use of QOX in this paper is to improve the search ability of DE. In order to add more variation to the search and reduce the number of control parameters, in our framework, the scaling factor F in the mutation operator is randomly chosen between 0 and 1 to generate a mutant vector if it will undergo QOX. For example, *DE*/*rand*/1 mutation has been revised to the following formulation for the mutant vector which will take part in QOX:

$$\vec{v}_{i,G} = \vec{x}_{r1,G} + rand(0,1) \cdot (\vec{x}_{r2,G} - \vec{x}_{r3,G}) \tag{13}$$

where $rand(0,1)$ denotes a uniformly distributed random number in $(0,1)$.

5.2. Algorithmic framework

It is important to emphasize that in this paper, instead of proposing a new DE variant, we intend to introduce a framework for improving the search ability of DE by making use of QOX. As an example of this, an orthogonal crossover based differential evolution (OXDE), which combines QOX with *DE*/*rand*/1/*bin*, is presented in Fig. 3.

The only major difference between OXDE and *DE*/*rand*/1/*bin* is that, in the former, for one randomly selected individual member $\vec{x}_{k,G}$ at each generation, QOX is applied to $\vec{x}_{k,G}$ and its mutant vector $\vec{v}_{k,G}$ to generate its trial vector $\vec{u}_{k,G}$. Step 3.1.5 and Step 3.1.7 in Fig. 3 are illustrated in Fig. 4 when $L_9(3^4)$ is used.

The above algorithmic framework can also be generalized to other DE variants by replacing *DE*/*rand*/1/*bin* with other DE variants.

Gong et al. [17] also used QOX in DE. They used QOX directly in individual solutions in the current population. They treated their QOX operator as a local search operator, and their QOX operator was independent of DE. In OXDE, QOX is embedded in DE to strengthen the search ability, and QOX works on both a mutant vector and a target vector to search the region defined by these two vectors, which could be a promising area in the search space. In a sense, OXDE attempts to combine the advantages of both QOX and DE mutations. More importantly, the main motivation of this paper is to improve the search ability of the commonly used crossover operators in DE by incorporating QOX, after realizing the inability of these crossover operators. Moreover, our method flexibly switches between QOX and binomial/exponential crossover in a straightforward manner.

6. Experimental study

6.1. Experimental settings

A suite of 24 test instances is used for our experimental studies. The first 10 test instances are widely used in the evolutionary computation community [31], and the other 14 test instances are the first 14 test instances designed for the CEC2005 Special Session on real-parameter optimization [38].

Since OXDE is presented to enhance the search ability of *DE*/*rand*/1/*bin*, the performance comparison is mainly done between *DE*/*rand*/1/*bin* and OXDE. The orthogonal array used in OXDE is $L_9(3^4)$. The settings of the parameters used in OXDE and *DE*/*rand*/1/*bin* (i.e., *NP*, *F*, *CR*, and FES_{max}) follow [31], that is, $NP = D$, $F = 0.9$, $CR = 0.9$, and $FES_{max} = 10{,}000 \times D$, where D is the number of variables.

Parameters:
- NP: population size;
- F: scaling factor;
- CR: crossover control parameter;
- $L_M(Q^K)$: orthogonal array used in QOX;
- FES_{max}: maximum number of function evaluations (FES);

Step 1) Initialization:
Step 1.1) Set the current generation number G=0.
Step 1.2) Randomly sample NP points $\vec{x}_{i,0},\ldots,\vec{x}_{NP,0}$ from S to form an initial population.
Step 1.3) Evaluate the objective function values of these points.
Step 1.4) FES=NP.
Step 2) Randomly select an index k from {1,…, NP} .
Step 3) For $i = 1,\ldots, NP$, do
Step 3.1)
If i is not k
Step 3.1.1) Mutation: Generate a mutant vector $\vec{v}_{i,G}$ by equation (2).
Step 3.1.2) Repair: If $\vec{v}_{i,G}$ is not feasible (i.e., it is not in S), use the repair operator (6) to make $\vec{v}_{i,G}$ feasible.
Step 3.1.3) Crossover: Mix $\vec{x}_{i,G}$ and $\vec{v}_{i,G}$ by equation (4), and generate a trial vector $\vec{u}_{i,G}$.
Step 3.1.4) Function Evaluation: Evaluate $f(\vec{u}_{i,G})$ and set FES=FES+1.
Else
Step 3.1.5) Mutation: Generate a mutant vector $\vec{v}_{i,G}$ by equation (13).
Step 3.1.6) Repair: If $\vec{v}_{i,G}$ is not feasible, use the repair operator (6) to make $\vec{v}_{i,G}$ feasible.
Step 3.1.7) Orthogonal Crossover: Mix $\vec{x}_{i,G}$ and $\vec{v}_{i,G}$ by making use of QOX based on $L_M(Q^K)$ to generate M trial vectors.
Step 3.1.8) Function Evaluation: Evaluate the objective function values of the M trial vectors and select the one with the smallest objective function value to be $\vec{u}_{i,G}$. Set FES=FES+M.
End If
Step 3.2) Selection and Replacement: If $f(\vec{u}_{i,G}) \le f(\vec{x}_{i,G})$, set $\vec{x}_{i,G+1} = \vec{u}_{i,G}$, otherwise, set $\vec{x}_{i,G+1} = \vec{x}_{i,G}$.
Step 4) If $FES \ge FES_{max}$, stop and output the vector with the smallest objective function value in the population, otherwise, set G=G+1 and go to Step 2.

Fig. 3. The framework of OXDE.

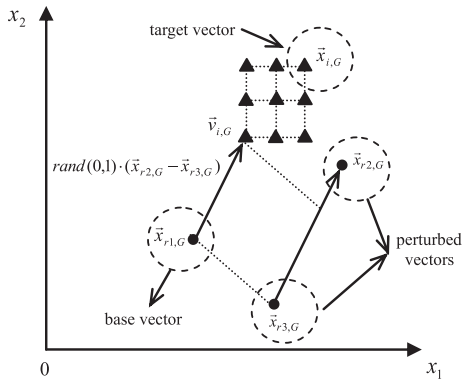

Fig. 4. Illustration of OXDE. $\vec{x}_{i,G}$ is the target vector, $\vec{x}_{r1,G}$, $\vec{x}_{r2,G}$, and $\vec{x}_{r3,G}$ are mutually exclusive vectors randomly chosen from the population, which are also different from $\vec{x}_{i,G}$, $\vec{v}_{i,G}$ is the mutant vector, and the triangle points represent the trial vectors obtained by QOX.

Fifty independent runs are carried out for each algorithm in each instance. All the experiments are performed on a computer with 1.66 GHz Dual-core Processor and 1 GB of RAM in Windows XP SP2. The average and standard deviation of the function error values $f(\vec{x}) - f(\vec{x}^*)$ among 50 runs are recorded on each instance, where $\vec{x}$ is the best solution found by the algorithm in a run and $\vec{x}^*$ is the global optimum of the test instance.

A run is successful if its function error value is not larger than the target error accuracy level ε, which is set to 10^{-2} for test instances F_6–F_{14}, and 10^{-6} for the rest of the test instances as suggested in [31]. We record the number of successful runs and report the success rate, the percentage of the successful runs among 50 runs.

In a successful run, FES_s is the number of *FES* needed for reaching the target error accuracy level. FES_s is set to FES_{max} in an unsuccessful run. We calculate the mean and standard derivation of FES_s among 50 independent runs to measure the convergence speed of the algorithm.

6.2. Comparison with DE/rand/1/bin

Table 1 presents the statistical results of the function error values obtained by *DE*/*rand*/1/*bin* and OXDE in 24 test instances with $D = 30$. Table 2 summarizes the statistics of FES_s in the instances where at least one algorithm has a successful run. The *t*-test at a 0.05 significance level has been used in comparison.

Table 1 indicates that OXDE outperforms *DE*/*rand*/1/*bin* in terms of the solution quality in 19 out of 24 test instances. There is no significant performance difference between OXDE and *DE*/*rand*/1/*bin* in the other five instances. Therefore, we can conclude that QOX greatly improves the search ability of *DE*/*rand*/1/*bin*, a classic DE variant.

The success rate and the convergence speed of *DE*/*rand*/1/*bin* and OXDE are compared in Table 2. Overall, OXDE is better than *DE*/*rand*/1/*bin* in terms of the success rate. It is also clear that OXDE needs fewer *FES* to reach the target error accuracy level than *DE*/*rand*/1/*bin* in the 11 selected test instances.

To provide more information about the convergence performance of these two algorithms, we have run the code of *DE*/*rand*/1/*bin* (which was obtained from the authors of [31]), and have plotted in Fig. 5 the evolution of the average function

Table 1
Experimental results of *DE*/*rand*/1/*bin* and OXDE over 50 independent runs for test instances with 30 variables, after 300,000 *FES*. "Mean error" and "std dev" indicate the average and standard deviation of the function error values, respectively. *t*-test is performed between OXDE and *DE*/*rand*/1/*bin*.

Inst.	*DE*/*rand*/1/*bin*	OXDE	Inst.	*DE*/*rand*/1/*bin*	OXDE
F_{sph}	5.73E−17 ± 2.03E−16‡	5.21E−59 ± 1.82E−58	F_3	3.63E+06 ± 2.06E+06‡	5.41E+05 ± 2.86E+05
F_{ros}	5.20E+01 ± 8.56E+01‡	4.78E−01 ± 1.30E+00	F_4	5.54E+01 ± 6.37E+01‡	2.58E+00 ± 3.91E+00
F_{ack}	1.37E−09 ± 1.32E−09‡	2.66E−15 ± 0.00E+00	F_5	1.08E+03 ± 5.31E+02‡	5.72E+00 ± 1.18E+01
F_{grw}	2.66E−03 ± 5.73E−03$^{\approx}$	1.82E−03 ± 4.44E−03	F_6	6.67E+01 ± 1.51E+02‡	7.97E−01 ± 1.61E+00
F_{ras}	2.55E+01 ± 8.14E+00‡	8.99E+00 ± 2.29E+00	F_7	7.59E−03 ± 8.96E−03$^{\approx}$	9.98E−03 ± 9.50E−03
F_{sch}	4.90E+02 ± 2.34E+02‡	0.00E+00 ± 0.00E+00	F_8	2.09E+01 ± 1.33E−01$^{\approx}$	2.09E+01 ± 5.48E−02
F_{sal}	2.52E−01 ± 4.78E−02‡	2.01E−01 ± 4.16E−02	F_9	2.43E+01 ± 6.23E+00‡	1.51E+01 ± 4.07E+00
F_{wht}	3.10E+02 ± 1.07E+02$^{\approx}$	3.35E+02 ± 7.83E+01	F_{10}	7.33E+01 ± 6.62E+01‡	4.70E+01 ± 4.71E+01
F_{pn1}	4.56E−02 ± 1.31E−01‡	1.03E−02 ± 7.32E−02	F_{11}	3.57E+01 ± 9.45E+00$^{\approx}$	3.36E+01 ± 1.13E+01
F_{pn2}	1.44E−01 ± 7.19E−01‡	2.25E−32 ± 6.37E−32	F_{12}	4.01E+03 ± 5.08E+03‡	2.94E+03 ± 3.61E+03
F_1	3.87E−14 ± 2.71E−14‡	1.27E−28 ± 1.84E−28	F_{13}	3.25E+00 ± 8.32E−01‡	2.02E+00 ± 6.12E−01
F_2	8.50E−02 ± 7.94E−02‡	5.69E−05 ± 6.82E−05	F_{14}	1.33E+01 ± 2.11E−01‡	1.32E+01 ± 1.96E−01

‡ indicates the *t* value is significant at a 0.05 level of significance by two-tailed *t*-test. "‡" and "≈" denote the performance of *DE*/*rand*/1/*bin* is worse than and similar to that of OXDE, respectively. The results of the first 20 test instances of *DE*/*rand*/1/*bin* are directly taken from [31].

Table 2
Experimental results of *DE*/*rand*/1/*bin* and OXDE over 50 independent runs for test instances with 30 variables, after 300,000 *FES*. "Mean FES" and "std dev" indicate the average and standard deviation of FES_s, respectively. Percentages in parentheses denote the success rates. *t*-test is performed between OXDE and *DE*/*rand*/1/*bin*.

Inst.	*DE*/*rand*/1/*bin* Mean FES ± std dev	OXDE Mean FES ± std dev	Inst.	*DE*/*rand*/1/*bin* Mean FES ± Std Dev	OXDE Mean FES ± std dev
F_{sph}	148650.8 ± 6977.7‡ (100%)	49110.4 ± 1649.4 (100%)	F_{pn1}	160955.2 ± 63176.3‡ (86%)	50991.4 ± 36166.1 (98%)
F_{ros}	–	197265.6 ± 41183.9 (88%)	F_{pn2}	156016.9 ± 31515.8‡ (96%)	51580.2 ± 3832.1 (100%)
F_{ack}	215456.1 ± 9721.4‡ (100%)	72420.4 ± 2084.9 (100%)	F_1	153450.1 ± 5780.4‡ (100%)	50011.8 ± 1413.1 (100%)
F_{grw}	190292.5 ± 63478.8‡ (76%)	91443.2 ± 91989.7 (84%)	F_6	–	196053.4 ± 55168.6 (80%)
F_{sch}	–	55573.4 ± 2304.4 (100%)	F_7	211778.8 ± 70080.3‡ (66%)	142883.0 ± 10912.9 (68%)
F_{wht}	–	295580.8 ± 31248.4 (2%)			

"‡" indicates the *t* value is significant at a 0.05 level of significance by two-tailed *t*-test. "‡" denotes that the performance of *DE*/*rand*/1/*bin* is worse than that of OXDE. "–" means the success rate is zero. The results of *DE*/*rand*/1/*bin* are directly taken from [31]. The instances on which *DE*/*rand*/1/*bin* and OXDE have zero success rate are excluded from this table.

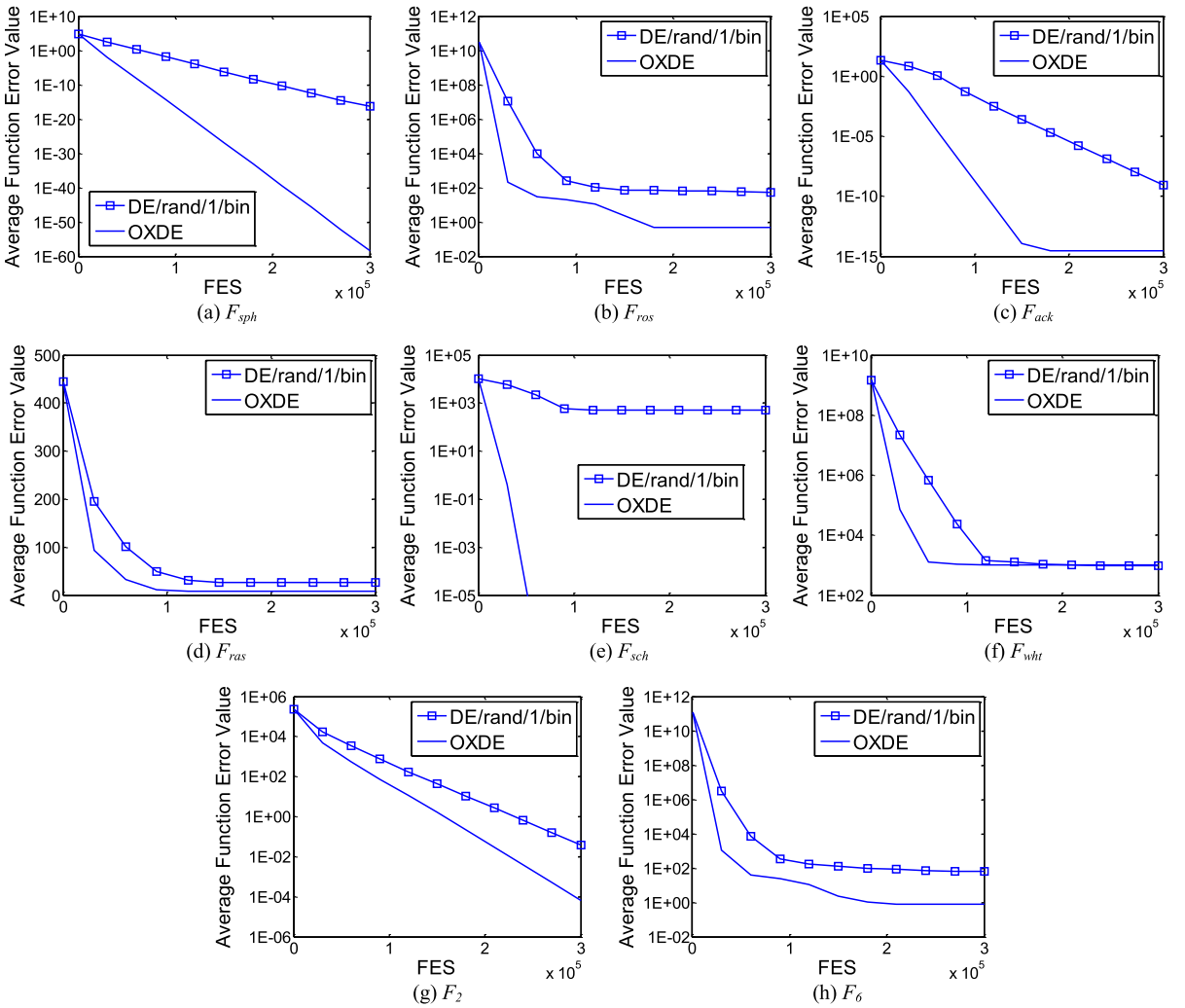

Fig. 5. Evolution of the average function error values with the number of *FES* for *DE*/*rand*/1/*bin* and OXDE on eight test instances with $D = 30$.

error values with the number of *FES* over 50 runs in eight instances. It is evident that overall, OXDE exhibits better convergence performance than *DE*/*rand*/1/*bin*.

From the above results, we can draw the conclusion that OXDE certainly outperforms *DE*/*rand*/1/*bin* in terms of both the solution quality and the convergence speed.

6.3. Effect of population size

In the above subsection, $NP = D = 30$. In the following, we study the effect of population size on the performance of *DE/rand/1/bin* and OXDE. To this end, the same experiments in the above subsection have been carried out in which $NP = 50$, 100, 200, and 300. The experimental results are presented in Table 3.

In the case of $NP = 50$ and 100, it is clear from Table 3 that the performance of OXDE is significantly better than that of *DE/rand/1/bin*. *DE/rand/1/bin* cannot beat OXDE in any test instances in terms of both the success rate and the solution quality.

It is also clear that OXDE performs significantly better than *DE/rand/1/bin* when $NP = 200$ and 300. However, one should note that no algorithms can have a single successful run in any test instances when $NP = 200$ and 300, which indicates that a large population size may severely deteriorate the performance of DE. It is because the iteration number will significantly

Table 3
Experimental results of *DE/rand/1/bin* and OXDE over 50 independent runs for test instances with 30 variables and varying population sizes, after 300,000 *FES*. "Mean error" and "Std Dev" indicate the average and standard deviation of the function error values, respectively. Percentages in parentheses denote the success rates. In the fields without parentheses, the success rates are zero. *t*-test is performed between *DE/rand/1/bin* and OXDE.

Inst.	*DE/rand/1/bin*	OXDE	Inst.	*DE/rand/1/bin*	OXDE
$NP = 50$			$NP = 100$		
F_{sph}	2.31E−02 ± 1.92E−02‡	1.35E−25 ± 1.96E−25 (100%)	F_{sph}	3.75E+03 ± 1.14E+03‡	2.94E−05 ± 9.60E−06
F_{ros}	3.70E+02 ± 4.81E+02‡	2.81E−01 ± 9.76E−01 (2%)	F_{ros}	4.03E+08 ± 2.59E+08‡	2.61E+01 ± 6.77E−01
F_{ack}	3.60E−02 ± 1.82E−02‡	8.43E−14 ± 3.97E−14 (100%)	F_{ack}	1.36E+01 ± 1.48E+00‡	1.68E−03 ± 3.48E−04
F_{grw}	5.00E−02 ± 6.40E−02‡	1.97E−04 ± 1.39E−03 (98%)	F_{grw}	3.57E+01 ± 1.26E+01‡	1.17E−03 ± 4.95E−03
F_{ras}	5.91E+01 ± 2.65E+01‡	7.29E+00 ± 5.32E+00	F_{ras}	2.63E+02 ± 2.79E+01‡	9.46E+01 ± 9.62E+00
F_{sch}	7.68E+02 ± 8.94E+02‡	0.00E+00 ± 0.00E+00 (100%)	F_{sch}	6.56E+03 ± 4.25E+02‡	1.64E−03 ± 1.82E−03
F_{sal}	8.72E−01 ± 1.59E−01‡	1.85E−01 ± 3.91E−02	F_{sal}	5.97E+00 ± 6.54E−01‡	2.08E−01 ± 2.18E−02
F_{wht}	8.65E+02 ± 1.96E+02‡	3.59E+02 ± 7.23E+00	F_{wht}	1.29E+14 ± 1.60E+14‡	5.81E+02 ± 3.48E+01
F_{pn1}	2.95E−04 ± 1.82E−04‡	1.85E−26 ± 4.12E−26 (100%)	F_{pn1}	6.94E+04 ± 1.58E+05‡	1.14E−06 ± 5.44E−07 (46%)
F_{pn2}	9.03E−03 ± 2.03E−02‡	8.13E−26 ± 1.04E−25 (100%)	F_{pn2}	6.60E+05 ± 7.66E+05‡	1.07E−05 ± 4.71E−06
F_1	1.69E−02 ± 1.80E−02‡	1.44E−25 ± 1.50E−25 (100%)	F_1	5.68E+03 ± 2.63E+03‡	2.29E−05 ± 8.67E−06
F_2	8.38E+02 ± 7.20E+02‡	1.93E+00 ± 1.70E+00	F_2	5.79E+04 ± 1.53E+04‡	1.34E+03 ± 3.86E+02
F_3	5.86E+07 ± 2.61E+07‡	2.00E+06 ± 7.33E+05	F_3	8.82E+08 ± 2.61E+08‡	1.74E+07 ± 5.37E+06
F_4	3.65E+03 ± 2.03E+03‡	5.19E+01 ± 4.10E+01	F_4	9.45E+04 ± 2.77E+04‡	3.52E+03 ± 1.03E+03
F_5	3.20E+03 ± 1.31E+03‡	8.64E−02 ± 1.25E−01	F_5	2.33E+04 ± 4.03E+03‡	7.04E+01 ± 2.09E+01
F_6	5.64E+02 ± 7.58E+02‡	4.51E−01 ± 1.21E+00 (48%)	F_6	7.27E+08 ± 5.08E+08‡	2.67E+01 ± 1.52E+00
F_7	9.54E−01 ± 9.75E−02‡	4.28E−03 ± 6.38E−03 (90%)	F_7	5.73E+02 ± 1.85E+02‡	2.04E−01 ± 8.82E−02
F_8	2.09E+01 ± 5.94E−02≈	2.09E+01 ± 4.64E−02	F_8	2.09E+01 ± 3.84E−02≈	2.09E+01 ± 5.89E−02
F_9	5.23E+01 ± 2.36E+01‡	1.32E+01 ± 5.07E+00	F_9	2.73E+02 ± 1.53E+01‡	9.48E+01 ± 9.00E+00
F_{10}	2.24E+02 ± 1.85E+01‡	1.58E+02 ± 4.54E+01	F_{10}	3.31E+02 ± 3.53E+01‡	1.86E+02 ± 1.15E+01
F_{11}	3.89E+01 ± 3.09E+00≈	3.93E+01 ± 1.16E+00	F_{11}	3.94E+01 ± 8.83E−01≈	3.95E+01 ± 1.20E+00
F_{12}	8.51E+03 ± 7.00E+03‡	2.56E+03 ± 3.66E+03	F_{12}	4.86E+05 ± 8.92E+04‡	3.68E+04 ± 4.13E+04
F_{13}	1.16E+01 ± 3.95E+00‡	8.02E+00 ± 1.55E+00	F_{13}	2.33E+01 ± 2.14E+00‡	1.21E+01 ± 8.44E−01
F_{14}	1.34E+01 ± 1.40E−01≈	1.34E+01 ± 1.48E−01	F_{14}	1.35E+01 ± 1.48E−01≈	1.35E+01 ± 1.22E−01
$NP = 200$			$NP = 300$		
F_{sph}	4.01E+04 ± 6.26E+03‡	5.87E+01 ± 1.17E+01	F_{sph}	1.96E+04 ± 2.00E+03‡	1.10E+03 ± 2.68E+02
F_{ros}	1.53E+10 ± 4.32E+09‡	1.26E+05 ± 4.40E+04	F_{ros}	3.97E+09 ± 8.92E+08‡	1.97E+07 ± 6.12 + 06
F_{ack}	2.02E+01 ± 2.20E−01‡	3.48E+00 ± 2.43E−01	F_{ack}	1.79E+01 ± 3.51E−01‡	8.52E+00 ± 4.54E−01
F_{grw}	3.73E+02 ± 6.03E+01‡	1.55E+00 ± 1.07E−01	F_{grw}	1.79E+02 ± 1.60E+01‡	1.11E+01 ± 1.76E+00
F_{ras}	3.63E+02 ± 2.12E+01‡	1.69E+02 ± 1.19E+01	F_{ras}	2.71E+02 ± 1.27E+01‡	2.00E+02 ± 1.50E+01
F_{sch}	6.88E+03 ± 2.55E+02‡	1.49E+03 ± 2.56E+02	F_{sch}	6.87E+03 ± 2.72E+02‡	4.18E+03 ± 3.77E+02
F_{sal}	1.34E+01 ± 8.41E−01‡	1.79E+00 ± 1.77E−01	F_{sal}	1.52E+01 ± 5.43E−01‡	4.41E+00 ± 3.00E−01
F_{wht}	2.29E+16 ± 1.16E+16‡	1.54E+09 ± 1.41E+09	F_{wht}	2.96E+16 ± 1.09E+16‡	5.03E+12 ± 2.62E−12
F_{pn1}	2.44E+07 ± 7.58E+06‡	2.99E+00 ± 7.85E−01	F_{pn1}	3.71E+07 ± 1.29E+07‡	3.17E+01 ± 1.86E+01
F_{pn2}	8.19E+07 ± 1.99E+07‡	1.15E+01 ± 2.36E+00	F_{pn2}	1.03E+08 ± 1.87E+07‡	4.83E+04 ± 3.47E+04
F_1	5.51E+04 ± 6.74E+03‡	2.87E+01 ± 5.34E+00	F_1	5.18E+03 ± 7.23E+02‡	4.78E+02 ± 1.03E+02
F_2	1.16E+05 ± 1.60E+04‡	1.16E+04 ± 1.93E+03	F_2	2.88E+04 ± 3.54E+03‡	1.83E+04 ± 3.16E+03
F_3	1.19E+09 ± 1.63E+08‡	5.88E+07 ± 1.20E+07	F_3	1.56E+08 ± 3.07E+07‡	9.17E+07 ± 1.80E+07
F_4	1.43E+05 ± 2.63E+04‡	1.85E+04 ± 2.75E+03	F_4	3.49E+04 ± 4.96E+03‡	2.59E+04 ± 3.88E+03
F_5	3.29E+04 ± 2.71E+03‡	2.38E+03 ± 2.62E+02	F_5	1.10E+04 ± 5.32E+02‡	5.17E+03 ± 3.35E+02
F_6	2.61E+10 ± 9.11E+09‡	5.21E+04 ± 1.90E+04	F_6	1.86E+08 ± 4.07E+07‡	3.64E+06 ± 1.38E+06
F_7	3.46E+03 ± 4.31E+02‡	3.30E+01 ± 7.12E+00	F_7	4.01E+03 ± 5.13E+02‡	3.50E+02 ± 5.33E+01
F_8	2.09E+01 ± 6.07E−02≈	2.09E+01 ± 4.77E−02	F_8	2.10E+01 ± 4.44E−02‡	2.09E+01 ± 5.44E−02
F_9	4.13E+02 ± 2.46E+01‡	1.56E+02 ± 1.33E+01	F_9	2.22E+02 ± 1.03E+01‡	1.84E+02 ± 1.21E+01
F_{10}	6.00E+02 ± 5.28E+01‡	2.08E+02 ± 1.30E+01	F_{10}	2.75E+02 ± 1.30E+01‡	2.28E+02 ± 1.53E+01
F_{11}	3.93E+01 ± 1.03E+00≈	3.95E+01 ± 1.09E+00	F_{11}	3.93E+01 ± 1.11E+00≈	3.92E+01 ± 1.39E+00
F_{12}	6.19E+05 ± 7.97E+04‡	4.00E+05 ± 4.47E+04	F_{12}	6.62E+05 ± 7.93E+04‡	5.12E+05 ± 6.57E+04
F_{13}	4.00E+01 ± 3.59E+00‡	1.58E+01 ± 1.17E+00	F_{13}	4.36E+01 ± 4.44E+00‡	1.93E+01 ± 1.24E+00
F_{14}	1.35E+01 ± 1.38E−01≈	1.35E+01 ± 1.45E−01	F_{14}	1.34E+01 ± 1.27E−01≈	1.34E+01 ± 1.65E−01

"‡" indicates the *t* value is significant at a 0.05 level of significance by two-tailed *t*-test. "‡" and "≈" and denote the performance of *DE/rand/1/bin* is worse than and similar to that of OXDE, respectively. The results of the first 20 test instances of *DE/rand/1/bin* are directly taken from [31].

decrease with the increase of the population size; thus, DE cannot find the high-quality solutions, and incomplete convergence may occur frequently under this condition. Actually, based on the analysis in [43], there is no agreement among researchers on the setting of the population size so far. Generally speaking, the setting of the population size is related to the number of variables. That is, if the number of variables is large, the population size should be large, and vice versa.

Some sample graphs for the performance comparison between the two contestant algorithms are given in Fig. 6.

6.4. Effect of the number of variables

In order to investigate the effect of the number of variables (i.e., D) on the performance of *DE/rand/1/bin* and OXDE, experiments have been carried out on the first ten test instances with $D = 10$, 50, 100, and 200, and the other test instances with $D = 10$ and 50.

When $D = 50$, 100, and 200, the population size (NP) is set to D as in Section 6.1. In the case of $D = 10$, the population size (NP) is set to 30 since $NP = 10$ is too small for a DE variant. FES_{max} is set to 10,000 $\times$ D. The experimental results are summarized in Table 4.

Table 4 shows that, in the case of $D = 10$, OXDE outperforms *DE/rand/1/bin* in 19 out of 24 instances. *DE/rand/1/bin* surpasses OXDE only in one instance (i.e., F_3). In the case of $D = 50$, OXDE beats DE in all instances except for F_8 and F_{11}. For F_8 and F_{11}, no significant performance difference is observed. When $D = 100$ and 200, OXDE is significantly better than *DE/rand/1/bin* in all instances. These observations demonstrate that the advantage of OXDE over *DE/rand/1/bin* increases as the number of variables increases.

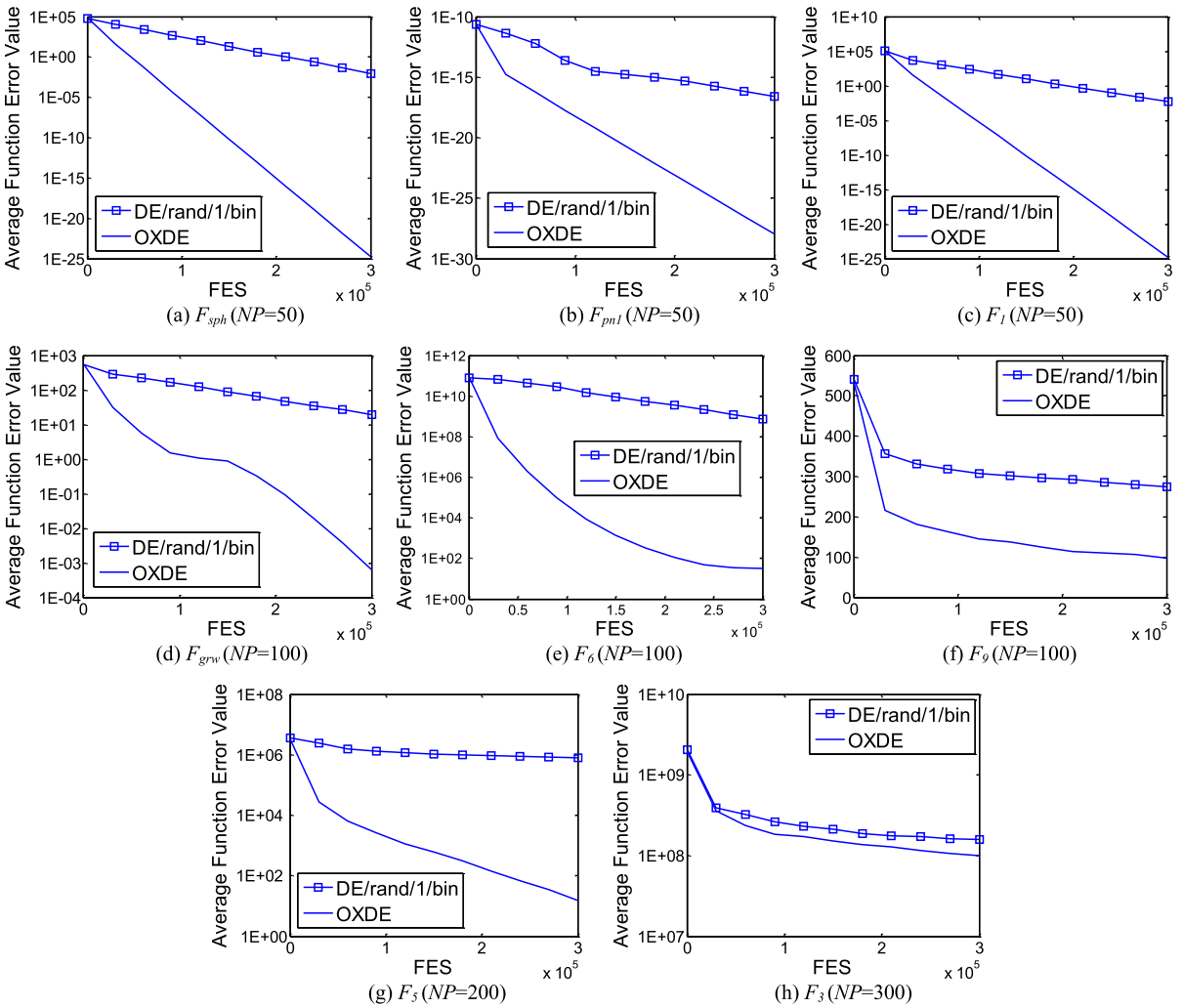

Fig. 6. Convergence graphs of eight representative test instances for *DE/rand/1/bin* and OXDE at $D = 30$ with varying population size.

Table 4
Experimental results of *DE/rand/1/bin* and OXDE over 50 independent runs for test instances with varying problem dimensionality. "Mean error" and "std dev" indicate the average and standard deviation of the function error values, respectively. Percentages in parentheses denote the success rates. In the fields without parentheses, the success rates are zero. *t*-test is performed between *DE/rand/1/bin* and OXDE.

Inst.	*DE/rand/1/bin* Mean error ± std dev	OXDE Mean error ± std dev	Inst.	*DE/rand/1/bin* Mean error ± std dev	OXDE Mean error ± std dev
D = 10			*D* = 50		
F_{sph}	3.26E−28 ± 5.83E−28‡ (100%)	7.59E−56 ± 1.29E−55 (100%)	F_{sph}	5.91E−02 ± 9.75E−02‡	1.13E−27 ± 1.56E−27 (100%)
F_{ros}	4.78E−01 ± 1.32E+00‡ (86%)	8.79E−27 ± 1.62E−26 (100%)	F_{ros}	1.13E+10 ± 2.34E+10‡	1.60E+01 ± 1.62E+01
F_{ack}	8.35E−15 ± 8.52E−15‡ (100%)	2.66E−15 ± 0.00E+00 (100%)	F_{ack}	2.39E−02 ± 8.90E−03‡	1.21E−14 ± 6.57E−15 (100%)
F_{grw}	5.75E−02 ± 3.35E−02‡	1.25E−02 ± 2.24E−02 (42%)	F_{grw}	7.55E−02 ± 1.14E−01‡	1.47E−04 ± 1.04E−03 (98%)
F_{ras}	1.85E+00 ± 1.68E+00‡ (26%)	1.59E−01 ± 5.45E−01 (90%)	F_{ras}	6.68E+01 ± 2.36E+01‡	1.73E+01 ± 4.04E+00
F_{sch}	1.42E+01 ± 3.93E+01‡	0.00E+00 ± 0.00E+00 (100%)	F_{sch}	1.07E+03 ± 5.15E+02‡	1.82E−11 ± 0.00E+00 (100%)
F_{sal}	1.07E−01 ± 2.77E−02≈	9.99E−02 ± 7.10E−09	F_{sal}	1.15E+00 ± 1.49E−01‡	2.59E−01 ± 4.92E−02
F_{wht}	1.81E+01 ± 1.59E+01≈	2.18E+01 ± 1.67E+01 (2%)	F_{wht}	1.43E+05 ± 4.10E+05‡	1.03E+03 ± 4.61E+01
F_{pn1}	3.85E−29 ± 7.28E−29‡ (100%)	4.71E−32 ± 1.12E−47 (100%)	F_{pn1}	3.07E−02 ± 7.93E−02‡	6.22E−03 ± 3.60E−02 (96%)
F_{pn2}	1.49E−28 ± 2.20E−28‡ (100%)	1.35E−32 ± 5.59E−48 (100%)	F_{pn2}	2.24E−01 ± 3.35E−01‡	3.21E−26 ± 2.31E−25 (100%)
F_1	0.00E+00 ± 0.00E+00≈ (100%)	0.00E+00 ± 0.00E+00 (100%)	F_1	1.50E−02 ± 1.09E−02‡	4.19E−27 ± 2.53E−27 (100%)
F_2	2.27E−15 ± 1.14E−14‡ (100%)	4.89E−27 ± 7.37E−27 (100%)	F_2	2.89E+04 ± 1.03E+04‡	3.06E+02 ± 1.15E+02
F_3	8.76E−06 ± 2.78E−05† (76%)	2.18E−04 ± 6.21E−04 (4%)	F_3	5.40E+08 ± 2.62E+08‡	2.36E+06 ± 9.04E+05
F_4	8.87E−14 ± 1.24E−13‡ (100%)	2.36E−24 ± 3.94E−24 (100%)	F_4	6.04E+04 ± 1.74E+04‡	4.79E+03 ± 2.00E+03
F_5	1.07E−03 ± 2.40E−03‡	0.00E+00 ± 0.00E+00 (100%)	F_5	5.81E+03 ± 1.12E+03‡	4.51E+02 ± 3.27E+02
F_6	3.19E−01 ± 1.10E+00‡ (92%)	4.13E−26 ± 2.23E−25 (100%)	F_6	1.29E+03 ± 1.98E+03‡	2.11E+01 ± 2.27E+01
F_7	1.56E−01 ± 1.63E−01‡	7.01E−02 ± 6.78E−02 (6%)	F_7	1.05E+00 ± 6.24E−02‡	3.74E−03 ± 7.40E−03 (88%)
F_8	2.04E+01 ± 1.08E−01≈	2.04E+01 ± 7.76E−02	F_8	2.11E+01 ± 2.93E−02≈	2.11E+01 ± 3.54E−02
F_9	2.01E+00 ± 1.41E+00‡ (10%)	1.19E−01 ± 3.83E−01 (90%)	F_9	7.65E+01 ± 2.30E+01‡	3.42E+01 ± 9.07E+00
F_{10}	1.26E+01 ± 7.26E+00‡	7.40E+00 ± 4.98E+00	F_{10}	4.24E+02 ± 2.98E+01‡	3.38E+02 ± 2.16E+01
F_{11}	2.00E+00 ± 2.90E+00‡ (34%)	1.13E+00 ± 1.84E+00 (46%)	F_{11}	7.26E+01 ± 1.17E+00≈	7.29E+01 ± 1.20E+00
F_{12}	4.60E+01 ± 1.71E+02‡ (74%)	1.66E+01 ± 1.01E+02 (78%)	F_{12}	3.90E+04 ± 1.84E+04‡	1.57E+04 ± 1.38E+04
F_{13}	9.34E−01 ± 3.72E−01‡	6.39E−01 ± 3.43E−01	F_{13}	2.02E+01 ± 6.93E+00‡	1.29E+01 ± 3.68E+00
F_{14}	3.47E+00 ± 4.83E−01‡	3.11E+00 ± 5.22E−01	F_{14}	2.32E+01 ± 1.41E−01‡	2.31E+01 ± 1.46E−01
D = 100			*D* = 200		
F_{sph}	4.28E+03 ± 1.27E+03‡	6.70E−08 ± 2.79E−08 (100%)	F_{sph}	1.26E+05 ± 1.06E+04‡	2.44E+00 ± 5.31E−01
F_{ros}	3.33E+08 ± 1.67E+08‡	9.83E+01 ± 1.66E+01	F_{ros}	2.97E+10 ± 3.81E+09‡	4.30E+03 ± 1.20E+03
F_{ack}	8.81E+00 ± 8.07E−01‡	5.76E−05 ± 2.77E−05	F_{ack}	1.81E+01 ± 2.26E−01‡	6.83E−01 ± 1.85E−01
F_{grw}	3.94E+01 ± 8.01E+00‡	4.94E−08 ± 3.11E−08 (100%)	F_{grw}	1.15E+03 ± 9.22E+01‡	5.46E−01 ± 7.93E−02
F_{ras}	8.30E+02 ± 6.51E+01‡	1.17E+02 ± 6.81E+01	F_{ras}	2.37E+03 ± 7.24E+01‡	1.03E+03 ± 7.94E+01
F_{sch}	2.54E+04 ± 2.15E+03‡	2.60E+01 ± 7.12E+01 (88%)	F_{sch}	6.66E+04 ± 1.32E+03‡	2.15E+01 ± 5.93E+01
F_{sal}	1.02E+01 ± 7.91E−01‡	4.49E−01 ± 4.75E−02	F_{sal}	3.69E+01 ± 1.80E+00‡	1.94E+00 ± 1.33E−01
F_{wht}	5.44E+15 ± 5.07E+15‡	7.07E+03 ± 6.36E+02	F_{wht}	3.13E+18 ± 9.48E+17‡	8.56E+07 ± 8.87E+07
F_{pn1}	6.20E+05 ± 7.38E+05‡	1.30E−02 ± 4.45E−02 (86%)	F_{pn1}	3.49E+08 ± 7.60E+07‡	9.64E−02 ± 7.40E−02
F_{pn2}	4.34E+06 ± 2.30E+06‡	2.25E−04 ± 1.55E−03 (68%)	F_{pn2}	8.08E+08 ± 1.86E+08‡	4.41E+00 ± 2.24E+00

"†" and "‡" indicate the *t* value is significant at a 0.05 level of significance by two-tailed *t*-test. "†", "‡" and "≈" denote the performance of *DE/rand/1/bin* is better than, worse than, and similar to that of OXDE, respectively. The results of the first 20 test instances of *DE/rand/1/bin* are directly taken from [31].

From Table 4, we can see that the performance of OXDE and *DE/rand/1/bin* degrades as the number of variables increases. It is not surprising since the search space will rapidly enlarge with the increase of the number of variables. Only OXDE can have successful runs when $D = 100$ and no algorithms can produce a successful run in any instances when $D = 200$.

Fig. 7 shows the average best fitness curves of the two compared methods over 50 independent runs for the selected test instances.

6.5. Runtime complexity of OXDE

For one pair of mutant and target vectors, the implementation of QOX contains three main steps: (1) divide the decision vector $(x_1,\ldots,x_D)$ into K factors (i.e., K subvectors); (2) determine the levels for each factor; and (3) use orthogonal array $L_M(Q^K)$ to generate the offspring. Based on the introduction and the example in Section 4, we can infer that QOX used in this paper is an inexpensive operator. Moreover, in our framework only one pair of mutant and target vectors is used to take part in QOX at each generation. Consequently, our framework does not impose any serious burden on the runtime complexity of the existing DE variants.

In order to further verify the above analysis, for OXDE and *DE/rand/1/bin*, 50 independent runs are conducted over 24 test instances, and the maximum number of *FES* is set to $10{,}000 \times D$. The settings of F and CR in OXDE and *DE/rand/1/bin* are the same as in Section 6.1. We record the mean and standard deviation of the runtime of OXDE and *DE/rand/1/bin* for all the 24 test instances in one run according to the following three cases: (1) $D = 30$ and $NP = D$; (2) $D = 30$ and $NP = 100$; and (3) $D = 50$ and $NP = D$.

Table 5 summarizes the experimental results of OXDE and *DE/rand/1/bin* using $10{,}000 \times D$ *FES* as the termination criterion. From Table 5, the runtime of OXDE is slightly higher than that of *DE/rand/1/bin* when $D = 30$ and $NP = D$. Specifically, in

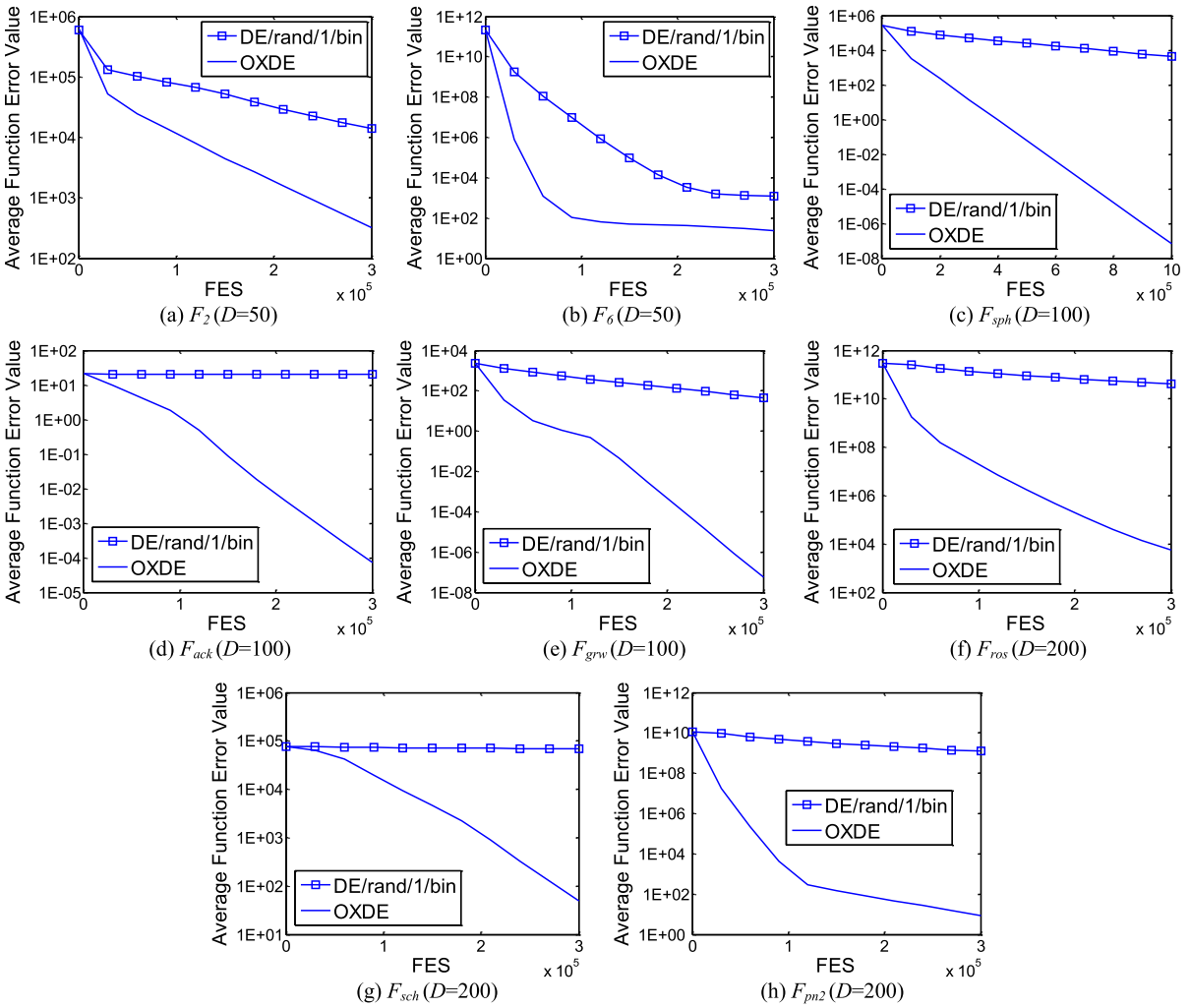

Fig. 7. Convergence graphs of eight representative test instances for *DE/rand/1/bin* and OXDE with varying dimensionality.

Table 5
Comparison of the runtime (in seconds) of *DE/rand/1/exp* and OXDE with 10,000 × *D FES*.

	DE/rand/1/bin Mean time ± std dev	OXDE Mean time ± std dev
$D = 30$; $NP = D$	627.80 ± 5.43	648.74 ± 4.28
$D = 30$; $NP = 100$	1001.25 ± 16.26	686.90 ± 2.34
$D = 50$; $NP = D$	2371.15 ± 72.61	1912.18 ± 134.28

this case, OXDE is on average 3.34% slower than *DE/rand/1/bin* due to the use of QOX. It is interesting to note that for the other two cases, OXDE is significantly faster than *DE/rand/1/bin*. This is because the function error values provided by OXDE decrease more rapidly than *DE/rand/1/bin* during the evolution. For example, when $D = 30$ and $NP = 100$, the final function error values achieved by *DE/rand/1/bin* and OXDE are 4.03E+08 and 2.61E+01 for F_{ros}, respectively; and when $D = 50$ and $NP = D$, the final function error values achieved by *DE/rand/1/bin* and OXDE are 1.13E+10 and 1.60E+01 for F_{ros}, respectively. As a result, more time is needed to compute the function error values, and the storage requirement increases for *DE/rand/1/bin* when solving some complex test instances.

As pointed out by Das et al. [7], an unfair advantage may be provided to algorithms that use lower computational overheads if we only use the maximum number of *FES* as the stopping criterion. According to the suggestion in [7], we select the target error accuracy level specified in Section 6.1 as the stopping criterion to compare the runtime of *DE/rand/1/bin* and OXDE. It is necessary to emphasize that in an unsuccessful run, when the maximum number of *FES* is reached, the procedure

Table 6
Comparison of the runtime (in seconds) of *DE*/*rand*/1/*exp* and OXDE with the target error accuracy level.

	DE/*rand*/1/*bin* Mean time ± std dev	OXDE Mean time ± std dev
$D = 30$; $NP = D$	595.69 ± 11.95	568.85 ± 5.02
$D = 30$; $NP = 100$	1001.25 ± 16.26	685.55 ± 3.56
$D = 50$; $NP = D$	2371.15 ± 72.61	1717.32 ± 21.48

will halt. The experimental results have been summarized in Table 6. As shown in Table 6, OXDE needs less runtime to reach the target error accuracy level, compared with *DE*/*rand*/1/*bin* when $D = 30$ and $NP = D$. This is due to the fact that QOX can greatly enhance the search ability of *DE*/*rand*/1/*bin*, and OXDE is able to achieve the successful run with a higher probability. Similar to Table 5, the runtime of OXDE is significantly less than that of *DE*/*rand*/1/*bin* for the other two cases. Since in the case of $D = 30$ and $NP = 100$ and in the case of $D = 50$ and $NP = D$, *DE*/*rand*/1/*bin* fails to solve all the test instances, the runtime is identical in Tables 5 and 6.

6.6. Can our framework improve other DE variants?

As pointed out previously, we attempt to present a framework to improve the search ability of DE by QOX and propose an illustrative algorithm OXDE. We have demonstrated that our framework can improve the search ability of a classic DE, that is, *DE*/*rand*/1/*bin*. A question which naturally arises is: can our framework improve other DE variants? To answer this question, we have applied the QOX operator based on $L_9(3^4)$ to three other classic DE variants: *DE*/*rand*/1/*exp*, *DE*/*rand*/2/*exp*, and *DE*/*rand*/2/*bin*, and four recent DE variants: jDE [6], SaDE [32], JADE [52], DEahcSPX [31], in the same way as in Section 5.

We have compared these algorithms with their QOX augmented algorithms. Our experimental results show that under our framework, QOX can greatly improve DEahcSPX, *DE*/*rand*/1/*exp*, *DE*/*rand*/2/*exp*, and *DE*/*rand*/2/*bin*, and it can also improve jDE, SaDE, and JADE to some extent. Due to space limitations, in this paper, we only report in Table 7 the comparison results between DEahcSPX and its QOX augmented version, OX-DEahcSPX, in the first ten instances with $D = 100$ and 200, and in Table 8 the comparison results between *DE*/*rand*/1/*exp*, *DE*/*rand*/2/*exp*, and *DE*/*rand*/2/*bin* and their respective QOX augmented versions in 24 test instances with $D = 30$. For DEahcSPX and OX-DEahcSPX, the settings of *NP*, *F*, and *CR* are the same as in [31], and for *DE*/*rand*/1/*exp*, *DE*/*rand*/2/*exp*, and *DE*/*rand*/2/*bin* and their QOX augmented versions, the settings of *NP*, *F*, and *CR* are the same as in Section 6.1.

Table 7 shows that in all the test instances, OX-DEahcSPX significantly outperforms DEahcSPX in terms of the solution quality. In the case of $D = 100$, DEahcSPX fails in all the runs for each instance to reach the target error accuracy level while the success rates in OX-DEahcSPX are not below 90% in five test instances.

Table 8 shows that, OX-DE/rand/1/exp, OX-DE/rand/2/exp, and OX-DE/rand/2/bin outperform their original versions in 13, 16, and 21 out of 24 test instances, respectively, in terms of the solution quality. Their performance is about the same as their original versions in the rest of the test instances except that OX-DE/rand/1/exp is surpassed by its original version in F_{wht} and F_{13}, and OX-DE/rand/2/exp is surpassed by its original version in F_{wht}, F_{12}, and F_{13}. In terms of the success rate, OX-DE/rand/1/exp, OX-DE/rand/2/exp, and OX-DE/rand/2/bin are better than or at least comparable to their original versions in all the test instances.

Table 7
Experimental results of DEahcSPX and OX-DEahcSPX over 50 independent runs for test instances with 100 and 200 variables, after 1,000,000 and 2,000,000 *FES*. "Mean error" and "std dev" indicate the average and standard deviation of the function error values, respectively. Percentages in parentheses denote the success rates. In the fields without parentheses, the success rates are zero. *t*-test is performed between DEahcSPX and OX-DEahcSPX.

Inst.	$D = 100$		$D = 200$	
	DEahcSPX Mean error ± std dev	OX-DEahcSPX Mean error ± std dev	DEahcSPX Mean error ± std dev	OX-DEahcSPX Mean error ± std dev
F_{sph}	5.01E+01 ± 8.94E+01‡	2.54E−09 ± 1.16E−09 (100%)	7.01E+03 ± 1.07E+03‡	6.81E−01 ± 1.69E−01
F_{ros}	1.45E+05 ± 1.11E+05‡	9.75E+01 ± 2.33E+01	1.11E+08 ± 2.63E+07‡	1.46E+03 ± 2.43E+02
F_{ack}	1.91E+00 ± 3.44E−01‡	1.33E−05 ± 1.16E−05	8.45E+00 ± 4.13E−01‡	6.22E−01 ± 2.94E−01
F_{grw}	1.23E+00 ± 2.14E−01‡	1.54E−09 ± 5.83E−10 (100%)	6.08E+01 ± 9.30E+00‡	1.90E−01 ± 3.97E−02
F_{ras}	4.75E+02 ± 6.55E+01‡	1.03E+02 ± 7.01E+01	1.53E+03 ± 8.31E+01‡	9.59E+02 ± 1.54E+02
F_{sch}	2.48E+04 ± 2.17E+03‡	2.17E+01 ± 6.58E+01 (90%)	6.61E+04 ± 1.44E+03‡	1.02E+01 ± 4.29E+01
F_{sal}	3.11E+00 ± 5.79E−01‡	4.35E−01 ± 5.25E−02	1.10E+01 ± 4.38E−01‡	1.72E+00 ± 1.11E−01
F_{wht}	4.06E+10 ± 6.57E+10‡	6.00E+03 ± 4.21E+02	4.21E+13 ± 1.74E+13‡	1.43E+06 ± 9.96E+05
F_{pn1}	4.34E+00 ± 1.75E+00‡	4.35E−03 ± 1.88E−02 (92%)	2.27E+01 ± 5.73E+00‡	5.15E−02 ± 4.81E−02
F_{pn2}	7.25E+01 ± 2.44E+01‡	1.76E−08 ± 6.25E−08 (100%)	6.24E+04 ± 4.77E+04‡	1.45E+00 ± 1.49E+00

"‡" indicates the *t* value is significant at a 0.05 level of significance by two-tailed *t*-test. "‡ and ≈" denote the performance of DEahcSPX is worse than and similar to that of OX-DEahcSPX, respectively. The results of DEahcSPX are directly taken from [31].

Table 8
Experimental results of *DE*/*rand*/1/*exp*, *DE*/*rand*/2/*exp*, *DE*/*rand*/2/*bin*, OX-DE/rand/1/exp, OX-DE/rand/2/exp, and OX-DE/rand/2/bin over 50 independent runs for test instances with 30 variables, after 300,000 *FES*. "Mean error" and "std dev" indicate the average and standard deviation of the function error values, respectively. Percentages in parentheses denote the success rates. In the fields without parentheses, the success rates are zero. *t*-test is performed between the original algorithms and the augmented algorithms.

Inst.	*DE*/*rand*/1/*exp* Mean error ± std dev	OX-DE/rand/1/exp Mean error ± std dev	*DE*/*rand*/2/*exp* Mean error ± std dev	OX-DE/rand/2/exp Mean error ± std dev	*DE*/*rand*/2/*bin* Mean error ± std dev	OX-DE/rand/2/bin Mean error ± std dev
F_{sph}	8.06E−44 ± 7.20E−44‡ (100%)	2.20E−88 ± 2.50E−88 (100%)	1.00E−19 ± 6.03E−20‡ (100%)	2.24E−47 ± 2.25E−47 (100%)	2.01E+04 ± 3.42E+03‡	8.42E−25 ± 8.68E−25 (100%)
F_{ros}	1.99E−14 ± 2.19E−14‡ (100%)	0.00E+00 ± 0.00E+00 (100%)	3.61E+00 ± 1.72E+00‡	4.08E−18 ± 1.05E−17 (100%)	3.92E+09 ± 1.42E+09‡	6.66E−01 ± 1.50E+00 (8%)
F_{ack}	6.07E−15 ± 7.03E−16‡ (100%)	2.66E−15 ± 0.00E+00 (100%)	9.44E−11 ± 2.88E−11‡ (100%)	2.66E−15 ± 0.00E+00 (100%)	1.80E+01 ± 4.66E−01‡	2.65E−13 ± 1.47E−13 (100%)
F_{grw}	4.43E−04 ± 1.77E−03‡ (94%)	0.00E+00 ± 0.00E+00 (100%)	2.14E−13 ± 8.02E−13‡ (100%)	0.00E+00 ± 0.00E+00 (100%)	1.67E+02 ± 2.84E+01‡	4.43E−04 ± 2.31E−03 (96%)
F_{ras}	0.00E+00 ± 0.00E+00$^{\approx}$ (100%)	0.00E+00 ± 0.00E+00 (100%)	5.68E−15 ± 1.72E−14‡ (100%)	0.00E+00 ± 0.00E+00 (100%)	2.84E+02 ± 1.11E+01‡	5.79E+01 ± 1.38E+01
F_{sch}	0.00E+00 ± 0.00E+00$^{\approx}$ (100%)	0.00E+00 ± 0.00E+00 (100%)	0.00E+00 ± 0.00E+00$^{\approx}$ (100%)	0.00E+00 ± 0.00E+00 (100%)	7.15E+03 ± 2.78E+02‡	0.00E+00 ± 0.00E+00 (100%)
F_{sal}	3.29E−01 ± 4.62E−02‡	1.87E−01 ± 3.85E−02	5.52E−01 ± 5.37E−02‡	1.78E−01 ± 4.14E−02	1.50E+01 ± 1.18E+00‡	2.11E−01 ± 3.77E−02
F_{wht}	7.03E+01 ± 3.07E+01†	2.84E+02 ± 8.22E+01	1.73E+02 ± 1.87E+01†	1.95E+02 ± 1.90E+01	3.78E+16 ± 1.50E+16‡	3.49E+02 ± 3.52E+01
F_{pn1}	1.57E−32 ± 5.52E−48$^{\approx}$ (100%)	1.57E−32 ± 5.52E−48 (100%)	3.28E−21 ± 2.35E−21‡ (100%)	1.57E−32 ± 5.52E−48 (100%)	4.12E+07 ± 1.69E+07‡	1.03E−02 ± 6.01E−02 (96%)
F_{pn2}	1.35E−32 ± 1.10E−47$^{\approx}$ (100%)	1.35E−32 ± 1.10E−47 (100%)	1.73E−20 ± 1.03E−20‡ (100%)	1.35E−32 ± 1.10E−47 (100%)	1.21E+08 ± 4.11E+07‡	4.66E−24 ± 1.41E−23 (100%)
F_1	0.00E+00 ± 0.00E+00$^{\approx}$ (100%)	0.00E+00 ± 0.00E+00 (100%)	3.45E−20 ± 1.97E−20‡ (100%)	0.00E+00 ± 0.00E+00 (100%)	5.76E+03 ± 1.02E+03‡	1.90E−24 ± 2.28E−24 (100%)
F_2	1.41E−02 ± 5.89E−03‡	6.74E−11 ± 7.12E−11 (100%)	4.54E+01 ± 1.03E+01‡	5.83E−03 ± 3.43E−03	2.91E+04 ± 5.01E+03‡	2.84E+01 ± 1.77E+01
F_3	2.19E+06 ± 1.23E+06‡	1.09E+06 ± 5.75E+05	2.78E+07 ± 7.51E+06‡	4.25E+06 ± 1.82E+06	1.89E+08 ± 3.78E+07‡	3.91E+06 ± 1.47E+06
F_4	8.24E+01 ± 2.87E+01‡	1.13E−04 ± 9.11E−05	1.56E+03 ± 3.75E+02‡	7.69E+00 ± 5.19E+00	3.82E+04 ± 5.13E+03‡	6.55E+02 ± 5.18E+02
F_5	3.75E+03 ± 9.34E+02‡	5.30E+02 ± 3.27E+02	5.93E+03 ± 7.21E+02‡	6.89E+02 ± 3.98E+02	1.12E+04 ± 8.82E+02‡	1.59E+01 ± 4.06E+01
F_6	4.55E−14 ± 2.41E−13‡ (100%)	6.87E−27 ± 1.88E−26 (100%)	2.53E+00 ± 1.51E+00‡	3.43E−17 ± 1.87E−16 (100%)	3.23E+08 ± 1.22E+08‡	1.17E+01 ± 1.89E+00 (58%)
F_7	8.36E−02 ± 6.60E−02‡ (16%)	1.09E−02 ± 8.30E−03 (68%)	1.03E+00 ± 3.92E−02‡	1.34E−02 ± 1.30E−02 (56%)	4.48E+03 ± 1.05E+03‡	9.70E−03 ± 7.79E−03 (64%)
F_8	2.09E+01 ± 4.82E−02$^{\approx}$	2.09E+01 ± 6.28E−02	2.09E+01 ± 5.28E−02$^{\approx}$	2.09E+01 ± 5.22E−02	2.09E+01 ± 4.51E−02$^{\approx}$	2.09E+01 ± 4.87E−02
F_9	0.00E+00 ± 0.00E+00$^{\approx}$ (100%)	0.00E+00 ± 0.00E+00 (100%)	0.00E+00 ± 0.00E+00$^{\approx}$ (100%)	0.00E+00 ± 0.00E+00 (100%)	2.42E+02 ± 1.17E+01‡	6.76E+01 ± 1.50E+01
F_{10}	1.12E+02 ± 1.45E+01‡	7.70E+01 ± 1.31E+01	1.43E+02 ± 1.52E+01‡	9.60E+01 ± 1.17E+01	2.93E+02 ± 1.65E+01‡	1.80E+02 ± 1.35E+01
F_{11}	2.91E+01 ± 1.76E+00$^{\approx}$	2.93E+01 ± 1.87E+00	2.90E+01 ± 1.81E+00$^{\approx}$	2.93E+01 ± 1.85E+00	3.93E+01 ± 1.12E+00$^{\approx}$	3.95E+01 ± 1.09E+00
F_{12}	3.40E+04 ± 8.40E+03‡	2.74E+04 ± 1.51E+04	3.86E+04 ± 7.68E+03†	4.40E+04 ± 7.94E+03	6.43E+05 ± 1.64E+05‡	2.42E+03 ± 2.49E+03
F_{13}	2.32E+00 ± 1.91E−01†	2.50E+00 ± 1.56E−01	2.84E+00 ± 2.46E−01†	3.04E+00 ± 2.40E−01	5.50E+01 ± 8.53E+00‡	9.11E+00 ± 9.51E−01
F_{14}	1.29E+01 ± 1.79E−01$^{\approx}$	1.29E+01 ± 2.42E−01	1.29E+01 ± 2.32E−01$^{\approx}$	1.29E+01 ± 1.87E−01	1.34E+01 ± 1.57E−01$^{\approx}$	1.34E+01 ± 1.48E−01

"†" and "‡" indicate the *t* value is significant at a 0.05 level of significance by two-tailed *t*-test. "†", "‡" and "≈" denote the performance of the corresponding algorithm is better than, worse than, and similar to that of the augmented algorithm, respectively.

These experimental results imply that our framework could be an effective way to improve the performance of other DE variants.

6.7. Compared with other OX-based DE

In this subsection, we compare the performance of OXDE with that of another OX-based DE proposed by Gong et al. [17]. As pointed out in Section 5, the main difference between the proposed OXDE and the method in [17] is that, for the former, QOX is embedded in DE in a straightforward manner; however, for the latter, QOX is executed independently with DE. To make the comparison fair, the following procedure introduced in [17] is inserted after Step 3 of Fig. 3 after the steps (i.e., Step 3.1.5– Step 3.1.8) for randomly choosing an individual from the population to undergo QOX are deleted from Fig. 3: (1) randomly select two individuals from the population; (2) combine these two individuals to produce M offspring by QOX, and select the best solution from the M offspring denoted as $\vec{B}$; (3) randomly select an individual $\vec{A}$ from the population; and (4) if $\vec{B}$ is better than $\vec{A}$, then replace $\vec{A}$. After these modifications, the algorithm obtained according to [17] is called Orth-DE.

Now, we apply Orth-DE to solve the test suite adopted in this paper at 30 dimensions. All control parameters are kept the same to ensure a fair comparison. Results for Orth-DE are summarized in Table 9, and the results of *DE*/*rand*/1/*bin* and OXDE are repeated in this table for the purpose of comparison. From Table 9, Orth-DE outperforms OXDE in four test instances; however, OXDE performs better than Orth-DE in 15 test instances. As discussed in Section 6.2, *DE*/*rand*/1/*bin* cannot show better performance than OXDE in any test instances. It is necessary to note that *DE*/*rand*/1/*bin* surpasses Orth-DE in nine test instances according to the obtained results, although Orth-DE performs better than *DE*/*rand*/1/*bin* in 12 test instances. Moreover, for test instances F_{ack}, F_{grw}, F_{pn1}, F_{pn2}, and F_7, the success rates provided by Orth-DE clearly decrease compared with *DE*/*rand*/1/*bin* and OXDE. The above phenomenon suggests that under the framework of Orth-DE, adding QOX into DE might have a side effect on the performance of the original DE for some test instances, although Orth-DE is capable of improving the performance of the original DE to a certain degree.

Based on the above discussion, we can conclude that embedding QOX in DE is a more effective manner of enhancing the performance of DE.

6.8. Compared with other state-of-the-art DE

This subsection presents a performance comparison among *DE*/*rand*/1/*bin*, OXDE, and one recent, state-of-the-art DE (denoted as ODE) introduced by Rahnamayan et al. [34] on the test suite adopted in this paper at 30 dimensions. The same parameter settings suggested in [34] have been used in the current experiment to assure a fair comparison. The population size is equal to 100, the scaling factor is $F = 0.5$, and the crossover control parameter is $C_R = 0.9$. In the current experiment, each run is implemented up to 300,000 FES. The mean and standard deviation of the function error values and the success rate of 50 independent runs for each algorithm have been summarized in Table 10.

According to the t-test, OXDE outperforms ODE in 13 test instances, while this number is six for ODE. OXDE yields results comparable to ODE for the rest of the test instances. By carefully looking at the results in Table 10, one can note that *DE*/*rand*/1/*bin* is able to beat ODE in nine test instances (F_{ros}, F_1, F_2, F_3, F_4, F_5, F_6, F_7, and F_8), which means that using opposition-based learning might deteriorate the performance of the original DE for some test instances.

In terms of the success rate, *DE*/*rand*/1/*bin*, ODE, and OXDE achieve a 100% success rate for F_{sph}, F_{ack}, F_{grw}, F_{pn1}, F_{pn2}, and F_1. In addition, for F_{ros} and F_{sch}, only OXDE has the capability to converge to the optima, and the success rate provided by OXDE is 2% and 100%, respectively. The success rate of *DE*/*rand*/1/*bin* and OXDE is 8% and 10% for F_6, respectively; however ODE cannot solve this test instance in any trial. For F_7, *DE*/*rand*/1/*bin* provides a 98% success rate which is slightly and significantly better than that of OXDE and ODE, respectively.

The above comparisons reveal that the performance of the proposed OXDE is statistically better than that of *DE*/*rand*/1/*bin* according to the current parameter settings, and that for the majority of test instances, OXDE performs better than ODE.

6.9. Comparison with other state-of-the-art EAs

We compare the performance of OXDE with that of three other state-of-the-art EAs: GL-25 [15], CMA-ES [19], and CLPSO [25]. For OXDE, GL-25, CMA-ES, and CLPSO, 25 independent runs are executed in 14 test instances F_1–F_{14} at $D = 30$. The maximum number of *FES* is set to 300,000 in all runs. Table 11 summarizes the experimental results. It is necessary to note that the experimental results of GL-25, CMA-ES, and CLPSO are directly taken from [43].

From Table 11, OXDE outperforms GL-25, CMA-ES, and CLPSO in 11, six, and eight out of 14 test instances, respectively. In addition, GL-25, CMA-ES, and CLPSO surpass OXDE in one, six, and four test instances, respectively. Thus, we can conclude that, overall, OXDE is better than GL-25 and CLPSO and is very competitive with CMA-ES.

The above discussion signifies that OXDE is a generally good global function optimizer.

6.10. Orthogonal crossover versus uniformly random sampling and Halton sampling

In this paper, we use QOX to sample nine representative points from the hyper-rectangle defined by the target vector and the mutant vector to enhance the search ability of DE. Actually, one can employ different sampling strategies instead of QOX

Table 9
Experimental results of *DE*/*rand*/1/*exp*, OXDE, and Orth-DE over 50 independent runs at $D = 30$, after 300,000 *FES*. "Mean error" and "std dev" indicate the average and standard deviation of the function error values, respectively. Result in parentheses denotes the success rate. In the fields without parentheses, the success rates are zero. *t*-test is performed between Orth-DE and each of *DE*/*rand*/1/*exp* and OXDE.

Inst.	*DE*/*rand*/1/*bin* Mean error ± std dev	OXDE Mean error ± std dev	Orth-DE Mean error ± std dev	Inst.	*DE*/*rand*/1/*bin* Mean error ± std dev	OXDE Mean error ± std dev	Orth-DE Mean error ± std dev
F_{sph}	5.73E−17 ± 2.03E−16‡ (100%)	5.21E−59 ± 1.82E−58‡ (100%)	4.50E−72 ± 3.12E−71 (100%)	F_3	3.63E+06 ± 2.06E+06‡	5.41E+05 ± 2.86E+05‡	1.48E+05 ± 8.20E+04
F_{ros}	5.20E+01 ± 8.56E+01‡	4.78E−01 ± 1.30E+00† (88%)	1.20E+00 ± 1.85E+00 (70%)	F_4	5.54E+01 ± 6.37E+01‡	2.58E+00 ± 3.91E+00$^{\approx}$	2.93E+00 ± 8.41E+00
F_{ack}	1.37E−09 ± 1.32E−09† (100%)	2.66E−15 ± 0.00E+00† (100%)	5.55E−01 ± 6.89E−01 (56%)	F_5	1.08E+03 ± 5.31E+02‡	5.72E+00 ± 1.18E+01†	2.35E+02 ± 2.44E+02
F_{grw}	2.66E−03 ± 5.73E−03† (76%)	1.82E−03 ± 4.44E−03† (84%)	7.17E−03 ± 1.29E−02 (60%)	F_6	6.67E+01 ± 1.51E+02‡	7.97E−01 ± 1.61E+00$^{\approx}$ (80%)	1.11E+00 ± 1.80E+00 (72%)
F_{ras}	2.55E+01 ± 8.14E+00†	8.99E+00 ± 2.29E+00†	3.63E+01 ± 9.94E+00	F_7	7.59E−03 ± 8.96E−03† (66%)	9.98E−03 ± 9.50E−03† (68%)	1.48E−02 ± 1.14E−02 (52%)
F_{sch}	4.90E+02 ± 2.34E+02‡	0.00E+00 ± 0.00E+00† (100%)	1.77E+02 ± 1.86E+02	F_8	2.09E+01 ± 1.33E−01†	2.09E+01 ± 5.48E−02†	2.10E+01 ± 6.86E−02
F_{sal}	2.52E−01 ± 4.78E−02†	2.01E−01 ± 4.16E−02†	6.27E−01 ± 2.39E−01	F_9	2.43E+01 ± 6.23E+00†	1.51E+01 ± 4.07E+00†	4.36E+01 ± 1.42E+01
F_{wht}	3.10E+02 ± 1.07E+02$^{\approx}$	3.35E+02 ± 7.83E+01$^{\approx}$ (2%)	3.12E+02 ± 1.38E+02 (2%)	F_{10}	7.33E+01 ± 6.62E+01‡	4.70E+01 ± 4.71E+01$^{\approx}$	5.65E+01 ± 2.42E+01
F_{pn1}	4.56E−02 ± 1.31E−01† (86%)	1.03E−02 ± 7.32E−02† (98%)	2.00E−01 ± 5.14E−01 (68%)	F_{11}	3.57E+01 ± 9.45E+00‡	3.36E+01 ± 1.13E+01‡	2.27E+01 ± 8.64E+00
F_{pn2}	1.44E−01 ± 7.19E−01‡ (96%)	2.25E−32 ± 6.37E−32† (100%)	3.33E−03 ± 9.23E−03 (82%)	F_{12}	4.01E+03 ± 5.08E+03$^{\approx}$	2.94E+03 ± 3.61E+03†	4.52E+03 ± 6.06E+03
F_1	3.87E−14 ± 2.71E−14‡ (100%)	1.27E−28 ± 1.84E−28† (100%)	4.42E−28 ± 6.66E−28 (100%)	F_{13}	3.25E+00 ± 8.32E−01†	2.02E+00 ± 6.12E−01†	4.25E+00 ± 9.92E−01
F_2	8.50E−02 ± 7.94E−02‡	5.69E−05 ± 6.82E−05‡	2.61E−08 ± 5.37E−08 (100%)	F_{14}	1.33E+01 ± 2.11E−01$^{\approx}$	1.32E+01 ± 1.96E−01$^{\approx}$	1.32E+01 ± 4.93E−01

."†" and "‡" indicate the *t* value is significant at a 0.05 level of significance by two-tailed *t*-test. "†", "‡" and "≈" denote the performance of the corresponding algorithm is better than, worse than, and similar to that of Orth-DE, respectively.

Table 10
Experimental results of *DE/rand/1/exp*, ODE, and OXDE over 50 independent runs at $D = 30$, after 300,000 *FES*. "Mean error" and "std dev" indicate the average and standard deviation of the function error values, respectively. Result in parentheses denotes the success rate. In the fields without parentheses, the success rates are zero. *t*-test is performed between OXDE and each of *DE/rand/1/bin* and ODE.

Inst.	*DE/rand/1/bin* Mean error ± std dev	ODE Mean error ± std dev	OXDE Mean error ± std dev	Inst.	*DE/rand/1/bin* Mean error ± std dev	ODE Mean error ± std dev	OXDE Mean error ± std dev
F_{sph}	6.88E−32 ± 9.01E−32‡ (100%)	2.53E−58 ± 4.07E−58† (100%)	4.68E−39 ± 4.10E−39 (100%)	F_3	4.82E+05 ± 2.65E+05$^{\approx}$	5.61E+05 ± 4.36E+05‡	4.18E+05 ± 3.06E+05
F_{ros}	2.33E+00 ± 1.33E+00‡	2.88E+01 ± 1.32E+01‡	1.12E+00 ± 1.06E+00 (2%)	F_4	1.69E−02 ± 1.53E−02‡	5.20E−02 ± 5.29E−02‡	1.01E−02 ± 9.80E−03
F_{ack}	2.66E−15 ± 0.00E+00$^{\approx}$ (100%)	2.66E−15 ± 0.00E+00$^{\approx}$ (100%)	2.66E−15 ± 0.00E+00 (100%)	F_5	1.84E−01 ± 1.69E−01‡	3.39E+00 ± 6.13E+00‡	1.80E−02 ± 2.25E−02
F_{grw}	0.00E+00 ± 0.00E+00$^{\approx}$ (100%)	0.00E+00 ± 0.00E+00$^{\approx}$ (100%)	0.00E+00 ± 0.00E+00 (100%)	F_6	1.94E+00 ± 1.51E+00‡ (8%)	5.53E+01 ± 4.98E+01‡	1.00E+00 ± 1.04E+00 (10%)
F_{ras}	1.27E+02 ± 3.16E+01‡	3.76E+01 ± 1.82E+01†	7.47E+01 ± 1.18E+01	F_7	3.46E−04 ± 1.84E−03$^{\approx}$ (98%)	9.74E−03 ± 8.98E−03‡ (66%)	1.03E−03 ± 3.19E−03 (96%)
F_{sch}	3.31E+01 ± 6.78E+01‡	1.98E+01 ± 4.25E+01‡	0.00E+00 ± 0.00E+00 (100%)	F_8	2.09E+01 ± 5.95E−02$^{\approx}$	2.10E+01 ± 5.08E−02‡	2.09E+01 ± 4.94E−02
F_{sal}	1.89E−01 ± 2.91E−02‡	1.49E−01 ± 5.02E−02†	1.27E−01 ± 4.13E−02	F_9	1.28E+02 ± 2.72E+01‡	5.18E+01 ± 2.06E+01†	7.03E+01 ± 1.11E+01
F_{wht}	4.83E+02 ± 5.81E+01‡	3.75E+02 ± 2.14E+01‡	3.62E+02 ± 2.07E+00	F_{10}	1.79E+02 ± 1.17E+01$^{\approx}$	5.01E+01 ± 4.79E+01†	1.72E+02 ± 1.02E+01
F_{pn1}	2.13E−32 ± 1.00E−32‡ (100%)	1.57E−32 ± 5.52E−48$^{\approx}$ (100%)	1.57E−32 ± 5.52E−48 (100%)	F_{11}	3.96E+01 ± 1.25E+00$^{\approx}$	8.55E+00 ± 8.81E+00†	3.94E+01 ± 1.16E+00
F_{pn2}	5.21E−32 ± 5.38E−32‡ (100%)	1.34E−32 ± 1.10E−47$^{\approx}$ (100%)	1.34E−32 ± 1.10E−47 (100%)	F_{12}	2.31E+03 ± 3.35E+03‡	2.49E+03 ± 2.21E+03‡	1.58E+03 ± 2.30E+03
F_1	1.31E−29 ± 4.87E−29‡ (100%)	2.31E−28 ± 2.35E−28‡ (100%)	0.00E+00 ± 0.00E+00 (100%)	F_{13}	1.49E+01 ± 1.26E+00‡	7.76E+00 ± 2.06E+00†	1.15E+01 ± 9.82E−01
F_2	4.06E−05 ± 5.86E−05$^{\approx}$	1.16E−04 ± 1.67E−04‡	5.35E−05 ± 7.89E−05	F_{14}	1.32E+01 ± 1.71E−01$^{\approx}$	1.32E+01 ± 3.02E−01$^{\approx}$	1.32E+01 ± 1.70E−01

"†" and "‡" indicate the *t* value is significant at a 0.05 level of significance by two-tailed *t*-test. "†", "‡" and "≈" denote the performance of the corresponding algorithm is better than, worse than, and similar to that of OXDE, respectively.

Table 11
Experimental results of GL-25, CMA-ES, CLPSO, and OXDE over 25 independent runs for 14 test instances at $D = 30$, after 300,000 *FES*. "Mean error" and "std dev" indicate the average and standard deviation of the function error values, respectively. *t*-test is performed between OXDE and each of GL-25, CMA-ES, and CLPSO.

Inst.	GL-25 Mean error ± std dev	CMA-ES Mean error ± std dev	CLPSO Mean error ± std dev	OXDE Mean error ± std dev
F_1	5.60E−27 ± 1.76E−26‡	1.58E−25 ± 3.35E−26‡	0.00E+00 ± 0.00E+00†	1.01E−28 ± 1.96E−28
F_2	4.04E+01 ± 6.28E+01‡	1.12E−24 ± 2.93E−25†	8.40E+02 ± 1.90E+02‡	5.68E−05 ± 7.61E−05
F_3	2.19E+06 ± 1.08E+06‡	5.54E−21 ± 1.69E−21†	1.42E+07 ± 4.19E+06‡	5.28E+05 ± 2.90E+05
F_4	9.07E+02 ± 4.25E+02‡	9.15E+05 ± 2.16E+06‡	6.99E+03 ± 1.73E+03‡	1.85E+00 ± 3.14E+00
F_5	2.51E+03 ± 1.96E+02‡	2.77E−10 ± 5.04E−11†	3.86E+03 ± 4.35E+02‡	1.82E+01 ± 5.81E+01
F_6	2.15E+01 ± 1.17E+00‡	4.78E−01 ± 1.32E+00$^{\approx}$	4.16E+00 ± 3.48E+00‡	7.97E−01 ± 1.63E+00
F_7	2.78E−02 ± 3.62E−02‡	1.82E−03 ± 4.33E−03†	4.51E−01 ± 8.47E−02‡	1.31E−02 ± 1.00E−02
F_8	2.09E+01 ± 5.94E−02$^{\approx}$	2.03E+01 ± 5.72E−01†	2.09E+01 ± 4.41E−02$^{\approx}$	2.09E+01 ± 5.23E−02
F_9	2.45E+01 ± 7.35E+00‡	4.45E+02 ± 7.12E+01‡	0.00E+00 ± 0.00E+00†	1.51E+01 ± 4.60E+00
F_{10}	1.42E+02 ± 6.45E+01‡	4.63E+01 ± 1.16E+01$^{\approx}$	1.04E+02 ± 1.53E+01‡	4.81E+01 ± 4.92E+01
F_{11}	3.27E+01 ± 7.79E+00$^{\approx}$	7.11E+00 ± 2.14E+00†	2.60E+01 ± 1.63E+00†	3.37E+01 ± 1.13E+01
F_{12}	6.53E+04 ± 4.69E+04‡	1.26E+04 ± 1.74E+04‡	1.79E+04 ± 5.24E+03‡	1.91E+03 ± 2.76E+03
F_{13}	6.23E+00 ± 4.88E+00‡	3.43E+00 ± 7.60E−01‡	2.06E+00 ± 2.15E−01$^{\approx}$	1.96E+00 ± 5.15E−01
F_{14}	1.31E+01 ± 1.84E−01†	1.47E+01 ± 3.31E−01‡	1.28E+01 ± 2.48E−01†	1.32E+01 ± 1.54E−01
†	1	6	4	
‡	11	6	8	
≈	2	2	2	

"†" and "‡" indicate the *t* value is significant at a 0.05 level of significance by two-tailed *t*-test. "†", "‡" and "≈" denote the performance of the corresponding algorithm is better than, worse than, and similar to that of OXDE, respectively.

for this purpose. In this subsection, two other sampling strategies are used and their performance is compared with that of QOX. The first is uniformly random sampling strategy and the second is Halton sampling strategy. Halton sampling is implemented by Halton distribution [11], which is a way to uniformly distribute the sampling points in a specified search space. Nine representative points generated by QOX, uniformly random sampling, and Halton sampling in a 2-dimensional search space $[0,1] \times [0,1]$ are shown in Fig. 8. From Fig. 8, it is clear that QOX is a more effective method for selecting nine representative points than the two competitors under this condition.

In order to verify that the achieved performance is really due to utilizing QOX, all parts of the proposed algorithm are kept untouched and instead of using QOX, either uniformly random sampling or Halton sampling is employed. After these modifications, the uniformly random sampling version of our algorithm is called URSDE, and the Halton sampling version of our algorithm is called HSDE.

According to the experimental studies, URSDE shows better performance than OXDE in four test instances (i.e., F_{sph}, F_{wht}, F_2, and F_4), while OXDE outperforms URSDE in eight test instances (i.e., F_{grw}, F_{ros}, F_{sch}, F_{pn2}, F_5, F_{11}, F_{12}, and F_{13}). Note that, the main shortcoming of URSDE is its sampling randomicity. Due to this shortcoming, the performance of URSDE is even significantly worse than *DE*/*rand*/1/*bin* in some test instances (such as F_{grw} and F_{sch}), which means the reliability of URSDE is not good.

In addition, OXDE is able to exhibit better performance than HSDE in nearly all test instances except for test instances F_{grw}, F_{sal}, F_{wht}, F_7, F_8, and F_{11}, and HSDE cannot perform better than OXDE even in one test instance. This is because Halton sampling with nine points only focuses on certain regions of the search space and cannot effectively represent the search space as shown in Fig. 8. As a result, some promising regions will be neglected. Furthermore, the nine points created by Halton distribution are fixed once the search space is specified. It is important to note that although the aim of Halton distribution is to uniformly sample the points from the specified search space, this aim can be achieved only when the number of sample points is relatively large as shown in Fig. 9. Since we only choose nine points from the hyper-rectangle

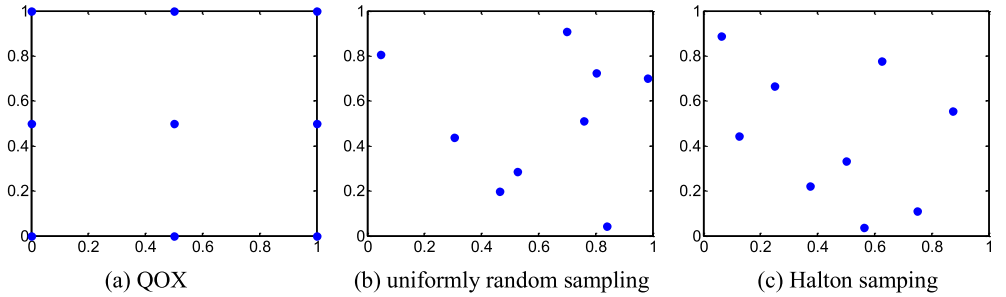

Fig. 8. Nine points produced by QOX, uniformly random sampling, and Halton sampling in a 2-dimensional search space $[0,1] \times [0,1]$, respectively.

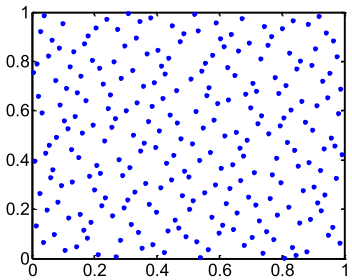

Fig. 9. 250 points produced by Halton sampling in a 2-dimensional search space $[0,1] \times [0,1]$.

defined by the target and mutant vectors in this paper, HSDE does not function effectively to optimize the 24 benchmark test instances.

The above discussion clearly demonstrates that the same level of improvement cannot be achieved via additional sampling strategies.

6.11. More discussions

In OXDE, which is an illustrative algorithm of our framework, the orthogonal array used is $L_9(3^4)$, only one target vector is randomly selected at each generation for QOX, and the scaling factor F is a uniform random number between 0 and 1 in the mutation operator before QOX, and is 0.9 otherwise. The main motivations for these choices are twofold: (1) we attempt to minimize the computational cost incurred by QOX; and (2) we aim at not increasing the number of control parameters dramatically. In this subsection, we study the following four issues:

- What is the effect of the frequency of the use of QOX?
- What is the effect of the size of the orthogonal array?
- Is it better to choose the best individual in the current population for QOX?
- Is it better to not use the random scaling factor F in the mutation operator before QOX?

The first experiment is designed to investigate the impact of the rate at which QOX is applied on the performance of OXDE. Five different rates are tested: 0.005, 0.01, 1/30, 0.05, and 0.1. Note that the rate of the original OXDE is 1/30 since QOX is only applied to one randomly selected individual in the population. Table 12 compares the performance for different rates. From Table 12, OXDE with a smaller rate (i.e., 0.005) achieves a slightly better error average for test instances F_{sal}, F_{pn1}, and F_8;however, it provides the worst error average for test instances F_{sph}, F_{ras}, F_{sch}, F_1, F_2, F_3, F_{10}, F_{13}, and F_{14}. Similarly, OXDE with a higher rate (i.e., 0.01) achieves a better error average for test instances F_{sph}, F_{wht}, F_3, F_{10}, F_{11}, and F_{14}; however, it provides the worst error average for test instances F_{ros}, F_{ack}, F_{grw}, F_{sal}, F_{pn1}, F_{pn2}, F_4, F_5,F_6, F_7, F_8, F_9, and F_{12}. On the contrary, OXDE with the rates of 0.01, 1/30, and 0.05 show overall superior performance for all the 24 test instances. As a consequence, a rate between 0.01 and 0.05 is recommended for applying QOX to DE in this paper.

In this paper, we use nine sample points of QOX for test instances from 50 to 200 dimensions. Although nine sample points are effective for low-dimensional test instances, higher-dimensional test instances might require more sample points. Therefore, the effect of the number of sample points of QOX on the performance of OXDE has been investigated in this subsection, by sampling 9 ($L_9(3^4)$), 25 ($L_{25}(5^6)$), and 49 ($L_{49}(7^8)$) points for test instances with 30 dimensions, and 9 ($L_9(3^4)$), 25 ($L_{25}(5^6)$), 49 ($L_{49}(7^8)$), and 121 ($L_{121}(11^{12})$) points for test instances with 200 dimensions. The experimental results are presented in Tables 13 and 14.

From Table 13, it is interesting to see that out of the 24 test instances, in 15 cases, OXDE with nine sampling points is better than the two competitors in terms of the error average for test instances with 30 dimensions. In three test instances (F_{grw}, F_{10}, and F_{11}), OXDE with 25 sampling points achieves the best average function error values, and in five test instances (F_{wht}, F_{pn1}, F_1, F_{12}, and F_{14}), OXDE with 49 sampling points provides the error average of higher quality.

As shown in Table 14, when the dimension of test instances is equal to 200, OXDE with 25 sample points and OXDE with 49 sample points exhibit similar results which are slightly better than the results provided by OXDE with nine sample points for all the test instances, signifying that the performance of OXDE can be further improved with the growth of the number of sample points. It seems that the average function error values derived from OXDE with 121 sample points are better than those provided by the other algorithms for most test instances, however, test instances F_{sch}, F_{wht}, F_{pn1}, and F_{pn2} are exceptions where the performance of OXDE with 121 sample points is outperformed by the other algorithms.

Based on the analysis performed in this experiment, we conclude that for relatively lower dimensional test instances (such as 30 dimensions), the use of nine sample points for OXDE turns out to be beneficial in most cases. With the increase in problem dimensionality of test instances, a relatively larger number of sample points has the capability to improve the performance of OXDE. Note, however, that it does not mean that the larger number of sample points, the better performance

Table 12
Experimental results of OXDE over 50 independent runs with varying frequency of QOX at $D = 30$. "Mean error" and "std dev" indicate the average and standard deviation of the function error values, respectively. Result in parentheses denotes the success rate. In the fields without parentheses, the success rates are zero. The best result for each test instance among the compared algorithms is highlighted in **boldface**.

Inst.	0.005	0.01	1/30	0.05	0.1
F_{sph}	5.18E−28 ± 2.06E−27 (100%)	3.65E−36 ± 1.18E−36 (100%)	5.21E−59 ± 1.82E−58 (100%)	1.01E−66 ± 1.86E−66 (100%)	**2.17E−75 ± 8.06E−75 (100%)**
F_{ros}	3.81E−01 ± 1.07E+00 (14%)	**2.39E−01 ± 9.56E−01 (84%)**	4.78E−01 ± 1.30E+00 (88%)	1.03E+00 ± 1.76E+00 (74%)	1.03E+00 ± 1.76E+00 (74%)
F_{ack}	5.79E−15 ± 2.55E−15 (100%)	**2.66E−15 ± 0.00E+00 (100%)**	**2.66E−15 ± 0.00E+00 (100%)**	9.31E−02 ± 2.82E−01 (90%)	4.25E−01 ± 7.89E−01 (70%)
F_{grw}	2.90E−03 ± 4.79E−03 (70%)	**1.07E−03 ± 3.07E−03 (88%)**	1.82E−03 ± 4.44E−03 (84%)	1.52E−03 ± 4.71E−03 (88%)	4.57E−03 ± 8.24E−03 (72%)
F_{ras}	1.80E+01 ± 7.02 + 00	1.41E+01 ± 5.15E+00	**8.99E+00 ± 2.29E+00**	9.11E+00 ± 2.53E+00	1.03E+01 ± 3.01E+00
F_{sch}	2.84E+01 ± 6.59E+01	1.18E+01 ± 4.31E+01	**3.81E−04 ± 0.00E+00**	**3.81E−04 ± 0.00E+00**	6.71E+00 ± 3.46E+01
F_{sal}	**1.91E−01 ± 2.65E−02**	**1.91E−01 ± 3.94E−02**	2.01E−01 ± 4.16E−02	2.38E−01 ± 5.64E−02	3.03E−01 ± 9.24E−02
F_{wht}	3.37E+02 ± 8.00E+01 (2%)	3.55E+02 ± 4.22E+01 (2%)	3.35E+02 ± 7.83E+01 (2%)	2.96E+02 ± 1.07E+02 (2%)	**2.80E+02 ± 1.17E+02 (4%)**
F_{pn1}	**4.14E−03 ± 2.05E−02 (96%)**	1.03E−02 ± 8.01E−02 (96%)	1.03E−02 ± 7.32E−02 (98%)	2.48E−02 ± 8.52E−02 (88%)	3.32E−02 ± 1.31E−01 (88%)
F_{pn2}	5.88E−27 ± 3.87E−26 (100%)	3.38E−32 ± 5.32E−32 (100%)	**2.25E−32 ± 6.37E−32 (100%)**	3.65E−32 ± 6.94E−32 (100%)	7.30E−02 ± 5.08E−01 (90%)
F_1	6.96E−28 ± 1.12E−27 (100%)	**2.92E−29 ± 6.61E−29 (100%)**	1.27E−28 ± 1.84E−28 (100%)	1.21E−28 ± 1.66E−28 (100%)	1.94E−28 ± 2.17E−28 (100%)
F_2	9.01E−03 ± 8.10E−03	2.89E−03 ± 4.32E−03	5.69E−05 ± 6.82E−05	**1.81E−05 ± 1.89E−05**	2.87E−05 ± 6.99E−05 (10%)
F_3	7.57E+05 ± 3.97E+05	6.00E+05 ± 3.05E+05	5.41E+05 ± 2.86E+05	5.89E+05 ± 3.76E+05	**4.09E+05 ± 2.06E+05**
F_4	6.48E+00 ± 7.58E+00	5.91E+00 ± 1.12E+01	**2.58E+00 ± 3.91E+00**	3.46E+00 ± 5.23E+00	1.61E+01 ± 2.42E+01
F_5	9.30E+00 ± 2.05E+01	**3.74E+00 ± 1.12E+01**	5.72E+00 ± 1.18E+01	2.01E+01 ± 2.89E+01	1.93E+02 ± 1.51E+02
F_6	8.59E−01 ± 1.60E+00 (40%)	**4.78E−01 ± 1.30E+00 (88%)**	7.97E−01 ± 1.61E+00 (80%)	7.97E−01 ± 1.61E+00 (80%)	1.27E+00 ± 1.87E+00 (68%)
F_7	5.41E−03 ± 8.42E−03 (78%)	**6.79E−03 ± 7.31E−03 (78%)**	9.98E−03 ± 9.50E−03 (68%)	1.09E−02 ± 1.05E−02 (68%)	1.63E−02 ± 1.40E−02 (38%)
F_8	**2.09E+01 ± 4.64E−02**	**2.09E+01 ± 4.44E−02**	**2.09E+01 ± 5.48E−02**	**2.09E+01 ± 4.70E−02**	2.10E+01 ± 5.23E−02
F_9	1.63E+01 ± 4.79E+00	**1.48E+01 ± 4.54E+00**	1.51E+01 ± 4.07E+00	1.89E+01 ± 5.23E+00	2.41E+01 ± 6.12E+00
F_{10}	9.25E+01 ± 7.51E+01	8.18E+01 ± 7.28E+01	4.70E+01 ± 4.71E+01	4.31E+01 ± 4.29E+01	**4.11E+01 ± 1.06E+01**
F_{11}	3.23E+01 ± 1.27E+01	3.35E+01 ± 1.22E+01	3.36E+01 ± 1.13E+01	3.29E+01 ± 1.16E+01	**2.58E+01 ± 1.29E+01**
F_{12}	3.64E+03 ± 4.65E+03	**2.87E+03 ± 3.54E+03**	2.94E+03 ± 3.61E+03	3.41E+03 ± 6.57E+03	4.54E+03 ± 7.73E+03
F_{13}	3.15E+00 ± 2.05E+00	2.51E+00 ± 1.73E+00	2.02E+00 ± 6.12E−01	**1.84E+00 ± 4.77E−01**	2.03E+00 ± 3.80E−01
F_{14}	1.34E+01 ± 1.47E−01	1.33E+01 ± 3.18E−01	1.32E+01 ± 1.96E−01	1.31E+01 ± 3.28E−01	**1.29E+01 ± 2.69E−01**

Table 13
Experimental results of OXDE over 50 independent runs with varying sampling points of QOX at $D = 30$. "Mean error" and "std dev" indicate the average and standard deviation of the function error values, respectively. Result in parentheses denotes the success rate. In the fields without parentheses, the success rates are zero. The best result for each test instance among the compared algorithms is highlighted in **boldface**.

Inst.	9	25	49	Inst.	9	25	49
F_{sph}	**5.21E−59 ± 1.82E−58 (100%)**	7.60E−49 ± 3.14E−48† (100%)	5.51E−37 ± 9.75E−37 (100%)	F_3	**5.41E+05 ± 2.86E+05**	6.34E+05 ± 3.24E+05	7.44E+05 ± 4.84E+05
F_{ros}	**4.78E−01 ± 1.30E+00 (88%)**	9.56E−01 ± 1.71E+00 (76%)	1.23E+00 ± 2.47E+00 (38%)	F_4	**2.58E+00 ± 3.91E+00**	1.27E+01 ± 2.86E+01	4.07E+01 ± 4.58E+01
F_{ack}	**2.66E−15 ± 0.00E+00 (100%)**	8.26E−02 ± 2.87E−01 (92%)	4.17E−02 ± 2.07E−01 (96%)	F_5	**5.72E+00 ± 1.18E+01**	3.68E+01 ± 4.88E+01	1.24E+02 ± 2.00E+02
F_{grw}	1.82E−03 ± 4.44E−03 (84%)	**9.85E−04 ± 3.37E−03 (90%)**	2.46E−03 ± 5.33E−03 (78%)	F_6	**7.97E−01 ± 1.61E+00 (80%)**	1.51E+00 ± 1.95E+00 (62%)	1.61E+00 ± 1.79E+00 (30%)
F_{ras}	**8.99E+00 ± 2.29E+00**	1.15E+01 ± 4.17E+00	1.27E+01 ± 3.69E+00	F_7	**9.98E−03 ± 9.50E−03 (68%)**	1.04E−02 ± 9.18E−03 (54%)	1.28E−02 ± 9.40E−03 (50%)
F_{sch}	**3.81E−04 ± 0.00E+00**	3.07E+01 ± 6.24E+01	1.29E+02 ± 1.16E+02	F_8	2.09E+01 ± 5.48E−02	2.09E+01 ± 4.23E−02	2.09E+01 ± 5.40E−02
F_{sal}	**2.01E−01 ± 4.16E−02**	2.51E−01 ± 6.43E−02	2.81E−01 ± 6.90E−02	F_9	**1.51E+01 ± 4.07E+00**	1.96E+01 ± 6.46E+00	2.36E+01 ± 6.52E+00
F_{wht}	3.35E+02 ± 7.83E+01 (2%)	3.26E+02 ± 8.30E+01 (2%)	**2.85E+02 ± 1.01E+02 (2%)**	F_{10}	4.70E+01 ± 4.71E+01	**4.68E+01 ± 4.07E+01**	4.97E+01 ± 3.64E+01
F_{pn1}	1.03E−02 ± 7.32E−02 (98%)	4.14E−02 ± 1.29E−01 (88%)	**8.29E−03 ± 2.84E−02 (92%)**	F_{11}	3.36E+01 ± 1.13E+01	**3.20E+01 ± 1.14E+01**	3.55E+01 ± 7.70E+00
F_{pn2}	**2.25E−32 ± 6.37E−32 (100%)**	3.87E−32 ± 4.72E−32 (100%)	1.04E−01 ± 7.32E−01 (88%)	F_{12}	2.94E+03 ± 3.61E+03	3.27E+03 ± 4.35E+03	**2.06E+03 ± 2.05E+03**
F_1	1.27E−28 ± 1.84E−28 (100%)	2.04E−29 ± 5.58E−29 (100%)	**1.13E−29 ± 4.87E−29 (100%)**	F_{13}	**2.02E+00 ± 6.12E−01**	2.33E+00 ± 6.99E−01	3.57E+00 ± 2.02E+00
F_2	**5.69E−05 ± 6.82E−05**	1.13E−03 ± 1.11E−03	1.11E−02 ± 1.38E−02	F_{14}	1.32E+01 ± 1.96E−01	1.29E+01 ± 3.32E−01	**1.26E+01 ± 3.62E−01**

Table 14
Experimental results of OXDE over 50 independent runs with varying sampling points of QOX at $D = 200$. "Mean error" and "Std Dev" indicate the average and standard deviation of the function error values, respectively. Result in parentheses denotes the success rate. In the fields without parentheses, the success rates are zero. The best result for each test instance among the compared algorithms is highlighted in **boldface**.

Inst.	9	25	49	121
F_{sph}	2.44E+00 ± 5.31E−01	7.32E−01 ± 1.75E−01	8.55E−01 ± 1.67E−01	**4.45E−01 ± 9.55E−02**
F_{ros}	4.30E+03 ± 1.20E+03	1.97E+03 ± 4.77E+02	2.43E+03 ± 6.07E+02	**1.06E+03 ± 2.11E+02**
F_{ack}	6.83E−01 ± 1.85E−01	3.04E−01 ± 2.07E−01	3.16E−01 ± 1.95E−01	**9.81E−02 ± 1.77E−02**
F_{grw}	5.46E−01 ± 7.93E−02	2.06E−01 ± 4.44E−02	2.45E−01 ± 4.57E−02	**1.63E−01 ± 2.69E−02**
F_{ras}	1.03E+03 ± 7.94E+01	8.33E+02 ± 1.95E+02	8.75E+02 ± 1.72E+02	**6.11E+02 ± 6.09E+01**
F_{sch}	1.76E+01 ± 6.80E+01	**1.22E+00 ± 3.47E−01**	5.95E+00 ± 3.07E+01	4.49E+03 ± 3.57E+02
F_{sal}	1.94E+00 ± 1.33E−01	1.33E+00 ± 7.36E−02	1.27E+00 ± 9.13E−02	**8.27E−01 ± 8.26E−02**
F_{wht}	8.56E+07 ± 8.87E+07	2.38E+07 ± 1.28E+07	**1.82E+07 ± 3.24E+06**	2.23E+08 ± 1.29E+08
F_{pn1}	9.64E−02 ± 7.40E−02	**4.70E−02 ± 5.65E−02**	5.17E−02 ± 4.27E−02	1.27E−01 ± 3.52E−02
F_{pn2}	4.41E+00 ± 2.24E+00	**3.13E+00 ± 3.53E+00**	3.83E+00 ± 3.09E+00	4.71E+00 ± 1.82E+00

of OXDE, since too large number of sample points might also result in the degradation of the performance for some test instances.

To study the third issue, we have tested a variant of OXDE, OXDE-1, in which the best individual, instead of a randomly selected one, is chosen for QOX. The comparison results between OXDE-I and OXDE are given in Table 15.

Table 15
Experimental results of OXDE and OXDE-1 over 50 independent runs for test instances with 30 variables, after 300,000 *FES*. "Mean error" and "std dev" indicate the average and standard deviation of the function error values, respectively. Percentages in parentheses denote the success rates. In the fields without parentheses, the success rates are zero. *t*-test is performed between OXDE and OXDE-1.

Inst.	OXDE Mean error ± std dev	OXDE-1 Mean error ± std dev	Inst.	OXDE Mean error ± std dev	OXDE-1 Mean error ± std dev
F_{sph}	5.21E−59 ± 1.82E−58† (100%)	1.75E−22 ± 3.91E−22 (100%)	F_3	5.41E+05 ± 2.86E+05†	7.66E+05 ± 4.81E+05
F_{ros}	4.78E−01 ± 1.30E+00† (88%)	1.00E+00 ± 1.86E+00	F_4	2.58E+00 ± 3.91E+00†	1.50E+01 ± 1.47E+01
F_{ack}	2.66E−15 ± 0.00E+00† (100%)	1.70E−12 ± 1.92E−12 (100%)	F_5	5.72E+00 ± 1.18E+01†	5.37E+01 ± 6.98E+01
F_{grw}	1.82E−03 ± 4.44E−03† (84%)	4.77E−03 ± 7.43E−03 (62%)	F_6	7.97E−01 ± 1.61E+00$^{\approx}$ (80%)	1.02E+00 ± 1.98E+00 (22%)
F_{ras}	8.99E+00 ± 2.29E+00†	2.38E+01 ± 7.48E+00	F_7	9.98E−03 ± 9.50E−03$^{\approx}$ (68%)	8.41E−03 ± 1.00E−02 (70%)
F_{sch}	0.00E+00 ± 0.00E+00† (100%)	1.12E+02 ± 1.09E+02	F_8	2.09E+01 ± 5.48E−02$^{\approx}$	2.09E+01 ± 5.02E−02
F_{sal}	2.01E−01 ± 4.16E−02‡	1.89E−01 ± 3.02E−02	F_9	1.51E+01 ± 4.07E+00†	2.07E+01 ± 6.41E+00
F_{wht}	3.35E+02 ± 7.83E+01$^{\approx}$ (2%)	3.22E+02 ± 9.78E+01 (2%)	F_{10}	4.70E+01 ± 4.71E+01†	1.23E+02 ± 7.22E+01
F_{pn1}	1.03E−02 ± 7.32E−02$^{\approx}$ (98%)	1.86E−02 ± 9.06E−02 (92%)	F_{11}	3.36E+01 ± 1.13E+01†	3.71E+01 ± 7.88E+00
F_{pn2}	2.25E−32 ± 6.37E−32† (100%)	8.78E−04 ± 3.01E−03 (92%)	F_{12}	2.94E+03 ± 3.61E+03†	3.49E+03 ± 4.75E+03
F_1	1.27E−28 ± 1.84E−28† (100%)	1.54E−22 ± 4.62E−22 (100%)	F_{13}	2.02E+00 ± 6.12E−01†	2.53E+00 ± 6.11E−01
F_2	5.69E−05 ± 6.82E−05†	3.01E−02 ± 2.94E−02	F_{14}	1.32E+01 ± 1.96E−01†	1.33E+01 ± 1.53E−01

"†" and "‡" indicate the *t* value is significant at a 0.05 level of significance by two-tailed *t*-test. "†", "‡" and "≈" denote the performance of OXDE is better than, worse than, and similar to that of OXDE-1, respectively.

Table 16
Experimental results of OXDE and OXDE-2 over 50 independent runs for instances with 30 variables, after 300,000 *FES*. "Mean error" and "std dev" indicate the average and standard deviation of the function error values, respectively. Percentages in parentheses denote the success rates. In the fields without parentheses, the success rates are zero. *t*-test is performed between OXDE and OXDE-2.

Inst.	OXDE Mean error ± std dev	OXDE-2 Mean error ± std dev	Inst.	OXDE Mean error ± std dev	OXDE-2 Mean error ± std dev
F_{sph}	5.21E−59 ± 1.82E−58† (100%)	4.62E−33 ± 5.31E−33 (100%)	F_3	5.41E+05 ± 2.86E+05†	8.23E+05 ± 4.48E+05
F_{ros}	4.78E−01 ± 1.30E+00$^{\approx}$ (88%)	2.36E−01 ± 9.56E−01 (94%)	F_4	2.58E+00 ± 3.91E+00†	2.00E+01 ± 1.78E+01
F_{ack}	2.66E−15 ± 0.00E+00† (100%)	2.94E−15 ± 9.73E−16 (100%)	F_5	5.72E+00 ± 1.18E+01$^{\approx}$	2.91E+00 ± 1.72E+01
F_{grw}	1.82E−03 ± 4.44E−03$^{\approx}$ (84%)	2.11E−03 ± 4.16E−03 (78%)	F_6	7.97E−01 ± 1.61E+00‡ (80%)	2.39E−01 ± 9.56E−01 (94%)
F_{ras}	8.99E+00 ± 2.29E+00†	4.86E+01 ± 3.77E+01	F_7	9.98E−03 ± 9.50E−03‡ (68%)	4.72E−03 ± 7.13E−03 (86%)
F_{sch}	0.00E+00 ± 0.00E+00$^{\approx}$ (100%)	0.00E+00 ± 0.00E+00 (100%)	F_8	2.09E+01 ± 5.48E−02$^{\approx}$	2.09E+01 ± 5.35E−02
F_{sal}	2.01E−01 ± 4.16E−02$^{\approx}$	1.82E−01 ± 3.57E−02	F_9	1.51E+01 ± 4.07E+00†	4.53E+01 ± 3.54E+01
F_{wht}	3.35E+02 ± 7.83E+01† (2%)	3.61E+02 ± 1.05E+01 (2%)	F_{10}	4.70E+01 ± 4.71E+01†	1.42E+02 ± 7.21E+01
F_{pn1}	1.03E−02 ± 7.32E−02$^{\approx}$ (98%)	8.29E−03 ± 5.86E−02 (98%)	F_{11}	3.36E+01 ± 1.13E+01†	3.67E+01 ± 7.60E+00
F_{pn2}	2.25E−32 ± 6.37E−32$^{\approx}$ (100%)	6.39E−32 ± 3.34E−31 (100%)	F_{12}	2.94E+03 ± 3.61E+03†	3.39E+03 ± 4.35E+03
F_1	1.27E−28 ± 1.84E−28‡ (100%)	2.62E−29 ± 8.04E−29 (100%)	F_{13}	2.02E+00 ± 6.12E−01†	1.07E+01 ± 1.71E+00
F_2	5.69E−05 ± 6.82E−05†	4.75E−02 ± 4.53E−02	F_{14}	1.32E+01 ± 1.96E−01†	1.33E+01 ± 2.04E−01

"†" and "‡" indicate the *t* value is significant at a 0.05 level of significance by two-tailed *t*-test. "†", "‡" and "≈" denote the performance of OXDE is better than, worse than, and similar to that of OXDE-2, respectively.

Table 15 shows that OXDE-1 significantly outperforms OXDE in terms of the solution quality only in one instance (F_{sal}), while OXDE is significantly better than OXDE-1 in 18 out of 24 test instances. Therefore, we can claim that it is better to randomly choose an individual in the current population as the target vector for QOX. The choice of the best individual as a target vector for QOX might deteriorate the search ability of the algorithm.

To address the fourth issue, we have tested another variant of OXDE, OXDE-2, in which F is set to 0.9 instead of $rand(0,1)$ in mutation before QOX. The comparison results between OXDE-2 and OXDE are presented in Table 16.

Table 16 suggests that OXDE-2 can beat OXDE only in three test instances: F_1, F_6, and F_7. However, OXDE significantly outperforms OXDE-2 in 13 test instances. Therefore, we can conclude that it is better to use random scaling factor F in the mutation operator before QOX. The reason why randomness in scaling factor F can improve the algorithm performance could be because it can introduce more variation to the search and thus strengthen the search ability.

7. Conclusion

The commonly used crossover operators in current popular DE only visit one vertex of the hyper-rectangle defined by the mutant and target vectors, this could confine the algorithm search ability. In this paper, we have attempted to introduce QOX into DE for overcoming this shortcoming. QOX is able to make a systematic and rational search in the region defined by the two parent solutions. We have suggested a framework which uses QOX in DE and proposed a hybrid algorithm of *DE*/*rand*/1/*bin* with QOX, OXDE. Our framework takes advantage of both DE mutation and QOX, and it is easy to implement and does not introduce any complicated operators. In this paper, extensive experiments have been carried out to study the performance of OXDE and to demonstrate that our framework can also be used to enhance the performance of other variants of DE. We have also experimentally studied several issues in OXDE.

We would like to point out that OX operators, like most reproduction operators in EAs, are not rotation-invariant processes, which implies that if we rotate the coordinate system, the set of the sample points will change. This dependence of performance on rotation is often seen as detrimental to a global optimizer. So, future work will focus on studying the behavior of our framework and further improving our framework to enable it to solve rotated problems more effectively. In this paper, we distribute fixed computational effort (i.e., nine *FES*) of QOX for DE at each generation during the evolution. Nevertheless, how to adaptively distribute the computational effort of QOX (including adaptively determining the number of the target vectors which undergo QOX and the number of the sample points for QOX) according to the population information obtained from the search is also an attractive topic for future research.

The MATLAB source code of OXDE can be downloaded from Q. Zhang's homepage: http://www.dces.essex.ac.uk/staff/qzhang/.

Acknowledgments

This work was supported in part by the National Natural Science Foundation of China (Grant Nos. 60805027, 90820302, and 61175064), and in part by the Research Fund for the Doctoral Program of Higher Education of China (Grant No. 200805330005). The authors thank the anonymous reviewers for their very helpful and constructive comments and suggestions.

References

[1] M.M. Ali, A. Törn, Population set-based global optimization algorithms: some modifications and numerical studies, Computers and Operations Research 31 (10) (2004) 1703–1725.
[2] S. Aydin, H. Temeltas, Fuzzy-differential evolution algorithm for planning time-optimal trajectories of a unicycle mobile robot on a predefined path, Advanced Robotics 18 (7) (2004) 725–748.
[3] A.S.S.M. Barkat Ullah, R. Sarker, D. Cornforth, A combined MA-GA approach for solving constrained optimization problems, in: Proceedings of the 6th IEEE/ACIS International Conference on Computer and Information Science (ICIS 2007), 2007, pp. 382–387.
[4] A.S.S.M. Barkat Ullah, R. Sarker, D. Cornforth, C. Lokan, An agent-based memetic algorithm (AMA) for solving constrained optimization problems, in: Proceedings of the 2007 IEEE Congress on Evolutionary Computation, 2007, pp. 999–1006.
[5] P. Bhowmik, S. Das, A. Konar, S. Das, A.K. Nagar, A new differential evolution with improved mutation strategy, in: Proceedings of the 2010 IEEE Congress on Evolutionary Computation, 2010, pp. 1–8.
[6] J. Brest, S. Greiner, B. Boskovic, M. Mernik, V. Zumer, Self-adapting control parameters in differential evolution: a comparative study on numerical benchmark problems, IEEE Transactions on Evolutionary Computation 10 (6) (2006) 646–657.
[7] S. Das, A. Abraham, U.K. Chakraborty, A. Konar, Differential evolution using a neighborhood-based mutation operator, IEEE Transactions on Evolutionary Computation 13 (3) (2009) 526–553.
[8] S. Das, A. Konar, Automatic image pixel clustering with an improved differential evolution, Applied Soft Computing 9 (1) (2009) 226–236.
[9] S. Das, A. Konar, U.K. Chakraborty, Two improved differential evolution schemes for faster global search, in: Proceedings of the Genetic and Evolutionary Computation Conference (GECCO), June 2005, pp. 991–998.
[10] S. Das, P.N. Suganthan, Differential evolution: a survey of the state-of-the-art, IEEE Transactions on Evolutionary Computation 15 (1) (2011) 4–31.
[11] R. Ewaryst, R. Schwabe, Halton and Hammersley sequences in multivariate nonparametric regression, Statistics and Probability Letters 76 (8) (2006) 803–812.
[12] H.Y. Fan, J. Lampinen, A trigonometric mutation operation to differential evolution, Journal of Global Optimization 27 (1) (2003) 105–129.
[13] K.T. Fang, Y. Wang, Number–Theoretic Methods in Statistics, Chapman and Hall, New York, 1994.
[14] V. Feoktistov, S. Janaqi, Generalization of the strategies in differential evolution, in: Proceedings of the 18th International Parallel and Distributed Processing Symposium, Santa Fe, 2004, pp. 165–170.
[15] C. Garcia-Martinez, M. Lozano, F. Herrera, D. Molina, A.M. Sanchez, Global and local real-coded genetic algorithms based on parent-centric crossover operators, European Journal of Operational Research 185 (3) (2008) 1088–1113.

[16] A. Ghosh, S. Das, A. Chowdhury, R. Giri, An improved differential evolution algorithm with fitness-based adaptation of the control parameters, Information Sciences 181 (18) (2011) 3749–3765.
[17] W. Gong, Z. Cai, C.X. Ling, Enhancing the performance of differential evolution using orthogonal design method, Applied Mathematics and Computation 206 (1) (2008) 56–69.
[18] W. Gong, Z. Cai, C.X. Ling, H. Li, Enhanced differential evolution with adaptive strategies for numerical optimization, IEEE Transactions on Systems, Man, and Cybernetics: Part B – Cybernetics 41 (2) (2011) 397–413.
[19] N. Hansen, A. Ostermeier, Completely derandomized self-adaptation in evolution strategies, Evolutionary Computation 9 (2) (2001) 159–195.
[20] S. Ho, L. Shu, J. Chen, Intelligent evolutionary algorithms for large parameter optimization problems, IEEE Transactions on Evolutionary Computation 8 (6) (2004) 522–541.
[21] J. Ilonen, J.K. Kamarainen, J. Lampinen, Differential evolution training algorithm for feed-forward neural networks, Neural Processing Letters 17 (1) (2003) 93–105.
[22] D. Jia, G. Zheng, M.K. Khan, An effective memetic differential evolution algorithm based on chaotic local search, Information Sciences 181 (15) (2011) 3175–3187.
[23] M.H. Lee, C.H. Han, K.S. Chang, Dynamic optimization of a continuous polymer reactor using a modified differential evolution algorithm, Industrial and Engineering Chemistry Research 38 (12) (1999) 4825–4831.
[24] Y.W. Leung, Y. Wang, An orthogonal genetic algorithm with quantization for global numerical optimization, IEEE Transactions on Evolutionary Computation 5 (1) (2001) 41–53.
[25] J.J. Liang, A.K. Qin, P.N. Suganthan, S. Baskar, Comprehensive learning particle swarm optimizer for global optimization of multimodal functions, IEEE Transactions on Evolutionary Computation 10 (3) (2006) 281–295.
[26] J. Liu, J. Lampinen, A fuzzy adaptive differential evolution algorithm, Soft Computing – A Fusion of Foundations Methodologies and Applications 9 (6) (2005) 448–462.
[27] R. Mallipeddi, P.N. Suganthan, Q.K. Pan, M.F. Tasgetiren, Differential evolution algorithm with ensemble of parameters and mutation strategies, Applied Soft Computing 11 (2) (2011) 1679–1696.
[28] E. Mezura-Montes, J. Velázquez-Reyes, C.A. Coello Coello, Modified differential evolution for constrained optimization, in: Proceedings of the 2006 IEEE Congress on Evolutionary Computation (CEC'2006), IEEE Press, Sheraton Vancouver Wall Centre Hotel, Vancouver, BC, Canada, 2006, pp. 332–339.
[29] F. Neri, G. Iacca, E. Mininno, Disturbed exploitation compact differential evolution for limited memory optimization problems, Information Sciences 181 (12) (2011) 2469–2487.
[30] N. Noman, H. Iba, Differential evolution for economic load dispatch problems, Electric Power Systems Research 78 (8) (2008) 1322–1331.
[31] N. Noman, H. Iba, Accelerating differential evolution using an adaptive local search, IEEE Transactions on Evolutionary Computation 12 (1) (2008) 107–125.
[32] A.K. Qin, V.L. Huang, P.N. Suganthan, Differential evolution algorithm with strategy adaptation for global numerical optimization, IEEE Transactions on Evolutionary Computation 13 (2) (2009) 398–417.
[33] A.K.Qin, P.N. Suganthan, Self-adaptive differential evolution algorithm for numerical optimization, in: Proceeding of the 2005 IEEE Congress on Evolutionary Computation, 2005, pp. 1785–1791.
[34] S. Rahnamayan, H.R. Tizhoosh, M.M.A. Salama, Opposition-based differential evolution, IEEE Transactions on Evolutionary Computation 12 (1) (2008) 64–79.
[35] R. Storn, System design by constraint adaptation and differential evolution, IEEE Transactions on Evolutionary Computation 3 (1) (1999) 22–34.
[36] R. Storn, K. Price, Differential evolution – A simple and efficient adaptive scheme for global optimization over continuous spaces, Berkeley, CA, Tech. Rep. TR-95-012, 1995.
[37] 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.
[38] P.N. Suganthan, N. Hansen, J.J. Liang, K. Deb, Y.P. Chen, A. Auger, S. Tiwari, Problem definitions and evaluation criteria for the CEC 2005 special session on real-parameter optimization, Nanyang Tech. Univ., Singapore and KanGAL, Kanpur Genetic Algorithms Lab., IIT, Kanpur, India, Tech. Rep., Rep. No. 2005005, May 2005.
[39] J. Sun, Q. Zhang, E.P.K. Tsang, DE/EDA: a new evolutionary algorithm for global optimization, Information Sciences 169 (3-4) (2005) 249–262.
[40] M.F. Tasgetiren, P.N. Suganthan, A multi-populated differential evolution algorithm for solving constrained optimization problem, in: Proceedings of the 2006 IEEE Congress on Evolutionary Computation (CEC'2006), IEEE Press, Sheraton Vancouver Wall Centre Hotel, Vancouver, BC, Canada, 2006, pp. 33–40.
[41] J. Teo, Exploring dynamic self-adaptive populations in differential evolution, Soft Computing 10 (8) (2006) 637–686.
[42] J. Tsai, T. Liu, J. Chou, Hybrid Taguchi-genetic algorithm for global numerical optimization, IEEE Transactions on Evolutionary Computation 8 (4) (2004) 365–377.
[43] Y. Wang, Z. Cai, Q. Zhang, Differential evolution with composite trial vector generation strategies and control parameters, IEEE Transactions on Evolutionary Computation 15 (1) (2011) 55–66.
[44] Y. Wang, C. Dang, An evolutionary algorithm for global optimization based on level set evolution and Latin squares, IEEE Transactions on Evolutionary Computation 11 (5) (2007) 579–595.
[45] Y. Wang, H. Liu, Z. Cai, Y. Zhou, An orthogonal design based constrained evolutionary optimization algorithm, Engineering Optimization 39 (6) (2007) 715–736.
[46] F.S. Wang, T.L. Su, H.J. Jang, Hybrid differential evolution for problems of kinetic parameter estimation and dynamic optimization of an ethanol fermentation process, Industrial and Engineering Chemistry Research 40 (13) (2001) 2876–2885.
[47] M. Weber, F. Neri, V. Tirronen, A study on scale factor in distributed differential evolution, Information Sciences 181 (12) (2011) 2488–2511.
[48] Z.Y. Yang, K. Tang, X. Yao, Large scale evolutionary optimization using cooperative coevolution, Information Sciences 87 (15) (2008) 2985–2999.
[49] S.Y. Zeng, L.S. Kang, L.X. Ding, An orthogonal multi-objective evolutionary algorithm for multi-objective optimization problems with constraints, Evolutionary Computation 12 (1) (2004) 77–98.
[50] Q. Zhang, Y.W. Leung, Orthogonal genetic algorithm for multimedia multicast routing, IEEE Transactions on Evolutionary Computation 3 (1) (1999) 53–62.
[51] Q. Zhang, W. Peng, S. Wu, Genetic algorithm + orthogonal design method: a new global optimization algorithm, in: Proceedings of the 4th Chinese Joint Conference of Artificial Intelligence, Qinghua University Press, Beijing, 1996, pp. 127–133.
[52] J. Zhang, A.C. Sanderson, JADE: adaptive differential evolution with optional external archive, IEEE Transactions on Evolutionary Computation 13 (5) (2009) 945–958.