Enhancing the performance of differential evolution using orthogonal design method

Wenyin Gong [a,1], Zhihua Cai [a] Liangxiao Jiang [a]

[a] *School of Computer Science,*
China University of Geosciences, Wuhan 430074, P. R. China

Abstract

Differential Evolution (DE) is a simple and efficient global optimization algorithm. It has been successfully applied to solve a wide range of real-world optimization problems. However, DE has been shown to have certain weaknesses, especially if the global optimum should be located using a limited number of function evaluations (NFEs). In this paper, we incorporate the orthogonal design method into DE to accelerate its convergence rate. The orthogonal design method is not only to be used to generate the initial population, but also to be applied to design the crossover operator. In addition, two models of DE method are investigated. Moreover, the self-adaptive parameter control is employed to avoid tuning the parameters of DE. Experiments have been conducted on 25 problems of diverse complexities. And the results indicate that our approach is able to find the optimal or close-to-optimal solutions in all cases. Compared with other state-of-the-art evolutionary algorithms (EAs), our approach performs better, or at least comparably, in terms of the quality and stability of the final solutions.

Key words: Differential evolution; global optimization; orthogonal design method; self-adaptive parameter control.

1. Introduction

Over the last few decades, evolutionary algorithms (EAs) have received much attention regarding their potential as global optimization techniques [1]. Inspired from the mechanisms of natural evolution and survival of the fittest, EAs utilize a collective learning process of a population of individuals. Descendants of individuals are generated using randomized operations such as mutation and recombination. Mutation corresponds to an erroneous self-replication of individuals, while recombination exchanges information between two or more existing individuals. According to a fitness measure, the selection process favors better individuals to reproduce more often than those that are relatively worse.

Global optimization problems arise in almost every field of science, engineering, and business. Many of these problems cannot be solved analytically, and consequently, they have to be addressed by numerical algorithms. A global minimization problem can be formalized as a pair (S, f), where $S \subseteq R^n$ is a bounded set on R^n and $f : S \rightarrow R$ is an n-dimensional real-valued function. The problem is to find a point $X_{min} \in S$ such that $f(X_{min})$ is a global minimum on S. More specifically, it is required to find an $X_{min} \in S$ such that

$$\forall X \in S : f(X_{min}) \leq f(X) \tag{1}$$

where f does not need to be continuous but it must be bounded

$$l_i \leq x_i \leq u_i, i = 1, 2, \cdots, n \tag{2}$$

In global optimization problems, the major challenge is that an algorithm may be trapped in the local optima of the objective function. This issue is particularly challenging when the dimension is high and there are numerous local optima. Recently, using the EAs to solve the global optimization has been very active, producing different kinds of EAs for optimization in the continuous domain [3] - [9].

Differential evolution (DE) [10] algorithm is a novel evolutionary algorithm for faster optimization, which mutation operator is based on the distribution of solutions in the population. And DE has won the third place at the first International Contest on Evolutionary Computation on a real-valued function test-suite [11]. DE is a simple yet powerful population based, direct search algorithm with the generation-and-test feature for globally optimizing functions using real valued parameters. Among the DE's advantages are its simple structure, ease of use, speed and robustness. Price & Storn [10] gave the working principle of DE with single scheme. Later on, they suggested ten dif-

[1] Corresponding author.
E-mail address: cug11100304@yahoo.com.cn (W. Gong), zhcai@cug.edu.cn (Z. Cai)

ferent schemes of DE [11]. It has been successfully used in solving single-objective optimization problems. However, DE has been shown to have certain weaknesses, especially if the global optimum should be located using a limited number of function evaluations (NFEs). In addition, although DE is particularly simple to work with, having only a few control parameters, choice of these parameters is often critical for the performance of DE [12], [7].

To remedy these defects of the DE technique mentioned above, in this paper, an improve version of DE, namely orthogonal based DE (ODE), is presented to solve the global optimization problems. Our approach combines several features of previous EAs in a unique manner. It is characterized by a) employing the orthogonal design method with quantization technique to generate the initial population, b) nesting the orthogonal design method in crossover operator to enhance the ability of local search, c) adopting the self-adaptive parameter control method to avoid tuning the parameters of DE, and d) presenting a new model of DE to accelerate the convergence rate. The advantages of ODE are its simplicity, efficiency, and flexibility. It is shown empirically that ODE has high performance in solving benchmark functions. It can find the optimal or close-to-optimal solutions in all cases. Compared with other state-of-the-art EAs, our approach performs better, or at least comparably, in terms of the quality and stability of the final solutions.

The rest of this paper is organized as follows. In section 2, we briefly introduce the DE method. Section 3 describes the orthogonal design method used in EAs. Our proposed approach is presented in detail in section 4. In Section 5, we test our algorithm through 25 benchmark problems. In addition, the experiment results are compared with those of some state-of-the-art EAs. The last section, section 6, is devoted to conclusions and future work.

2. A brief introduction to DE

DE algorithm [10] is a simple evolutionary algorithm that creates new candidate solutions by combining the parent individual and several other individuals of the same population. A candidate replaces the parent only if it has better fitness. This is a rather greedy selection scheme that often outperforms traditional EAs. Unlike GA that uses binary coding to represent problem parameters, DE is a simple yet powerful population based, direct search algorithm with the generation-and-test feature for globally optimizing functions using real valued parameters. It has been successfully used in solving single-objective optimization problems. Among the DE's advantages are its simple structure, ease of use, speed and robustness. Due to these advantages, it has got many real-world applications, such as data mining [13], [14], pattern recognition, digital filter design, neural network training, etc. [15].

The DE algorithm in pseudo-code is shown in Algorithm 1. Where n is the number of decision variables, NP is the size of the parent population P, F is the mutation weighting factor, CR is the probability of crossover operator, U^i is the offspring, C^i is the i-th member of the offspring population C, rndint$(1, n)$ is a uniformly random integer number between 1 and n, and $\mathrm{rnd}_j[0, 1)$ is a uniformly random real number in $[0, 1)$. Many variants of creation of a candidate are possible. We use the DE scheme DE/rand/1/exp (see lines 6 and 13 of Algorithm 1) described in Algorithm 1 (more details on DE/rand/1/exp and other DE schemes can be found in [11]).

Algorithm 1 DE algorithm with DE/rand/1/exp: **model1**

1: Generate the initial population P
2: Evaluate the fitness for each individual in P
3: **while** The halting criterion is not satisfied **do**
4: **for** $i = 1$ to NP **do**
5: Select uniform randomly $r_1 \neq r_2 \neq r_3 \neq i$
6: j_{rand} = rndint$(1, n)$
7: $L = 0$
8: $U^i = P^i$
9: **repeat**
10: $U^i_j = P^{r_1}_j + F \times (P^{r_2}_j - P^{r_3}_j)$
11: $j_{rand} = (j_{rand} + 1) \bmod n$
12: $L = L + 1$
13: **until** $\mathrm{rnd}_j[0, 1) > CR$ or $L > n$
14: Evaluate the offspring U^i
15: $C^i = U^i$
16: **end for**
17: **for** $i = 1$ to NP **do**
18: **if** C^i is better than P^i **then**
19: $P^i = C^i$
20: **end if**
21: **end for**
22: **end while**

Recently, many researchers are working the improvement of DE, and many variants of DE are presented. Hybridization with other different algorithms is one direction for improvement. Fan and Lampinen [16] proposed a new version of DE which uses an additional mutation operation called trigonometric mutation operation. They showed that the modified DE algorithm can outperform the classic DE algorithm for some benchmarks and real-world problems. Sun *et al*. [17] proposed a new hybrid algorithm based on a combination of DE and estimation of distribution algorithm. This technique uses a probability model to determine promising regions in order to focus the search process on those areas. Gong *et al*. [18] employed the two level orthogonal crossover to improve the performance of DE. They show that the proposed approach performs better the classical DE in terms of the quality, speed, and stability of the final solutions. Noman and Iba [7] incorporated local search (LS) into the classical DE. They present a LS technique to solve this problem by adaptively adjusting the length of the search, using a hill-climbing heuristic. Through the experiments, they show that the proposed new version of DE performs better, or at least comparably, to classic DE algorithm.

Some other studies focus on the adaptive parameter control of DE. Liu and Lampinen [20] proposed a fuzzy adaptive differential evolution (FADE) which uses fuzzy logic controllers to adapt the mutation and crossover control parameters. Brest

et al. [5] proposed self-adapting control parameter settings. Salman *et al*. [21] proposed a self-adaptive DE (SDE) algorithm which eliminates the need for manual tuning of control parameters. In SDE, the mutation weighting factor F is self-adapted by a mutation strategy similar to the mutation operator of DE. Nobakhti and Wang [22] proposed a Randomised Adaptive Differential Evolution (RADE) method, which uses a simple randomised self-adaptive scheme is proposed for the DE mutation weighting factor F.

Most recently, Rahnamayan *et al*. [8], [9] proposed a novel initialization approach which employs opposition-based learning to generate initial population. Through a comprehensive set of benchmark functions they show that replacing the random initialization with the opposition-based population initialization in DE can accelerate convergence speed.

3. Orthogonal design method in EAs

Orthogonal design method [23] with both orthogonal array (OA) and factor analysis (such as the statistical optimal method) is developed to sample a small, but representative set of combinations for experimentation to obtain good combinations. OA is a fractional factorial array of numbers arranged in rows and columns, where each row represents the levels of factors in each combination, and each column represents a specific factor that can be changed from each combination. It can assure a balanced comparison of levels of any factor. The term "main effect" designates the effect on response variables that one can trace to a design parameter. The array is called orthogonal because all columns can be evaluated independently of one another, and the main effect of one factor does not bother the estimation of the main effect of another factor.

In a discrete single objective optimization problem, when there are N factors (variables) and each factor has Q levels, the search space consists of Q^N combinations of levels. When N and Q are large, it may not be possible to do all Q^N experiments to obtain optimal solutions. Therefore, it is desirable to sample a small, but representative set of combinations for experimentation, and based on the sample, the optimal may be estimated. The orthogonal design was developed for the purpose [23]. The selected combinations are scattered uniformly over the space of all possible combinations Q^N. And the orthogonal design is an important tool for robust design. Robust design is an engineering methodology for optimizing the product and process conditions which are minimally sensitive to the causes of variation, and which produce high-quality products with low development and manufacturing costs.

Recently, some researchers combined EAs with the orthogonal design method to solve optimization problems [18], [24] - [32]. Zhang and Leung [24] proposed incorporating orthogonal design method into the GA to solve the multimedia multicast routing problems, so that the resulting algorithm would be more robust and statistically sound. Leung and Wang [25] designed a GA called the orthogonal GA with quantization (OGA/Q) for global numerical optimization with continuous variables. They developed a quantization technique to complement the orthogonal design, so that the resulting methodology would enhance GAs for optimization with continuous variables. Ho and Chen [26] proposed an efficient EA with a novel orthogonal array crossover for obtaining the optimal solution to the polygonal approximation problem. Tsai *et al*. [30] presented a hybrid Taguchi-genetic algorithm for global numerical optimization, where the Taguchi method involving a two-level orthogonal array and a signal-to-noise-ratio is inserted between the crossover and mutation operations of a traditional GA. Gong *et al*. [18] incorporated the two level orthogonal crossover into DE method to solve the optimization problems. Wang *et al*. [32] combined EA with the orthogonal crossover operator that is the same as presented in [25] to constrained optimization problems. They show that their approach not only quickly converges to optimal or near-optimal solutions, but also displays a very high performance compared with another two state-of-the-art techniques. More recently, orthogonal design has been generalized to deal with multi-objective optimization problems [27], [29] and [31].

4. Our proposed approach

Inspired by the ideas from the orthogonal design method successfully used in EAs ([18], [24] - [32]), in this work, we propose an improved version of DE algorithm to solve global optimization problems, where our approach integrates established techniques in existing EA's in a single unique algorithm. Our proposed DE algorithm is named ODE. Six crucial procedures of ODE will be discussed as follows.

4.1. *A new model of DE*

As described in Algorithm 1, we can see that if the offspring is better than its parent, the offspring is not accepted immediately (see lines 17 and 21). In this work, we propose a new model of DE, namely model2 (the Algorithm 1 is named model1). The difference between model1 and model2 is that in model2 when the offspring is better than its parent, it is copied into the current population *immediately*. In this manner, the accepted offspring may be selected among the three solutions (r_1, r_2, r_3) and contribute to create better offspring. Hence, it can accelerate the convergence rate. The new model is described in Algorithm 2.

Algorithm 2 The new model of DE algorithm: **model2**

```
1:  Generate the initial population P
2:  Evaluate the fitness for each individual in P
3:  while The halting criterion is not satisfied do
4:    for i = 1 to NP do
5:      Select uniform randomly r1 ≠ r2 ≠ r3 ≠ i
6:      Generate the offspring U^i using DE/rand/1/exp
7:      Evaluate the offspring U^i
8:      if U^i is better than P^i then
9:        P^i = U^i {Replace the parent immediately}
10:     end if
11:   end for
12: end while
```

4.2. *Orthogonal initial population*

Before solving an optimization problem, we usually have no information about the location of the global minimum. It is desirable that an algorithm starts to explore those points that are scattered evenly in the decision space. In our presented manner, the algorithm can evenly scan the feasible solution space once to locate good points for further exploration in subsequent iterations. As the algorithm iterates and improves the population of points, some points may move closer to the global minimum. In order to generate a uniformly distributed initial population, in this work we apply the quantization technique and the orthogonal design method to generate this initial population.

4.2.1. *Design of the orthogonal array*

To design an orthogonal array (OA), in this research, we use $L_R(Q^C)$ to denote the OA with different level Q, where Q is odd and use $R = Q^J$ to indicate the number of the rows of OA, where J is a positive integer fulfilling

$$C = \frac{Q^J - 1}{Q - 1} \tag{3}$$

C denotes the number of the columns of OA in the above equation. The OA needs to find a proper J and Q to satisfy

$$\begin{aligned} &minimize: R = Q^J \\ &subject\ to: C = \frac{Q^J - 1}{Q - 1} \geq n \\ &\qquad\quad R \geq NP \end{aligned} \tag{4}$$

where n is the number of the variables, NP is the size of the evolutionary population. In this study, we adopt the algorithm described in [25] to construct an OA. In particular, we use $L(R, C)$ to indicate the OA; and $a_{i,j}$ to denote the level of the jth factor in the ith combination in $L(R, C)$. If $C > n$, we delete the last $C - n$ columns to get an OA with n factors. The algorithm to generate the OA is described in Algorithm 3.

4.2.2. *Quantization*

For one decision variable X_j with the boundary $[l_j, u_j]$, we quantize the domain into Q levels $\alpha_1^j, \alpha_2^j, \cdots, \alpha_Q^j$, where the design parameter Q is odd and α_i is given by

$$\alpha_k^j = l_j + (k-1)(\frac{u_j - l_j}{Q - 1}), 1 \leq k \leq Q \tag{5}$$

In other words, the domain $[l_j, u_j]$ is quantized $Q-1$ fractions, and any two successive levels are same as each other.

4.2.3. *Generation of Initial Population*

After constructing a proper OA and quantizing the domain of each decision variable, we can generate the orthogonal population (OP) which can scatter uniformly over the feasible solution space. The algorithm for generating the OP is described in Algorithm 4, where $OP_{i,j}$ is the j-th variable of the i-th individual of OP, n is the number of variables, and $eval$ is the current NFEs.

Algorithm 3 Construction of orthogonal array

```
/* Construct the basic columns */
for k = 1 to J do
    j = (Q^{k-1} - 1)/(Q - 1) + 1
    for i = 1 to R do
        a_{i,j} = ⌊(i-1)/Q^{J-k}⌋ mod Q
    end for
end for
/* Construct the nonbasic columns */
for k = 2 to J do
    j = (Q^{k-1} - 1)/(Q - 1) + 1
    for s = 1 to j - 1 do
        for t = 1 to Q - 1 do
            for i = 1 to R do
                a_{i,(j+(s-1)(Q-1)+t)} = (a_{i,s} × t + a_{i,j}) mod Q
            end for
        end for
    end for
end for
Increment a_{i,j} by one for all i ∈ [1, R] and j ∈ [1, C]
```

Algorithm 4 Construction of initial population

```
/* Construction of orthogonal population (OP) */
eval = 0
for i = 1 to R do
    for j = 1 to n do
        k = a_{i,j}
        OP_{i,j} = α_k^j
    end for
    Evaluate OP_i and eval++
end for
/* Construction of initial population */
Sort the OP
Select the best NP solution from OP to generate the initial population
```

4.3. *Orthogonal crossover operator*

In order to balance the exploration and exploitation, in our work we use the new crossover based on the orthogonal design and quantization technique as the local search operator to enhance the exploitation ability. The orthogonal crossover operator is original proposed in [25]; and it is also adopted in [32]. The orthogonal crossover operator acts on two parents. It quantizes the solution space defined by these parents into a finite number of points, and then applies orthogonal design to select a small, but representative sample of points as the potential offspring. The brief description of the operator is shown in Algorithm 5 (more details can be found in [25] and [32]).

4.4. *Self-adaptive parameter control*

As mentioned above, one difficulty in the use of DE arises in that the choice of its parameters is mainly based on empirical evidence and practical experience. However, the performance of DE is very sensitive to the choice of these parameters [5].

Algorithm 5 The orthogonal crossover operator

1: **Input:** Parameters $Q1$ and $C1$
2: Select the smallest $J1$ fulfilling $(Q1^{J1}-1)/(Q1-1) \geq C1$
3: Randomly select two solutions from the population
4: Quantize the domain formed by the two solutions
5: Randomly generate $C1-1$ integers $k_1, \cdots, k_{C1-1}$, such that $1 < k_1 < \cdots < k_{C1-1} < n$
6: Apply $L_{R1}(Q1^{C1})$ to generate $R1$ potential offspring
7: Select the best solution B from $R1$ offspring
8: **Output:** The best solution B

In order to avoid tuning the parameters of CR and F, in this work, a self-adaptive parameter control technique is adopted. The technique is similar to the method proposed in [5], where the self-adaptive parameter control is implemented in the individual level. The parameters CR^i and F^i are encoded in each individual P^i. Moreover, in our proposed technique each individual contains an *accepting_flag* f^i, if the offspring U^i generated by the parent P^i is accepted, $f^i = 1$; otherwise, $f^i = 0$. Our proposed self-adaptive parameter control technique is introduced as follows

$$F^i = \begin{cases} \text{rnd}[0.1, 1], \ \text{rnd}[0,1] < \tau_1 \text{ and } f^i == 0 \\ F^i, \qquad \text{otherwise} \end{cases} \tag{6}$$

$$CR^i = \begin{cases} N(0.9, 0.05), \ \text{rnd}[0,1] < \tau_2 \text{ and } f^i == 0 \\ CR^i, \qquad \text{otherwise} \end{cases} \tag{7}$$

where $\text{rnd}[a, b]$ is the uniform random variable between a and b. $N(0.9, 0.05)$ is a normal distribution. τ_1 and τ_2 indicate probabilities to adjust factors F^i and CR^i.

The rationale behind using a normal distribution $N(0.9, 0.05)$ for CR^i is that $N(0.9, 0.05)$ will generate values in the range of $[0.9 - 3 \times 0.05, 0.9 + 3 \times 0.05]$ which gives more probability to values surrounding 0.9. When $CR^i > 1.0$, reset it to 1.0. The reason for preferring values surrounding 0.9 is that $CR^i = 0.9$ can obtain *good* results for many problems [19].

The main difference between our proposed self-adaptive parameter control technique and the method proposed in [5] is that in our technique the accepting_flag f^i is encoded in each individual. The reason is that if $f^i == 1$ means the parent can generate a *good* offspring with its factors F^i and CR^i, hence these factors can be used in the next generation. In this manner, our approach can enhance the ability of the self-adaptive parameter control. To verify this improvement, we will make addition experiments to compare our method with the method used in [5] in the next section 5.6.

4.5. *Handling the boundary constraint of variables*

After using the DE/rand/1/exp scheme to generate a new solution, if one or more of the variables in the new solution are outside their boundaries, i.e. $x_i \notin [l_i, u_i]$, the following repair rule is applied:

$$x_i = \begin{cases} l_i + \text{rnd}_i[0,1] \times (u_i - l_i) & \text{if } x_i < l_i \\ u_i - \text{rnd}_i[0,1] \times (u_i - l_i) & \text{if } x_i > u_i \end{cases} \tag{8}$$

where $\text{rnd}_i[0, 1]$ is the uniform random variable from [0,1] in each dimension i.

4.6. *Main procedure of ODE*

For the global optimization problems, the proposed ODE works similar to the model2 described in Algorithm 2. The differences are: i) the initial population of ODE is generated using Algorithm 4, ii) the orthogonal crossover operator is employed to enhance the local search ability; and iii) the factors CR^i and F^i are calculated self-adaptively. The main procedure is introduced in Algorithm 6.

Algorithm 6 Main procedure of our approach: **ODE-model2**

1: Generate the initial population P using Algorithm 4
2: Generate the OA for orthogonal crossover
3: **while** The halting criterion is not satisfied **do**
4: **for** $i = 1$ to NP **do**
5: Select uniform randomly $r_1 \neq r_2 \neq r_3 \neq i$
6: Calculate F^i and CR^i as shown in Eqn. (6) and Eqn. (7)
7: Generate the offspring U^i using DE/rand/1/exp
8: Evaluate the offspring U^i
9: **if** U^i is better than P^i **then**
10: $P^i = U^i$ {Replace the parent *immediately*}
11: **end if**
12: **end for**
13: Adopt the orthogonal crossover to generate an offspring B (see Algorithm 5)
14: Randomly select a solution A from the population P
15: **if** B is better than A **then**
16: $A = B$
17: **end if**
18: **end while**

5. Experiments

In order to validate the performance of ODE, we have carried out different experiments using the test suit described in Table 1. The test suite consists of 25 unconstrained single-objective benchmark functions with different characteristics chosen from the literature. All of the functions are **minimization** problems. A more detailed description of each function is given in [2] and [25], where the functions were divided into three classes: functions with no local minima, many local minima, and a few local minima. If the number of test problems were smaller, it would be very difficult to make a general conclusion. Using a test set which is too small also has the potential risk that the algorithm is biased (optimized) toward the chosen set of problems. Such bias might not be useful for other problems of interest.

Functions $f_{01} - f_{13}$, f_{24} and f_{25} are high-dimensional problems. Functions $f_{01} - f_{05}$ are unimodal. Function f_{06} is the

Table 1
The 25 benchmark functions used in our experimental study, where n is the number of variables, "optimal" is the minimum value of the function, and $S \subseteq R^n$. A detail description of all functions can be found in [2] and [25].

Test Functions	n	S	optimal
$f_{01} = \sum_{i=1}^{n} x_i^2$	30	$[-100, 100]^n$	0
$f_{02} = \sum_{i=1}^{n} \lvert x_i \rvert + \prod_{i=1}^{n} \lvert x_i \rvert$	30	$[-10, 10]^n$	0
$f_{03} = \sum_{i=1}^{n} (\sum_{j=1}^{i} x_j)^2$	30	$[-100, 100]^n$	0
$f_{04} = \max_i \{\lvert x_i \rvert, 1 \le i \le n\}$	30	$[-100, 100]^n$	0
$f_{05} = \sum_{i=1}^{n-1} [100(x_{i+1} - x_i^2)^2 + (x_i - 1)^2]$	30	$[-30, 30]^n$	0
$f_{06} = \sum_{i=1}^{n-1} (\lfloor x_i + 0.5 \rfloor)^2$	30	$[-100, 100]^n$	0
$f_{07} = \sum_{i=1}^{n} x_i^4 + random[0, 1)$	30	$[-1.28, 1.28]^n$	0
$f_{08} = \sum_{i=1}^{n} (-x_i \sin(\sqrt{\lvert x_i \rvert}))$	30	$[-500, 500]^n$	-12569.48662
$f_{09} = \sum_{i=1}^{n} (x_i^2 - 10\cos(2\pi x_i) + 10)$	30	$[-5.12, 5.12]^n$	0
$f_{10} = -20\exp(-0.2\sqrt{\frac{1}{n}\sum_{i=1}^{n} x_i^2}) - \exp(\frac{1}{n}\sum_{i=1}^{n}\cos(2\pi x_i)) + 20 + \exp(1)$	30	$[-32, 32]^n$	0
$f_{11} = \frac{1}{4000}\sum_{i=1}^{n} x_i^2 - \prod_{i=1}^{n}\cos(\frac{x_i}{\sqrt{i}}) + 1$	30	$[-600, 600]^n$	0
$f_{12} = \frac{\pi}{n}\{10\sin^2(\pi y_i) + \sum_{i=1}^{n-1}(y_i - 1)^2 \cdot [1 + 10\sin^2(\pi y_{i+1})] + (y_n - 1)^2\} + \sum_{i=1}^{n} u(x_i, 10, 100, 4)$	30	$[-50, 50]^n$	0
$f_{13} = \frac{1}{10}\{\sin^2(3\pi x_1) + \sum_{i=1}^{n-1}(x_i - 1)^2[1 + \sin^2(3\pi x_{i+1})] + (x_n - 1)^2[1 + \sin^2(2\pi x_n)]\} + \sum_{i=1}^{n} u(x_i, 5, 100, 4)$	30	$[-50, 50]^n$	0
$f_{14} = \left[\frac{1}{500} + \sum_{j=1}^{25}\frac{1}{j + \sum_{i=1}^{2}(x_i - a_{ij})^6}\right]^{-1}$	2	$[-65.536, 65.536]^n$	0.998
$f_{15} = \sum_{i=1}^{11}\left[a_i - \frac{x_1(b_i^2 + b_i x_2)}{b_i^2 + b_i x_3 + x_4}\right]^2$	4	$[-5, 5]^n$	0.003075
$f_{16} = 4x_1^2 - 2.1x_1^4 + \frac{1}{3}x_1^6 + x_1x_2 - 4x_2^2 + 4x_2^4$	2	$[-5, 5]^n$	-1.0316285
$f_{17} = (x_2 - \frac{5.1}{4\pi^2}x_1^2 + \frac{5}{\pi}x_1 - 6)^2 + 10(1 - \frac{1}{8\pi})\cos x_1 + 10$	2	$[-5, 10] \times [0, 15]$	0.398
$f_{18} = [1 + (x_1 + x_2 + 1)^2(19 - 14x_1 + 3x_1^2 - 14x_2 + 6x_1x_2 + 3x_2^2)] \times [30 + (2x_1 - 3x_2)^2(18 - 32x_1 + 12x_1^2 + 48x_2 - 36x_1x_2 + 27x_2^2)]$	2	$[0, 1]^n$	3
$f_{19} = -\sum_{i=1}^{4} c_i \exp[-\sum_{j=1}^{n} a_{ij}(x_j - p_{ij})^2]$	3	$[0, 1]^n$	-3.86
$f_{20} = -\sum_{i=1}^{4} c_i \exp[-\sum_{j=1}^{n} a_{ij}(x_j - p_{ij})^2]$	6	$[0, 1]^n$	-3.32
$f_{21} = -\sum_{i=1}^{5}[(x - a_i)(x - a_i)^T + c_i]^{-1}$	4	$[0, 10]^n$	-10.1532
$f_{22} = -\sum_{i=1}^{7}[(x - a_i)(x - a_i)^T + c_i]^{-1}$	4	$[0, 10]^n$	-10.40294
$f_{23} = -\sum_{i=1}^{10}[(x - a_i)(x - a_i)^T + c_i]^{-1}$	4	$[0, 10]^n$	-10.53641
$f_{24} = -\sum_{i=1}^{n}\sin x_i \sin^{20}(\frac{i \times x_i^2}{\pi})$	100	$[0, \pi]^n$	-99.2784
$f_{25} = \frac{1}{n}\sum_{i=1}^{n}(x_i^4 - 16x_i^2 + 5x_i)$	100	$[-5, 5]^n$	-78.33636

Table 2
Comparison with ODE/2, ODE/1, DE/1 and DE/2 on $f_{01} - f_{25}$. A result in **Boldface** indicates that a better result obtained, hereinafter.

F	Number of successful runs				Mean NFEs of successful runs				Mean time of successful runs (in *s*)			
	ODE/1	ODE/2	DE/1	DE/2	ODE/1	ODE/2	DE/1	DE/2	ODE/1	ODE/2	DE/1	DE/2
f_{01}	50	50	50	50	45,758	**35,235**	67,066	53,548	0.61438	0.37002	0.74070	**0.30656**
f_{02}	50	50	50	50	49,370	**36,914**	67,006	45,912	0.76460	0.49724	0.93606	**0.39628**
f_{03}	50	50	50	50	120,197	**95,520**	203,232	144,076	2.04940	**1.15154**	5.44628	1.52286
f_{04}	50	50	50	50	144,562	**126,731**	292,802	189,680	1.97096	**1.16576**	3.24654	1.55036
f_{05}	50	50	50	49	249,288	**232,171**	264,804	236,808	3.04561	1.58274	2.58126	**1.55169**
f_{06}	50	50	50	50	26,559	**20,051**	39,106	30,286	0.51590	0.34658	0.53066	**0.25436**
f_{07}	50	50	18	46	87,545	**83,072**	393,190	328,491	1.51686	**1.11970**	6.29418	4.21772
f_{08}	50	50	50	50	50,880	**42,346**	116,952	955,90	0.89660	**0.61996**	1.67410	0.98034
f_{09}	50	50	50	50	68,967	**63,763**	208,942	168,732	0.99896	**0.67624**	2.50854	1.29378
f_{10}	50	50	50	50	48,200	**36,802**	68,136	53,784	0.81238	0.50188	0.94564	**0.45938**
f_{11}	50	50	50	50	44,269	**34,010**	66,618	51,602	0.76216	0.48592	0.88414	**0.46190**
f_{12}	50	50	50	50	31,139	**23,648**	47,790	36,290	1.11812	**0.76506**	1.46412	0.89940
f_{13}	50	50	50	50	43,807	**33,409**	67,020	50,236	1.42348	**0.97246**	1.93056	1.18938
f_{14}	50	50	50	50	3,707	**3,383**	4,186	3,702	0.07462	0.05970	0.06934	**0.04788**
f_{15}	50	50	50	50	1,163	1,124	1,144	**946**	0.02562	0.02064	0.00878	**0.00464**
f_{16}	50	50	50	50	1,179	1,016	1,154	**998**	0.02246	0.01818	0.00722	**0.00248**
f_{17}	50	50	50	50	1,744	1,584	1,688	**1,356**	0.02316	0.01840	0.01064	**0.00312**
f_{18}	50	50	50	50	1,896	1,621	1,928	**1,556**	0.02750	0.02056	0.01278	**0.00532**
f_{19}	50	50	50	50	1,192	**946**	1,132	1,038	0.02186	0.01938	0.00784	**0.00378**
f_{20}	50	50	50	50	5,244	**4,059**	5,354	14,504	0.06447	**0.03842**	0.04780	0.07564
f_{21}	50	50	50	50	7,132	**5,473**	7,234	5,918	0.06756	0.03648	0.05742	**0.02374**
f_{22}	50	50	50	50	6,263	**5,053**	6,258	5,066	0.06064	0.03726	0.04842	**0.02062**
f_{23}	50	50	50	50	6,355	**4,782**	6,352	5,184	0.06408	0.03592	0.05156	**0.02432**
f_{24}	0	0	0	0	–	–	–	–	–	–	–	–
f_{25}	50	50	50	50	52,085	**47,940**	171,053	120,130	5.14173	4.16752	5.63183	**3.27023**
Σ	**1,200**	**1,200**	1,168	1,195	1,098,501	**940,653**	2,110,147	1,645,433	22.08311	**14.72756**	35.13641	18.56578

step function which has one minimum and is discontinuous. Function f_{07} is a noisy quadratic function. Functions $f_{08} - f_{13}$, f_{24} and f_{25} are multimodal functions where the number of local minima increases exponentially with the problem dimension [2], [25]. For f_{24} as an example, if the solution is to be accurate up to two decimal places, there are $(\pi/0.01)^{100} = 5.19 \times 10^{249}$ possible points in the feasible solution space, and there are $100! = 9.33 \times 10^{157}$ local minima. Functions $f_{14} - f_{23}$ are low-dimensional functions which have only a few local minima.

5.1. *Experimental setup*

For all problems, we used the following parameters [2]:
- Population size: *NP* = 100;
- Parameters of orthogonal initial population: To generate a minimal OA, we use $J = 2$, if $n > 11, Q = n-1$; otherwise, $Q = 11$;
- Parameters of orthogonal crossover: $Q1 = 3$, $C1 = 4$. Consequently, $J1$ can be found to be 2;
- $\tau_1 = \tau_2 = 0.1$.
- Halting criterion: When the maximum of NFEs (Max_eval) is reached, the execution of the algorithm is stopped. And the Max_eval is different for different problems.

All the algorithms are implemented in C++ and the experiments are done on a P-IV 3.0 GHz machine with 512 MB RAM under WIN-XP platform. For each function 50 independent runs with different random seeds are taken. It is worth to note that since the initial population of ODE is generated by Algorithm 4, for the same function the initial population is the same for all 50 runs.

5.2. *Comparison of ODE/2 with ODE/1, DE/1 and DE/2*

The focus of this subsection was to compare the performance of the proposed model2 of DE with the original model1. Moreover, the performance of orthogonal design method used in DE is also verified. Where ODE/2 is introduced in Algorithm 6. ODE/1 is similar to ODE/2, the only difference is that in ODE/1 the offspring is not accepted immediately (see Algorithm 1). For DE/1 and DE/2, the orthogonal initial population and orthogonal crossover operator are not used. Note that the self-adaptive parameter control method is still adopted for DE/1 and DE/2. For each DE 50 independent runs with different random seeds are taken and the number of successful runs (out of 50 runs), mean NFEs and execution time of success-

[2] For ODE, we have chosen a reasonable set of value and have not made any effort in finding the best parameter settings. We leave this task for a future study.

Table 3
Comparison of the quality of the final results between ODE/2 and DE/2 on $f_{01} - f_{25}$.

F	Max_eval	MNFEs		Mean Best		Std		Optimal
		ODE/2	DE/2	ODE/2	DE/2	ODE/2	DE/2	
f_{01}	150,000	150,000	150,000	**2.06E-23**	1.64E-18	**1.83E-23**	5.29E-18	0
f_{02}	200,000	200,000	200,000	**1.43E-18**	2.97E-15	**8.11E-19**	5.78E-15	0
f_{03}	500,000	500,000	500,000	**5.25E-27**	3.53E-20	**9.66E-27**	3.53E-20	0
f_{04}	500,000	500,000	500,000	**2.72E-15**	9.73E-10	**9.30E-15**	5.00E-10	0
f_{05}	500,000	**428,776**	494,788	**0**	2.55E-29	**0**	1.15E-28	0
f_{06}	150,000	**22,640**	30,454	0	0	0	0	0
f_{07}	300,000	300,000	300,000	**0.00145**	0.00598	**4.20E-04**	0.00125	0
f_{08}	300,000	**90,381**	167,324	-12569.48662	-12569.48662	0	0	-12596.48662
f_{09}	300,000	**127,666**	247,626	0	0	0	0	0
f_{10}	150,000	150,000	150,000	**4.67E-13**	3.19E-10	**1.86E-13**	1.10E-10	0
f_{11}	200,000	**109,853**	138,236	0	0	0	0	0
f_{12}	150,000	150,000	150,000	**6.73E-26**	4.99E-20	**9.27E-26**	3.68E-20	0
f_{13}	150,000	150,000	150,000	**4.37E-24**	4.42E-18	**3.67E-24**	4.66E-18	0
f_{14}	10,000	**9,552**	9,796	0.998	0.998	**0**	7.92E-15	0.998
f_{15}	150,000	**32,430**	34,484	3.08E-04	3.08E-04	0	0	0.0003075
f_{16}	10,000	10,000	10,000	-1.03163	-1.03163	**0**	9.16E-14	-1.0316825
f_{17}	10,000	10,000	10,000	0.39789	0.39789	2.01E-10	**6.35E-11**	0.39789
f_{18}	10,000	10,000	10,000	3	3	**0**	1.34E-14	3
f_{19}	10,000	10,000	10,000	-3.86278	-3.86278	2.68E-15	2.68E-15	-3.86
f_{20}	20,000	20,000	20,000	**-3.322**	-3.31962	**1.13E-12**	0.01681	-3.32
f_{21}	10,000	10,000	10,000	-10.1532	-10.1532	**1.04E-06**	1.29E-05	-10.1532
f_{22}	10,000	10,000	10,000	-10.40294	-10.40294	**2.49E-08**	5.84E-08	-10.40294
f_{23}	10,000	10,000	10,000	-10.53641	-10.53641	**2.35E-08**	5.80E-08	-10.53641
f_{24}	500,000	500,000	500,000	**-97.93352**	-80.61412	**0.35098**	0.80576	-99.2784
f_{25}	500,000	**272,188**	500,000	**-78.33233**	-78.33233	**0**	4.29E-14	-78.33236

ful runs are recorded. For a particular problem and a particular algorithm, a run is said to be a successful run if the best objective function value found in that run lies within 0.5%, i.e. $|f(X_{best}) - f(X_{optimal})| \leq 0.005$, accuracy of the best know objective function value of that problem. The maximal NFEs (Max_eval) are also fixed to be 500,000 for all DEs. The results are list in Table 2. From this table it can be observed that except function f_{24} both ODE/1 and ODE/2 solve all problems in all 50 runs. For function f_{07}, DE/1 and DE/2 can not solve this problem in all 50 runs. For function f_{05}, DE/2 traps in the local optima once. In 20 test functions the mean NFEs of successful runs required by ODE/2 are the least among ODE/1, DE/1 and DE/2. Moreover, the overall mean NFEs of 24 functions required by ODE/2 are least. For DE/2, it requires the least mean NFEs of successful runs in 4 functions. With respect to the mean time of successful runs, DE/2 requires the least time in 16 functions whereas ODE/2 requires least time in 8 functions.

Compared the performance of model1 with model2 (i.e. ODE/1 vs. ODE/2 and DE/1 vs. DE/2), ODE/2 (DE/2) is better than ODE/1 (DE/1) in terms of the three criteria abovementioned.

To verify the performance of orthogonal design method used in DE, from table 2 we can see that ODE/1 (ODE/2) is better than DE/1 (DE/2) in terms of the overall mean NFEs. However, the mean time required by ODE/1 and ODE/2 are more than DE/1 and DE/2. The reason is that in our implementation the OA is generated in each run. As mentioned above for ODE the initial population is same in each run, so in order to reduce the time-consuming we need not to generate the OA in each run. Because we focus on enhancing the robust and accelerating the convergence rate of DE in this work, the time-consuming of construction of OA is not considered.

In view of the above discussions we conclude that i) the performance of model2 is better than that of model1; ii) the orthogonal design method can enhance the performance of DE; iii) our proposed ODE/2 outperforms ODE/1, DE/1 and DE/2 in terms of the overall performance. Hence in the following experiments we only consider ODE/2 and DE/2.

5.3. *Comparison of the quality of the final results between ODE/2 and DE/2*

In this experiment, we compare the quality of the final results obtained by ODE/2 and DE/2. We set the parameters as described in the subsection 5.1. In order to compare the quality of the solutions found by ODE/2 and DE/2, we list: i) the mean NFEs ("MNFEs"), ii) the mean best function value ("Mean Best"), and iii) the standard deviation of the function values ("Std"). The mean results of 50 independent runs are summarized in Table 3. Furthermore, some representative graphs comparing the convergence characteristics of ODE/2 with DE/2 are shown in Fig. 1.

From Table 3, we see that the mean function values found by ODE/2 are equal or close to the optimal ones, and the standard deviations of the function values are relatively small. These results indicate that ODE/2 can find optimal or close-to-optimal solutions, and its solution quality is quite stable.

Moreover, Table 3 compares the results between ODE/2 and DE/2 to show the performance of the orthogonal design method. Recall that DE/2 is the same as ODE/2, except that it uses random sampling instead of orthogonal sampling. From this table it can be observed that ODE/2 obtain better or the same results compared with DE/2 in terms of mean NFEs, mean function values and the standard deviation of function values. In 8 functions ODE/2 requires less function evaluations than DE/2. In addition, ODE/2 gives smaller mean function values than DE/2 in 12 functions. Furthermore, ODE/2 obtains smaller standard deviations of function values than DE/2 in 18 functions. Also, from Fig. 1 we can see that the convergence rate of ODE/2 is faster than DE/2, especially for the high-dimensional problems. These results demonstrate that orthogonal design method used in DE can effectively enhance the performance of DE. In addition, from Fig. 1 it can be seen that the best solution of the initial population of ODE/2 is closer to the global optimum (e.g. f_{08}, f_{19} and f_{25}) after using the orthogonal initial population, and hence, it can be further explored in the subsequent iterations.

5.4. *Comparison of ODE/2 with FEP and CEP*

In the experiment, we compare the performance of ODE/2 with FEP and CEP. The mean results of 50 independent runs are summarized in Table 4. Results for the FEP and CEP algorithms are taken from [2].

From Table 4, it is clear to see that ODE/2 obtains better results on the 23 benchmark functions than FEP and CEP. ODE/2 gives smaller mean function values than FEP and CEP, and hence its mean solution quality is better. In addition, ODE/2 is able to obtain smaller standard deviations of function values, it means that the solution quality of ODE/2 is more stable than FEP and CEP. Moreover, in 7 functions ODE/2 requires less mean NFEs.

5.5. *Comparison of ODE/2 with OGA/Q, HTGA, LEA and HTGA/T*

In this subsection the performance of ODE/2 is compared with four EAs (OGA/Q [25], HTGA [30], LEA [6], and HTGA/T [6]), which also incorporate the experiment design method into EAs. The four algorithms are briefly described as follows.

- The orthogonal genetic algorithm with quantization (OGA/Q) [25]: OGA/Q uses an orthogonal design to generate the initial population. And the orthogonal design was used to construct a crossover operator.
- The hybrid Taguchi-genetic algorithm (HTGA) [30]: HTGA combines the Taguchi method into a genetic algorithm. The Taguchi method is a robust experimental design approach that uses both the orthogonal design method and the SNR.
- The level-set evolution and Latin squares based EA (LEA) [6]: LEA combines the level-set evolution and Latin squares with EA. The Latin squares were used to generate the initial population and to design the crossover operator.
- HTGA without the Taguchi method (HTGA/T) [6]: HTGA/T is the same as HTGA, only expect the Taguchi method is removed from HTGA.

The mean results of 50 independent runs are summarized in Table 5. Results for the OGA/Q, HTGA, LEA and HTGA/T algorithms are taken from [25], [30], [6] and [6], representatively. From this table it can be seen that each of ODE/2, OGA/Q, HTGA and LEA is able to find the optimal or close-to-optimal solution with small standard deviations for these 13 functions. HTGA/T performs worst among the five algorithms. It can not find the optimal solutions for all 13 functions. And the standard deviations of HTGA/T are larger than ODE/2, OGA/Q, HTGA and LEA. For five functions (f_{08}, f_{12}, f_{13}, f_{24} and f_{25}), ODE/2 can provide the better mean function values than OGA/Q, HTGA, LEA and HTGA/T. In addition, the mean NFEs required by ODE/2 are smaller than them for the five functions. ODE/2 performs a little better than OGA/Q and LEA, especially for the functions with many local minima. ODE/2 performs worse than HTGA for eight out of 13 functions. Note that the number of generations used by HTGA was much smaller than 1000 and the solutions could not be further improved by more generations. Thus, although the mean number of function evaluations used by HTGA is smaller than that used by ODE/2, ODE/2 has the stronger ability to find better solutions than HTGA for higher dimensional problems. These results also indicate the Taguchi method can effectively improve the performance of the genetic algorithm, especially for 30 or lower dimensional problems [6]. For example, for functions f_{24} and f_{25}, which are 100-dimensional problems, ODE/2 can provide better results with smaller mean NFEs than HTGA. Moreover, ODE/2 performs remarkably better than HTGA/T, where the Taguchi method is removed from HTGA.

5.6. *The accepting_flag for self-adaptive DE*

In the experiment, we test the performance of the accepting_flag for self-adaptive DE. To verify the performance of the accepting_flag, the accepting_flag is removed from ODE/2, and the resulting algorithm, denoted as ODE/2/a, is executed to compare with ODE/2. The mean results of 50 independent runs are summarized in Table 6 for eight selected respective functions (f_{01}, f_{05}, f_{08}, f_{13}, f_{19}, f_{21}, f_{24} and f_{25}).

From Table 6 it can be observed that ODE/2 is a little better than ODE/2/a in terms of the mean NFEs, mean best function values and the standard deviations of function values. These results indicate that the accept_flag encoded in each individual can improve the performance of DE.

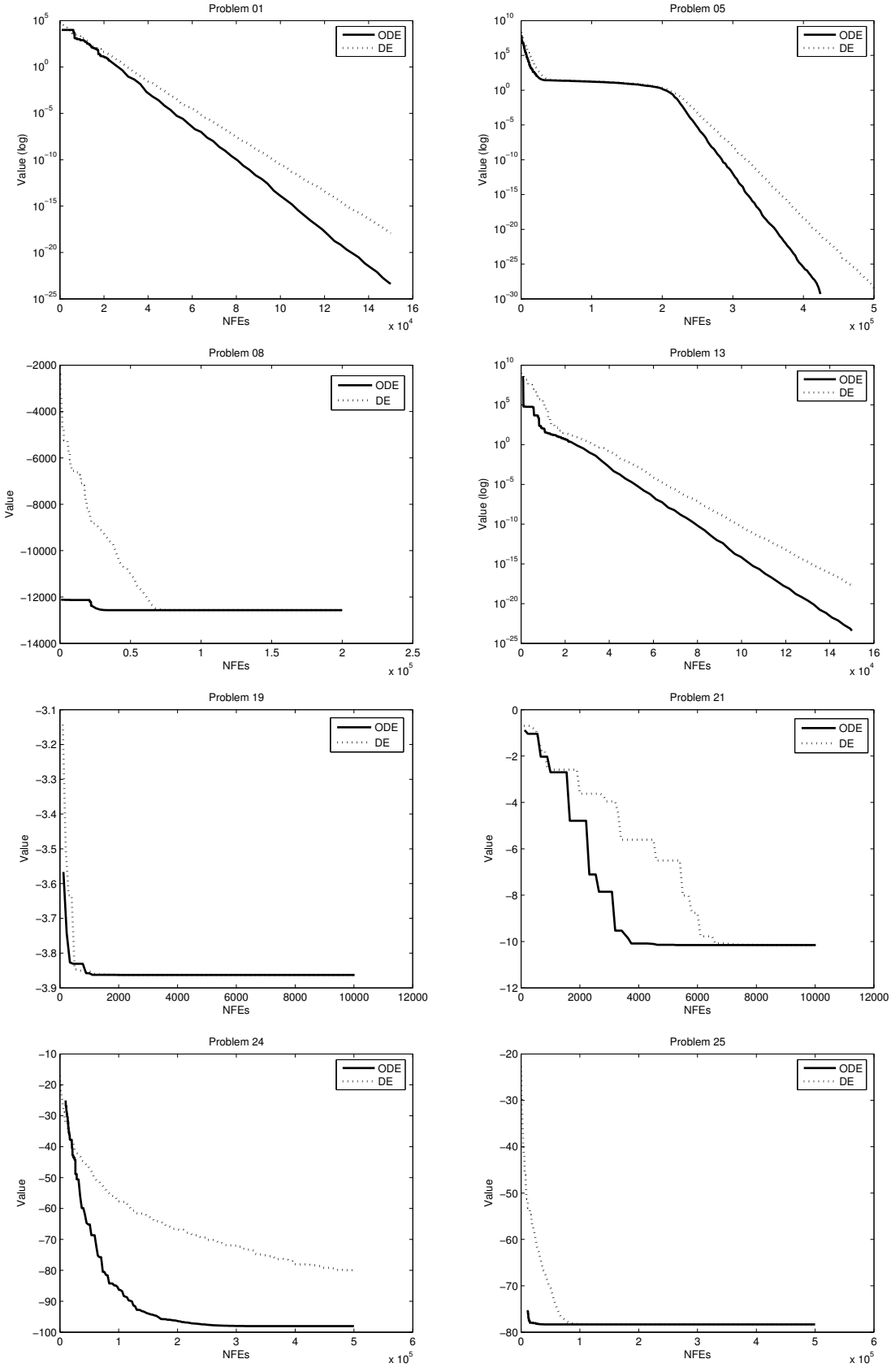

Fig. 1. Convergence curves of ODE/2 and DE/2 algorithm for selected functions. X axis represents number of function evaluations (NFEs) and Y axis represents function values.

Table 4
Comparison of the performance of ODE/2 with FEP and CEP on $f_{01} - f_{23}$.

F	MNFEs			Mean Best			Std		
	ODE/2	FEP	CEP	ODE/2	FEP	CEP	ODE/2	FEP	CEP
f_{01}	150,000	150,000	150,000	**2.06E-23**	5.70E-4	2.20E-4	**1.83E-23**	1.30E-4	5.90E-4
f_{02}	200,000	200,000	200,000	**1.43E-18**	8.10E-3	2.60E-3	**8.11E-19**	7.70E-4	1.70E-4
f_{03}	500,000	500,000	500,000	**5.25E-27**	1.60E-2	5.00E-2	**9.66E-27**	1.40E-2	6.60E-2
f_{04}	500,000	500,000	500,000	**2.72E-15**	0.3	2.0	**9.30E-15**	0.5	1.2
f_{05}	**428,776**	2,000,000	2,000,000	**0**	5.06	6.17	**0**	5.87	13.61
f_{06}	**22,640**	150,000	150,000	**0**	**0**	577.76	**0**	**0**	1125.76
f_{07}	300,000	300,000	300,000	**0.00145**	0.00760	0.01800	**4.20E-04**	2.60E-3	6.40E-3
f_{08}	**90,381**	900,000	900,000	**-12569.48662**	-12554.5	-7917.1	**0**	52.6	634.5
f_{09}	**127,666**	500,000	500,000	**0**	4.60E-2	89.0	**0**	1.20E-2	23.1
f_{10}	150,000	150,000	150,000	**4.67E-13**	1.80E-2	9.2	**1.86E-13**	2.1E-3	2.8
f_{11}	**109,853**	200,000	200,000	**0**	1.60E-2	8.60E-2	**0**	2.20E-2	0.12
f_{12}	150,000	150,000	150,000	**6.73E-26**	9.20E-6	1.76	**9.27E-26**	3.60E-6	2.4
f_{13}	150,000	150,000	150,000	**4.37E-24**	1.60E-4	1.4	**3.67E-24**	7.30E-5	3.7
f_{14}	**9,552**	10,000	10,000	**0.998**	1.22	1.66	**0**	0.56	1.19
f_{15}	**32,430**	400,000	400,000	**3.08E-04**	5.00E-4	4.70E-4	**0**	3.20E-4	3.00E-4
f_{16}	10,000	10,000	10,000	**-1.03163**	-1.03	-1.03	**0**	4.90E-7	4.90E-7
f_{17}	10,000	10,000	10,000	**0.39789**	0.398	0.398	**2.01E-10**	1.50E-7	1.50E-7
f_{18}	10,000	10,000	10,000	**3.00**	3.02	**3.00**	**0**	0.11	0
f_{19}	10,000	10,000	10,000	**-3.86278**	-3.86	-3.86	**2.68E-15**	1.40E-5	1.40E-2
f_{20}	20,000	20,000	20,000	**-3.322**	-3.27	-3.28	**1.13E-12**	5.90E-2	5.80E-2
f_{21}	10,000	10,000	10,000	**-10.1532**	-5.52	-6.86	**1.04E-06**	1.59	2.67
f_{22}	10,000	10,000	10,000	**-10.40294**	-5.52	-8.27	**2.49E-08**	2.12	2.95
f_{23}	10,000	10,000	10,000	**-10.53641**	-6.57	-9.10	**2.35E-08**	3.14	2.92

5.7. *Discussions of the experimental results*

The comparison results in Table 2- 6 can be summarized as follows.

– The proposed model2 of DE is better than the original model1 in terms of the average NFEs and average execution time. Model2 can reduce the computational complexity and accelerate the convergence rate.
– The orthogonal design method can enhance the performance of DE. It can remarkably accelerate the convergence rate for the high-dimensional problems.
– Our approach, ODE/2, can obtain optimal or close-to-optimal solutions with small standard deviations for all the 25 test functions.
– ODE/2 performs better than CEP, FEP, and HTGA/T. In addition, it performs a little better than OGA/Q and LEA, especially for the functions with many local minima. Moreover, ODE/2 can find better solutions than HTGA on 100-dimensional problems f_{24} and f_{25}.
– The accept_flag encoded in each individual can improve the performance of DE.

6. Conclusions

In this paper, we have presented a improved DE algorithm to solve the global optimization problems with continuous variables. We combine the orthogonal design method into DE to accelerate its convergence rate. Moreover, an improved model of DE is proposed to reduce the computational complexity of the original DE. To avoid tuning the factors of DE, the self-adaptive parameter control technique is adopted in our approach. We executed the proposed algorithm to solve 25 benchmark problems with high or low dimensions, where some of these problems have numerous local minima. The computational experiments show that the proposed ODE/2 can find optimal or close-to-optimal solutions, solutions, and it is more competitive than almost all of the compared algorithms for these test problems.

Our future work consists on investigating the effect of different parameter settings on the performance of our approach, and adding a diversity mechanism in ODE to solve the constrained optimization problems.

Table 6
Comparison of ODE/2 with ODE/2/a on eight functions.

F	MNFEs		Mean Best		Std	
	ODE/2	ODE/2/a	ODE/2	ODE/2/a	ODE/2	ODE/2/a
f_{01}	150,000	150,000	**2.06E-23**	1.15E-22	**1.83E-23**	5.36E-23
f_{05}	**428,776**	431,359	0	0	0	0
f_{08}	**90,381**	92,371	-12569.49	-12569.49	0	0
f_{13}	150,000	150,000	**4.37E-24**	1.18E-23	3.67E-24	**1.45E-24**
f_{19}	10,000	10,000	-3.86278	-3.86278	**2.68E-15**	2.80E-15
f_{21}	10,000	10,000	-10.1532	-10.1532	**1.04E-06**	1.95E-6
f_{24}	500,000	500,000	**-97.93352**	-97.85130	**0.1510**	0.2658
f_{25}	**272,188**	273,955	-78.33233	-78.33233	0	0

Table 5
Comparison of ODE/2 with OGA/Q, HTGA, LEA and HTGA/T.

F	Algorithms	MNFEs	Mean Best	Std	F	Algorithms	MNFEs	Mean Best	Std
	ODE/2	100,000	2.241E-16	8.918E-18		ODE/2	150,000	4.670E-13	1.860E-13
	OGA/Q	112,559	**0**	**0**		OGA/Q	112,421	4.440E-16	3.989E-17
f_{01}	HTGA	**20,844**	**0**	**0**	f_{10}	HTGA	**16,632**	**0**	**0**
	LEA	110,674	4.727E-16	6.218E-17		LEA	105,926	3.274E-16	3.0OlE-17
	HTGA/T	153,109	0.079182	0.055237		HTGA/T	183,107	0.07235	0.09174
	ODE/2	100,000	8.120E-16	7.326E-17		ODE/2	109,853	**0**	**0**
	OGA/Q	112,612	**0**	**0**		OGA/Q	134,000	**0**	**0**
f_{02}	HTGA	**14,285**	**0**	**0**	f_{11}	HTGA	**20,999**	**0**	**0**
	LEA	110,031	4.247E-19	4.236E-19		LEA	130,498	6.104E-16	2.513E-17
	HTGA/T	163,372	0.07153	0.08218		HTGA/T	200,125	0.02998	0.03897
	ODE/2	100,000	5.3965E-12	5.366E-14		ODE/2	**50,000**	**2.996E-7**	7.203E-7
	OGA/Q	112,576	**0**	**0**		OGA/Q	134,556	6.019E-6	1.159E-6
f_{03}	HTGA	**26,469**	**0**	**0**	f_{12}	HTGA	66,457	1.000E-6	**0**
	LEA	110,604	6.783E-18	5.429E-18		LEA	132,642	2.482E-6	2.276E-6
	HTGA/T	166,108	0.06164	0.05216		HTGA/T	200,125	0.02998	0.03897
	ODE/2	100,000	1.9561E-7	1.3306E-7		ODE/2	**50,000**	**6.962E-6**	3.586E-6
	OGA/Q	112,893	**0**	**0**		OGA/Q	134,143	1.869E-4	2.615E-5
f_{04}	HTGA	**21,261**	**0**	**0**	f_{13}	HTGA	59,003	1.000E-4	**0**
	LEA	111,105	2.683E-16	6.257E-17		LEA	130,213	1.734E-4	1.205E-4
	HTGA/T	166,572	0.02407	0.04195		HTGA/T	200,356	0.04487	0.04326
	ODE/2	100,000	2.384E-3	3.326E-4		ODE/2	**150,000**	**-94.27**	2.248E-3
	OGA/Q	112,652	6.301E-3	4.069E-4		OGA/Q	302,773	-92.83	2.626E-2
f_{07}	HTGA	**20,844**	**1.000E-3**	**0**	f_{24}	HTGA	265,693	-92.83	**0**
	LEA	111,093	5.136E-3	4.432E-4		LEA	289,863	-93.01	0.02314
	HTGA/T	160,804	0.081563	0.07335		HTGA/T	316,235	-74.2280	0.06117
	ODE/2	**90,381**	**-12569.48662**	**0**		ODE/2	**200,000**	**-78.332**	1.059E-5
	OGA/Q	302,166	-12569.4537	6.447E-4		OGA/Q	245,930	-78.300	6.288E-3
f_{08}	HTGA	163,468	-12569.4600	**0**	f_{25}	HTGA	216,535	-78.303	**0**
	LEA	287,365	-12569.4542	4.831E-4		LEA	243,895	-78.310	6.127E-3
	HTGA/T	293,204	-10117.1532	5.3528		HTGA/T	326,136	-50.4200	5.02819
	ODE/2	127,666	**0**	**0**					
	OGA/Q	224,710	**0**	**0**					
f_{09}	HTGA	**16,267**	**0**	**0**					
	LEA	223,803	2.103E-18	3.359E-18					
	HTGA/T	257,251	0.055346	O.O641					

Acknowledgment

This work is supported by the Humanities Base Project of Hubei Province under grant No. 2004B0011 and the Natural Science Foundation of Hubei Province under grant No. 2003ABA043.

References

[1] T. Bäck and H.P. Schwefel. An overview of evolutionary algorithms for parameter optimization. Evolutionary Computation 1 (1993) 1 - 23.

[2] X. Yao, Y. Liu, G. Lin. Evolutionary programming made faster. IEEE Transactions on Evolutionary Computation 3 (1999) 82 - 102.

[3] K. Deep, M. Thakur. A new crossover operator for real coded genetic algorithms. Applied Mathematics and Computation 188 (2007) 895-911.

[4] K. Deep, M. Thakur. A new mutation operator for real coded genetic algorithms. Applied Mathematics and Computation 193 (2007) 211-230.

[5] J. Brest, S. Greiner, B. Bošković, *et al*. Self-adapting control parameters in differential evolution: a comparative study on numerical benchmark problems. IEEE Transactions on Evolutionary Computation 10 (2006) 646 - 657.

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

[7] N. Noman, H. Iba. Accelerating differential evolution using an adaptive local search. IEEE Transactions on Evolutionary Computation, DOI: 10.1109/TEVC.2007.895272 (in press).

[8] S. Rahnamayan, H.R. Tizhoosh,M.M.A. Salama. A novel population initialization method for accelerating evolutionary algorithms. Computers and Mathematics with Applications 53 (2007) 1605 - 1614.

[9] S. Rahnamayan, H.R. Tizhoosh,M.M.A. Salama. Opposition-based differential evolution. IEEE Transactions on Evolutionary Computation, DOI: 10.1109/TEVC.2007.894200 (in press).

[10] R. Storn, K. Price, Differential Evolution–A simple and efficient heuristic for global optimization over continuous spaces Journal of Global Optimization 11 (1997) 341 - 359.

[11] R. Storn, K. Price, Home page of differential evolution. Available online at http://www.ICSI.Berkeley.edu/~storn/code.html. (2003).

[12] R. Gäperle, S.D. Müler, P. Koumoutsakos, A parameter study for differential evolution. WSEAS NNA-FSFS-EC 2002. [Online]:

http://citeseer.ist.psu.edu/526865.html
[13] B. Alatas, E. Akin, A. Karci. MODENAR: Multi-objective differential evolution algorithm for mining numeric association rules. Applied Soft Computing 8 (2008) 646 - 656.
[14] S. Das, A. Abraham, A. Konar. Automatic clustering using an improved differential evolution algorithm. IEEE Transaction on Systems Man and Cybernetics A, DOI: 10.1109/TSMCA.2007.909595 (in press).
[15] K. Price, R. Storn, J. Lampinen. Differential evolution: a practical approach to global optimization. Berlin: Springer-Verlag, 2005.
[16] H.Y. Fan, J. Lampinen. A trigonometric mutation operation to differential evolution. Journal of Global Optimization 27 (2003) 105 - 129.
[17] J. Sun, Q. Zhang, E. Tsang. DE/EDA: A new evolutionary algorithm for global optimization. Information Science. 169 (2004) 249 - 262.
[18] W.Y. Gong, Z.H. Cai, C.X. Ling. ODE: A fast and robust differential evolution based on orthogonal design. AI 2006: Advances in Artificial Intelligence - 19th Australian Joint Conference on Artificial Intelligence, LNAI 4304 (2006) 709-718.
[19] J. Vesterstroem, R. Thomsen. A comparative study of differential evolution, particle swarm optimization, and evolutionary algorithms on numerical benchmark problems. IEEE Congress Evolutionary Computation (CEC-04), (2004) 1980 - 1987.
[20] J. Liu, J. Lampinen. A fuzzy adaptive differential evolution algorithm. Soft Computing - A Fusion of Foundations, Methodologies and Applications, 9 (2005) 448 - 642.
[21] A. Salman, A.P. Engelbrecht, M.G.H. Omran. Empirical analysis of self-adaptive differential evolution. European Journal of Operational Research 183 (2007) 785 - 804.
[22] A. Nobakhti, H. Wang. A simple self-adaptive differential evolution algorithm with application on the ALSTOM gasifier. Applied Soft Computing 8 (2008) 350 - 370.
[23] K.T. Fang, C.X. Ma. Orthogonal and uniform design (in Chinese). Beijing, China: Science Press, 2001.
[24] Q. Zhang, Y.W. Leung. Orthogonal genetic algorithm for multimedia multicast routing. IEEE Transactions on Evolutionary Computation, 3 (1999) 53 - 62.
[25] Y.W. Leung, Y. Wang. An orthogonal genetic algorithm with quantization for global numerical optimization. IEEE Transactions on Evolutionary Computation, 5 (2001) 41 - 53.
[26] S.Y. Ho, Y.C. Chen. An efficient evolutionary algorithm for accurate polygonal approximation. Pattern Recognition, 34 (2001) 2305 - 2317.
[27] S.Y. Ho, L. Shu, J.H. Chen. Intelligent evolutionary algorithms for large parameter optimization problems. IEEE Transactions on Evolutionary Computation, 8 (2004) 522 - 541.
[28] W. Zhong, J. Liu, M. Xue, *et al*. A multi-agent genetic algorithm for global numerical optimization. IEEE Transactions Systems Man Cybernetics B, 34 (2004) 1128 - 1141.
[29] S.Y. Zeng, L.S. Kang, L.X. Ding. An Orthogonal Multiobjective Evolutionary Algorithm for Multi-objective Optimization Problems with Constraints. Evolutionary Computation 12 (2004) 77 - 98.
[30] J.T. Tsai, T.K. Liu, J.H. Chou. Hybrid Taguchi-genetic algorithm for global numerical optimizatio. IEEE Transactions on Evolutionary Computation, 8 (2004) 365 - 377.
[31] Z.H. Cai, W.Y. Gong, Y.Q. Huang. A Novel Differential Evolution Algorithm based on ϵ-domination and Orthogonal Design Method for Multiobjective Optimization, Fourth International Conference on Evolutionary Multi-Criterion Optimization, LNCS 4403 (2007) 286 - 301.
[32] Y. Wang, H. Liu, Z.X. Cai, *et al*. An orthogonal design based constrained evolutionary optimization algorithm. Engineering Optimization, 39 (2007) 715 - 736.