Journal of Intelligent Learning Systems and Applications, 2012, 4, 1-19
http://dx.doi.org/10.4236/jilsa.2012.41001 Published Online February 2012 (http://www.SciRP.org/journal/jilsa)

Differential Evolution Using Opposite Point for Global Numerical Optimization

Youyun Ao[1], Hongqin Chi[2]

[1]School of Computer and Information, Anqing Teachers College, Anqing, China; [2]College of Information, Mechanical and Electrical Engineering, Shanghai Normal University, Shanghai, China.
Email: youyun_ao@tom.com

Received September 25th, 2010; revised May 8th, 2011; accepted May 20th, 2011

ABSTRACT

The Differential Evolution (DE) algorithm is arguably one of the most powerful stochastic optimization algorithms, which has been widely applied in various fields. Global numerical optimization is a very important and extremely difficult task in optimization domain, and it is also a great need for many practical applications. This paper proposes an opposition-based DE algorithm for global numerical optimization, which is called GNO2DE. In GNO2DE, firstly, the opposite point method is employed to utilize the existing search space to improve the convergence speed. Secondly, two candidate DE strategies "DE/rand/1/bin" and "DE/current to best/2/bin" are randomly chosen to make the most of their respective advantages to enhance the search ability. In order to reduce the number of control parameters, this algorithm uses an adaptive crossover rate dynamically tuned during the evolutionary process. Finally, it is validated on a set of benchmark test functions for global numerical optimization. Compared with several existing algorithms, the performance of GNO2DE is superior to or not worse than that of these algorithms in terms of final accuracy, convergence speed, and robustness. In addition, we also especially compare the opposition-based DE algorithm with the DE algorithm without using the opposite point method, and the DE algorithm using "DE/rand/1/bin" or "DE/current to best/2/bin", respectively.

Keywords: Differential Evolution; Evolutionary Algorithm; Global Numerical Optimization; Stochastic Optimization

1. Introduction

Global numerical optimization problems arise in almost every field such as industry and engineering design, applied and social science, and statistics and business, etc. The aim of global numerical optimization is to find global optima of a generic objective function. In this paper, we are most interested in the following global numerical minimization problem [1,2]:

$$\min\{f(\boldsymbol{x})\},\quad \boldsymbol{L}\le \boldsymbol{x}\le \boldsymbol{U} \tag{1}$$

where $f(\boldsymbol{x})$ is the objective function to be minimized, $\boldsymbol{x}=(x_1,x_2,\cdots,x_n)\in R^n$ is the real-parameter variable vector, $\boldsymbol{L}=(l_1,l_2,\cdots,l_n)$ is the lower bound of the variables and $\boldsymbol{U}=(u_1,u_2,\cdots,u_n)$ is the upper bound of the variables, respectively, such that $x_i\in[l_i,u_i]$.

Many real-world global numerical optimization problems have many objective functions that are non-differentiable, non-continuous, non-linear, noisy, flat, random, or that have many local minima, multiple dimensions, etc. However, the major challenge of the global numerical optimization is that the problems to be optimized have many local optima and multiple dimensions. Such problems are extremely difficult to be optimized and find reliable global optima [3,4]. Therefore, increasing requirements for solving global numerical optimization in various application domains have encouraged many researchers to find a reliable global numerical optimization algorithm. However, in the last decades, this problem remains intractable, theoretically at least [5].

In the global numerical optimization, the traditional methods can be usually classified into two main categories [5,6]: deterministic and probabilistic global numerical optimization methods. During the global numerical optimization process, the first stage is usually to find specific heuristic information involved in problem. Most of deterministic methods rely on the heuristic information to escape from local minima. On the other hand, almost probabilistic methods rely on a probability to determine whether or not search should depart from the neighborhood of a local minimum. Evolutionary algorithms (including genetic algorithm (GA) [7], evolution strategy (ES) [8], genetic programming (GP) [9], and evolutionary programming (EP) [10]) are inspired from the evolution of nature and relatively recent optimization methods. These algorithms have the potential to over-

come the limitations of traditional global numerical optimization methods, mainly in terms of unknown system parameters, multiple local minima, non-differentiability, or multiple dimensions, etc. [5,11].

Lately, some new methods for global numerical optimization were gradually introduced. Particle Swarm Optimization (PSO) was originally proposed by J. Kennedy as a simulation of social behavior, and it was initially introduced as an optimization method in 1995 [12]. PSO has been a member of the wide category of Swarm Intelligence methods for solving global numerical optimization problems [13-15]. Differential Evolution (DE) was introduced by Storn and Price in 1995, and developed to optimize real-parameter functions [3,16,17]. DE mainly uses the distance and direction information from the current population to guide its further search, and it mainly has three advantages: 1) finding the true global minimum regardless of the initial parameter values; 2) fast convergence; 3) using a few control parameters. In addition, DE is simple, fast, easy to use, very easily adaptable and useful for optimizing multimodal search spaces [18-22]. Recently, DE has been shown to produce superior performance, and perform better than GA and PSO over some global numerical optimization problems [13,14]. Therefore, DE is very promising in solving global numerical optimization problems.

This paper proposes an opposition-based DE algorithm for global numerical optimization (GNO2DE). This algorithm employs the opposite point method to utilize the existing search spaces to speed the convergence [21-24]. Usually, different problems require different settings for the control parameters. Generally, adaptation is introduced into an evolutionary algorithm, which can improve the ability to solve a general class of problems, without user interaction. In order to improve the adaptation and reduce the control parameter, GNO2DE uses a dynamic mechanism to dynamically tune the crossover rate CR during the evolutionary process. Moreover, GNO2DE can enhance the search ability by randomly selecting a candidate from strategies "DE/rand/1/bin" and "DE/current to best/2/bin". Numerical experiments clearly show that GNO2DE is feasible and effective.

The remainder of this paper is organized as follows. Section 2 briefly introduces the basic idea of the DE algorithm. Section 3 describes in detail the proposed GNO2DE algorithm. Section 4 presents the experimental setup adopted and provides an analysis of the experimental results obtained from our empirical study. Finally, our conclusions and some possible paths for the future research are provided in Section 5.

2. The Classical DE Algorithm

The DE algorithm is a population-based stochastic optimization algorithm like many evolutionary algorithms such as genetic algorithms using three similar genetic operators: crossover, mutation, and selection [7]. The main difference in generating better solutions is that genetic algorithms mainly rely on crossover while DE mainly relies on mutation operation. The DE algorithm uses mutation operation as a search mechanism and selection operation to direct the search toward the prospective regions in the search space. The DE algorithm also uses a non-uniform crossover that can take child vector parameters from one parent more often than it does from others. By using the components of the existing population members to generate trial vectors, the recombination (*i.e.*, crossover) operator efficiently shuffles information about successful combinations, enabling the search for a better solution space [3,16,17].

A global numerical optimization problem consisting of n parameters can be represented by a n-dimensional vector. In DE, a population of NP solution vectors is randomly created at the start, where $NP > 4$. The population is successfully improved by applying mutation, crossover, and selection operators [13,25,26].

2.1. Randomly Initializing Population

Like other many evolutionary algorithms, the DE algorithm starts with an initial population, which is randomly generated when no preliminary knowledge about the solution is available. In DE, let us assume that an individual $\boldsymbol{x}_{i,G} = \left(x_{i,1,G}, x_{i,2,G}, \cdots, x_{i,n,G}\right)$ stands for the ith individual of population P_G (population size NP) at the generation G. The population $P_0 = \left\{\boldsymbol{x}_{1,0}, \boldsymbol{x}_{2,0}, \cdots, \boldsymbol{x}_{NP,0}\right\}$ is initialized as

$$\forall i \le NP,\ \forall j \le n : x_{i,j,0} = l_j + rand_j \times \left(u_j - l_j\right) \quad (2)$$

where NP is the population size, n is the number of variables, $rand_j$ is a uniformly distributed random number in the range [0,1], and $x_{i,j,0}$ is the jth variable of the ith individual at the initial generation, which is initialized within the jth range $\left[l_j, u_j\right]$.

2.2. Mutation Operation

In the mutation phase, DE randomly selects three distinct individuals from the current population. For each target vector $\boldsymbol{x}_{i,G}$, the ith mutant vector is generated based on the three selected individuals as follows:

$$\boldsymbol{v}_{i,G+1} = \boldsymbol{x}_{r_1,G} + F \times \left(\boldsymbol{x}_{r_2,G} - \boldsymbol{x}_{r_3,G}\right) \quad (3)$$

where $i = 1, 2, \cdots, NP$, random indexes $r_1, r_2, r_3 \in \{1, 2, \cdots, NP\}$ are randomly chosen integers, mutually different, and they are also chosen to be different from the running index i, so that NP must be greater or equal to four to allow for this condition. The scaling

factor F is a control parameter of the DE algorithm, which controls the amplification of the differential variation ($\boldsymbol{x}_{r_2,G} - \boldsymbol{x}_{r_3,G}$). And the scaling factor F is a real constant factor in the range $[0,2]$ and is often set to 0.5 in the real applications [27].

The above strategy is called "DE/rand/1/bin", it is not the only variant of DE mutation which has been proven to be useful for real-valued optimization. In order to classify the variants of DE mutation, the notation: $DE/x/y/z$ is introduced where 1) x specifies the vector to be mutated which currently can be "rand" (a randomly chosen population vector) or "best" (the vector of lowest cost from the current population); 2) y is the number of difference vectors used; 3) z denotes the crossover scheme, there are two crossover schemes often used, namely, "bin" (*i.e.*, the binomial recombination) and "exp" (*i.e.*, the exponential recombination). Usually, there are the following several differential DE schemes often used in the global optimization [3]:

"DE/best/1/bin":

$$\boldsymbol{v}_{i,G+1} = \boldsymbol{x}_{best} + F \times \left(\boldsymbol{x}_{r_1,G} - \boldsymbol{x}_{r_2,G} \right) \tag{4}$$

"DE/current to best/2/bin":

$$\boldsymbol{v}_{i,G+1} = \boldsymbol{x}_i + F \times \left(\boldsymbol{x}_{best} - \boldsymbol{x}_{i,G} \right) + F \times \left(\boldsymbol{x}_{r_1,G} - \boldsymbol{x}_{r_2,G} \right) \tag{5}$$

"DE/best/2/bin":

$$\boldsymbol{v}_{i,G+1} = \boldsymbol{x}_{best} + F \times \left(\boldsymbol{x}_{r_1,G} - \boldsymbol{x}_{r_2,G} \right) + F \times \left(\boldsymbol{x}_{r_3} - \boldsymbol{x}_{r_4} \right) \tag{6}$$

"DE/rand/2/bin":

$$\boldsymbol{v}_{i,G+1} = \boldsymbol{x}_{r_1} + F \times \left(\boldsymbol{x}_{r_2} - \boldsymbol{x}_{r_3} \right) + F \times \left(\boldsymbol{x}_{r_4} - \boldsymbol{x}_{r_5} \right) \tag{7}$$

where $\boldsymbol{x}_{best}$ is the best individual of the current population G. The scaling factor F is the control parameter of the DE algorithm.

2.3. Crossover Operation

In order to increase the diversity of the perturbed parameter vectors, the crossover operator is introduced. The new individual is generated by recombining the original vector $\boldsymbol{x}_{i,G} = \left(x_{i,1,G}, x_{i,2,G}, \cdots, x_{i,n,G} \right)$ and the mutant vector $\boldsymbol{v}_{i,G+1} = \left(v_{i,1,G+1}, v_{i,2,G+1}, ..., v_{i,n,G+1} \right)$ according to the following formula:

$$w_{i,j,G+1} = \begin{cases} v_{i,j,G+1}, \\ \text{if } \left(rand[0,1] \le CR \right) \vee \left(j = rand[1,n] \right) \\ x_{i,j,G}, \quad \text{otherwise} \end{cases} \tag{8}$$

where $rand[0,1]$ stands for a uniformly distributed random number in the range [0,1], and $rand[1,n]$ is a randomly chosen index from the set $\{1,2,\cdots,n\}$ to ensure that at least one of the variables should be changed and $\boldsymbol{w}_{i,G+1}$ does not directly duplicate $\boldsymbol{x}_{i,G}$. And the crossover rate CR is a real constant in the range [0,1], one of control parameters of the DE algorithm. After crossover, if one or more of the variables in the new solution are outside their boundaries, the following repair rule is applied [25]:

$$w_{i,j,G+1} = \begin{cases} \frac{1}{2} \cdot \left(w_{i,j,G+1} + l_j \right), & \text{if } w_{i,j,G+1} < l_j \\ l_j + \frac{1}{2} \cdot \left(w_{i,j,G+1} - u_j \right), & \text{if } w_{i,j,G+1} > u_j \\ w_{i,j,G+1}, & \text{otherwise} \end{cases} \tag{9}$$

2.4. Selection Operation

After mutation and crossover, the selection operation selects to decide that the new individual $\boldsymbol{w}_{i,G+1}$ or the original individual $\boldsymbol{x}_{i,G}$ will survive to be a member of the next generation. If the fitness value of the new individual $\boldsymbol{w}_{i,G+1}$ is better than that of the original one $\boldsymbol{x}_{i,G}$ then the new individual $\boldsymbol{w}_{i,G+1}$ is to be an offspring in the next generation (G + 1) else the new individual $\boldsymbol{w}_{i,G+1}$ is discarded and the original one $\boldsymbol{x}_{i,G}$ is retained in the next generation. For a minimization problem, we can use the following selection rule:

$$\boldsymbol{x}_{i,G+1} = \begin{cases} \boldsymbol{w}_{i,G+1}, & \text{if } f\left(\boldsymbol{w}_{i,G+1} \right) < f\left(\boldsymbol{x}_{i,G} \right), \\ \boldsymbol{x}_{i,G}, & \text{otherwise.} \end{cases} \tag{10}$$

where $f(\cdot)$ is the fitness function, and $\boldsymbol{x}_{i,G+1}$ is the offspring of $\boldsymbol{x}_{i,G}$ in the next generation (G + 1).

2.5. The General Framework of the DE Algorithm

The above operations (*i.e.*, mutation, crossover, and selection) are repeated NP (population size) times to generate the next population of the current population. These successive generations are generated until the predefined termination criterion is satisfied. The main steps of the DE algorithm are given in **Figure 1**.

1: Randomly initialize the starting population P_0.
2: Evaluate the initial population P_0.
3: **repeat**
4: **for** each individual in the current population P_G **do**
5: Perform mutation operation.
6: Perform crossover operation.
7: Evaluate the new individual.
8: Perform selection operation.
9: **end for**
10: Generate the next generation population P_{G+1} through 4-9,
11: and let $G \leftarrow G+1$.
12: **until** (the predefined termination criterion is achieved).

Figure 1. The generic framework of the DE algorithm.

3. The Proposed GNO2DE Algorithm

Similar to all population-based optimization algorithms, two main steps are distinguishable for the DE, population initialization and producing new generations by evolutionary operations such as selection, crossover, and mutation. GNO2DE enhances these two steps using the opposite point method. The opposite point method has been proven to be an effective method to evolutionary algorithms for solving global numerical problems. When evaluating a point to a given problem, simultaneously computing its opposite point can provide another chance for finding a point closer to the global optimum. The concept of the opposite point is defined as follows [21-24]:

Definition 1 Let us assume that $\boldsymbol{x}_{i,G}$ is the ith point of the population P_G (population size NP) at the generation G in the n-dimensional space. The opposite point $\boldsymbol{o}_{i,G} = \left(o_{i,1,G}, o_{i,2,G}, \cdots, o_{i,n,G}\right)$ is completely defined by its components as follows:

$$o_{i,j,G} = l_j + u_j - x_{i,j,G} \tag{11}$$

where $i = 1, 2, \cdots, NP$, $j = 1, 2, \cdots, n$, l_j and u_j are the lower and the upper limits of the variable $x_{i,j,G}$, respectively.

3.1. Generating the Initial Population Using the Opposite Point Method

Generally, population-based Evolutionary Algorithms randomly generate the initial population within the boundaries of parameter variables. In order to improve the quality of the initial population, we can obtain fitter starting candidate solutions by utilizing opposite points, even when there is no a priori knowledge about the solution (s). The procedure of generating the initial population using the opposite point method is given as follows:

Step 1: Randomly initialize the starting population P_0 (population size NP).

Step 2: Calculate the opposite population of P_0 using the opposite point method, and obtain the opposite population OP_0.

Step 3: Select the *NP* fittest individuals from $P_0 \cup OP_0$ as the initial population P_0.

3.2. Evolving the Population Using the Opposite Point Method

By applying a similar approach to the current population, the evolutionary process can be forced to jump to a new solution candidate, which may be fitter than the current one. After generating new population by selection, crossover, and mutation, the opposite population is calculated and the NP fittest individuals are selected from the union of the current population and the opposite population. Following steps describe the procedure:

Step 1: The offspring population P_{G+1} of the current population P_G is generated after performing the corresponding successive DE operations (*i.e.*, mutation, crossover, and selection).

Step 2: Calculate the opposite population of P_{G+1} using the opposite point method, and obtain the opposite population OP_{G+1}.

Step 3: Select the NP fittest individuals from $P_{G+1} \cup OP_{G+1}$ as the next generation population P_{G+1}.

Step 4: Let $G \leftarrow G+1$.

3.3. Adaptive Crossover Rate *CR*

In DE, the aim of crossover is to improve the diversity of the population, and there is a control parameter CR (*i.e.*, the crossover rate) to control the diversity. The smaller diversity is easy to result in the premature convergence, while the larger diversity reduces the convergence speed. In conventional DE, the crossover rate CR is a constant value in the range [0,1]. Inspired by non-uniform mutation, this paper introduces an adaptive crossover rate CR, which is defined as follows [28]:

$$CR = r \cdot \left(1 - \left(\frac{t}{T}\right)^b\right) \tag{12}$$

where r is a uniform random number from [0,1], t and T are the current generation number and the maximal generation number, respectively. The parameter b is a shape parameter determining the degree of dependency on the iteration number and usually is set to 2 or 3. In this study, b is set to 3.

The property of CR causes the crossover operator to search the solution space uniformly initially when t is small, while to search the solution space very locally when t is large. This strategy increases the probability of generating a new number close to its successor than a random choice. Therefore, at the early stage, GNO2DE uses a bigger crossover rate CR to search the solution space to preserve the diversity of solutions and prevent premature convergence; at the later stage, GNO2DE employs a smaller crossover rate CR to search the solution space to enhance the local search and prevent the fitter solutions found from being destroyed. The relation of generation vs crossover rate CR is plotted in **Figure 2**.

3.4. Adaptive Mutation Strategies

In subsection 2.2, we have described a few useful mutation schemes, where "DE/rand/1/bin" and "DE/current to best/2/bin" are the most often used in practical applications mainly due to their good performance [17,19]. To overcome their respective disadvantages and utilize their

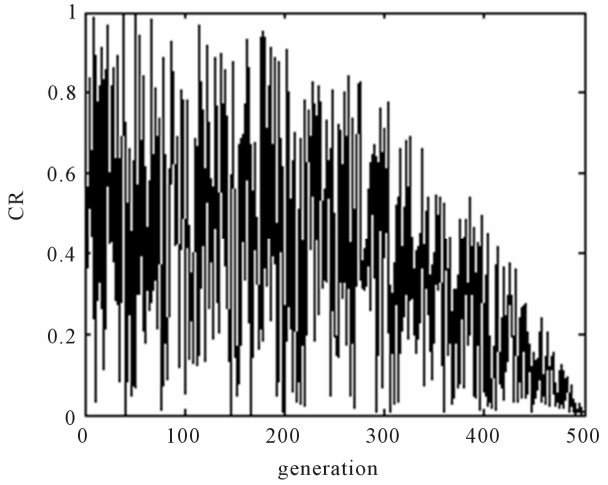

Figure 2. The graph for generation vs crossover rate *CR*.

cooperative advantages, GNO2DE randomly chooses a mutation scheme from two candidates "DE/rand/1/bin" (*i.e.*, Equation (3)) and "DE/current to best/2/bin" (*i.e.*, Equation (5)), and the new mutant vector $\boldsymbol{v}_{i,G+1}$ is generated according to the following formula [27]:

$$\boldsymbol{v}_{i,G+1} = \begin{cases} \text{Equation (3)}, & \text{if } rand[0,1] \le 0.5 \\ \text{Equation (5)}, & \text{otherwise} \end{cases} \tag{13}$$

where $rand[0,1]$ is a uniform random number from the range [0,1].

3.5. Approaching of Boundaries

In the given optimization problem, it has to be ensured that some boundary values are not outside their limits. Several possibilities exist for this task: 1) The positions that beyond the boundaries are newly generated until the positions within the boundaries are satisfied; 2) the boundary-exceeding values are replaced by random numbers in the feasible region; 3) The boundary is approached asymptotically by setting the boundary-offending value to the middle between old position and boundary [29]:

$$w_{i,j,G+1} = \begin{cases} \frac{1}{2}\cdot\left(l_j + w_{i,j,G+1}\right), & \text{if } w_{i,j,G+1} < l_j \\ \frac{1}{2}\cdot\left(w_{i,j,G+1} + u_j\right), & \text{if } w_{i,j,G+1} > u_j \\ w_{i,j,G+1}, & \text{otherwise} \end{cases} \tag{14}$$

After crossover, if one or more of the variables in the new vector $\boldsymbol{w}_{i,G+1}$ are outside their boundaries, the violated variable value $w_{i,j,G+1}$ is either reflected back from the violated boundary or set to the corresponding boundary value using the repair rule as follows [30]:

$$w_{i,j,G+1} = \begin{cases} \frac{1}{2}\cdot\left(l_j + w_{i,j,G+1}\right), & \text{if } \left(p \le 1/3\right) \wedge \left(w_{i,j,G+1} < l_j\right) \\ l_j, & \text{if } \left(1/3 < p \le 2/3\right) \wedge \left(w_{i,j,G+1} < l_j\right) \\ 2l_j - w_{i,j,G+1}, & \text{if } \left(p > 2/3\right) \wedge \left(w_{i,j,G+1} < l_j\right) \\ \frac{1}{2}\cdot\left(w_{i,j,G+1} + u_j\right), & \text{if } \left(p \le 1/3\right) \wedge \left(w_{i,j,G+1} > u_j\right) \\ u_j, & \text{if } \left(1/3 < p \le 2/3\right) \wedge \left(w_{i,j,G+1} > u_j\right) \\ 2u_j - w_{i,j,G+1}, & \text{if } \left(p > 2/3\right) \wedge \left(w_{i,j,G+1} > u_j\right) \end{cases} \tag{15}$$

where p is a probability and a uniformly distributed random number in the range [0,1].

3.6. The Framework of the GNO2DE Algorithm

DE 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 value. The initial population is selected randomly in a uniform manner between the lower and upper bounds defined for each variable. These bounds are specified by the user according to the nature of the problem. After initialization, DE performs mutation, crossover, selection etc., in an evolution process. The general framework of the GNO2DE algorithm is described in **Figure 3**.

4. Numerical Experiments

4.1. Benchmark Functions

In order to test the robustness and effectiveness of GNO2DE, we use a well-known test set of 23 benchmark functions [1,2,31-33]. This relatively large set is necessary in order to reduce biases in evaluating algorithms.

1: Generate the initial population P_0 (population size NP)
2: using the opposite point method.
3: Evaluate the initial population P_0 .
4: **repeat**
5: **for** each individual in the current population P_G **do**
6: Perform mutation operation.
7: Perform crossover operation.
8: Evaluate the new individual.
9: Perform selection operation.
10: **end for**
11: Generate the next generation population P_{G+1} through 4-10,
12: and calculate the opposite population OP_{G+1} .
13: Select the NP fittest individuals from $P_{G+1} \bigcup OP_{G+1}$
14: as the next generation population P_{G+1} .
15: Let $G \leftarrow G+1$.
16: **until** (the predefined termination criterion is achieved).

Figure 3. The general framework of GNO2DE.

The complete description of all these functions and the corresponding parameters involved are described in **Table 1** and **APPENDX**. These functions can be divided into three different categories with different complexities:

Table 1. The 23 benchmark test functions f_1 - f_{23}.

f	n	S	$f_{\min}$
$f_1(\boldsymbol{x})=\sum_{i=1}^{n}x_i^2$	30/100	$[-5.12,5.12]^n$	$f_1(\mathbf{0})=0$
$f_2(\boldsymbol{x})=\sum_{i=1}^{n}\lvert x_i\rvert+\prod_{i=1}^{n}\lvert x_i\rvert$	30/100	$[-10,10]^n$	$f_2(\mathbf{0})=0$
$f_3(\boldsymbol{x})=\sum_{i=1}^{n}\left(\sum_{j=1}^{i}x_j\right)^2$	30/100	$[-100,100]^n$	$f_3(\mathbf{0})=0$
$f_4(\boldsymbol{x})=\max_i\{\lvert x_i\rvert,1\le i\le n\}$	30/100	$[-100,100]^n$	$f_4(\mathbf{0})=0$
$f_5(\boldsymbol{x})=\sum_{i=1}^{n-1}\left(100\left(x_{i+1}-x_i^2\right)^2+\left(x_i-1\right)^2\right)$	30/100	$[-30,30]^n$	$f_5(\mathbf{1})=0$
$f_6(\boldsymbol{x})=\sum_{i=1}^{n}\left(\lfloor x_i+0.5\rfloor\right)^2$	30/100	$[-100,100]^n$	$f_6(\boldsymbol{p})=0,-0.5\le p_i\le 0.5$
$f_7(\boldsymbol{x})=\sum_{i=1}^{n}ix_i^4+rand[0,1)$	30/100	$[-1.28,1.28]^n$	$f_7(\mathbf{0})=0$
$f_8(\boldsymbol{x})=-\sum_{i=1}^{n}x_i\sin\left(\sqrt{\lvert x_i\rvert}\right)$	30/100	$[-500,500]^n$	$f_8(\mathbf{420.97})=-418.9829n$
$f_9(\boldsymbol{x})=\sum_{i=1}^{n}\left(x_i^2-10\cos(2\pi x_i)+10\right)$	30/100	$[-5.12,5.12]^n$	$f_9(\mathbf{0})=0$
$f_{10}(\boldsymbol{x})=-20\exp\left(-0.2\sqrt{\frac{1}{n}\sum_{i=1}^{n}x_i^2}\right)-\exp\left(\frac{1}{n}\sum_{i=1}^{n}\cos(2\pi x_i)\right)+20+\mathrm{e}$	30/100	$[-32,32]^n$	$f_{10}(\mathbf{0})=0$
$f_{11}(\boldsymbol{x})=\frac{1}{4000}\sum_{i=1}^{n}x_i^2-\prod_{i=1}^{n}\cos\left(\frac{x_i}{\sqrt{i}}\right)+1$	30/100	$[-600,600]^n$	$f_{11}(\mathbf{0})=0$
$f_{12}(\boldsymbol{x})=\frac{\pi}{n}\left\{10\sin^2(\pi y_1)+\sum_{i=1}^{n-1}(y_i-1)^2\left[1+10\sin^2(\pi y_{i+1})\right]+(y_n-1)^2\right\}+\sum_{i=1}^{n}u(x_i,10,100,4)$	30/100	$[-50,50]^n$	$f_{12}(-\mathbf{1})=0$
$f_{13}(\boldsymbol{x})=0.1\left\{\sin^2(3\pi x_1)+\sum_{i=1}^{n-1}(x_i-1)^2\left[1+\sin^2(3\pi x_{i+1})\right]+(x_n-1)^2\left[1+\sin^2(2\pi x_n)\right]\right\}+\sum_{i=1}^{n}u(x_i,5,100,4)$	30/100	$[-50,50]^n$	$f_{13}(\mathbf{1})=0$
$f_{14}(\boldsymbol{x})=\left(0.002+\sum_{j=1}^{25}\left(j+\sum_{i=1}^{2}\left(x_i-a_{ij}\right)^6\right)^{-1}\right)^{-1}$	2	$[-65.54,65.54]^n$	$f_{14}(-\mathbf{31.95})=0.998$
$f_{15}(\boldsymbol{x})=\sum_{i=1}^{11}\left[a_i-\frac{x_1\left(b_i^2+b_ix_2\right)}{b_i^2+b_ix_3+x_4}\right]^2$	4	$[-5,5]^n$	$f_{15}(0.1928,0.1908,0.1231,0.1358)=0.0003075$
$f_{16}(\boldsymbol{x})=4x_1^2-2.1x_1^4+\frac{1}{3}x_1^6+x_1x_2-4x_2^2+4x_2^4$	2	$[-5,5]^n$	$f_{16}(-0.09,0.71)=-1.0316$
$f_{17}(\boldsymbol{x})=\left(x_2-\frac{5.1}{4\pi^2}x_1^2+\frac{5}{\pi}x_1-6\right)^2+10\left(1-\frac{1}{8\pi}\right)\cos(x_1)+10$	2	$[-5,15]^n$	$f_{17}(9.42,2.47)=0.397887$
$f_{18}(\boldsymbol{x})=\left[1+(x_1+x_2+1)^2\left(19-14x_1+3x_1^2-14x_2+6x_1x_2+3x_2^2\right)\right]\times\left[30+(2x_1-3x_2)^2\times\left(18-32x_1+12x_1^2+48x_2-36x_1x_2\right)+27x_2^2\right]$	2	$[-2,2]^n$	$f_{18}(0,-1)=3$
$f_{19}(\boldsymbol{x})=-\sum_{i=1}^{4}c_i\exp\left[-\sum_{j=1}^{3}a_{ij}\left(x_j-p_{ij}\right)^2\right]$	3	$[0,1]^n$	$f_{19}(0.114,0.556,0.852)=-3.86278$
$f_{20}(\boldsymbol{x})=-\sum_{i=1}^{4}c_i\exp\left[-\sum_{j=1}^{6}a_{ij}\left(x_j-p_{ij}\right)^2\right]$	6	$[0,1]^n$	$f_{20}(0.201,0.15,0.477,0.275,0.311,0.657)=-3.32237$
$f_{21}(\boldsymbol{x})=-\sum_{i=1}^{5}\left[(x-a_i)(x-a_i)^{\mathrm{T}}+c_i\right]^{-1}$	4	$[0,10]^n$	$f_{21}(\approx\mathbf{4})=-10.1532$
$f_{22}(\boldsymbol{x})=-\sum_{i=1}^{7}\left[(x-a_i)(x-a_i)^{\mathrm{T}}+c_i\right]^{-1}$	4	$[0,10]^n$	$f_{22}(\approx\mathbf{4})=-10.402$
$f_{23}(\boldsymbol{x})=-\sum_{i=1}^{10}\left[(x-a_i)(x-a_i)^{\mathrm{T}}+c_i\right]^{-1}$	4	$[0,10]^n$	$f_{23}(\approx\mathbf{4})=-10.53649$

1) unimodal functions (f_1 - f_7), which are relatively easy to be optimized, but the difficulty increases as the dimensions of the problems increase (see **Figure 4**); 2) multimodal functions (f_8 - f_{13}), which have many local minima, represent the most difficult class of problems for many optimization algorithms (see **Figure 5**); 3) multimodal functions (f_{14} - f_{23}), which contain only few local optima (see **Figure 6**). It is interesting to note that some functions have unique features: f_6 is a discontinuous step function having a single optimum; f_7 is a noisy function involving a uniformly distributed random variable within the range [0,1]. In unimodal functions the convergence rate is our main interest, as the optimization is not a hard problem. Obviously, for multimodal functions the quality of the final results is more important because it reflects the ability of the designed algorithm to escape from local optima.

4.2. Discussion of Parameter Settings

In order to setup the parameters, we firstly discuss the convergence characteristic of each function of dimen-

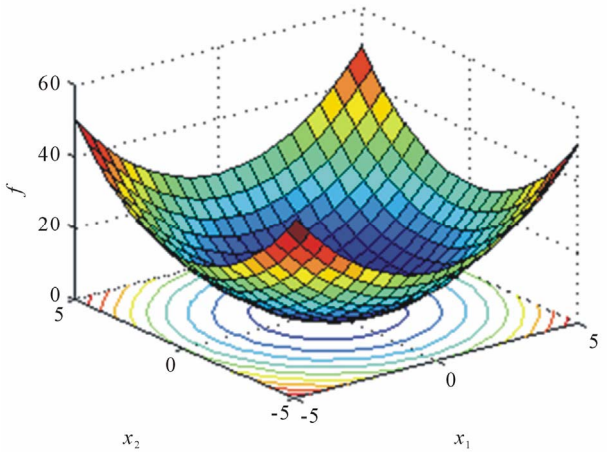

Figure 4. Graph for one unimodal function.

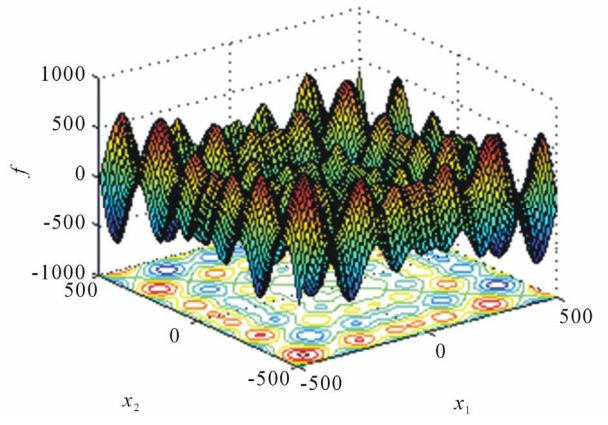

Figure 5. Graph for one multimodal function with many local minima.

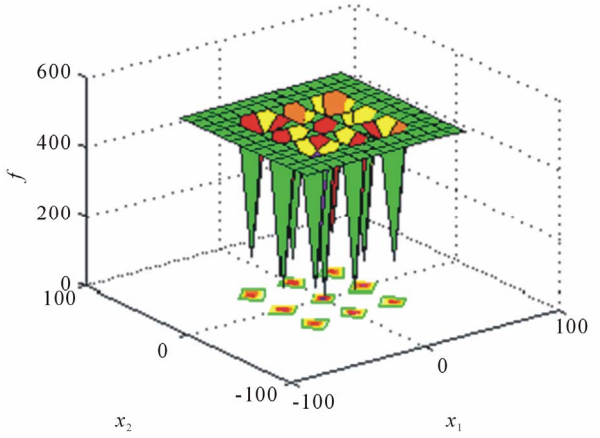

Figure 6. Graph for one multimodal function containing only few local optima.

sionality 30 or lower. The parameters used by GNO2DE are listed in the following: the control parameter $F = 0.5$, the population size $NP = 100$, the maximal generation number $T = 500$ for functions f_1 - f_4, f_{21} - f_{23}, $T = 1500$ for functions f_5 - f_{20}, respectively. For convenience of illustration, we plot the convergence graphs for benchmark test functions f_1 - f_{23} in **Figures 7-12**.

Figures 7-12 clearly show that GNO2DE can achieve better convergence for each function of f_1 - f_4, f_6 - f_7, f_9 - f_{20}, and f_{21} - f_{23}, when evaluated by 100,000 FES (the number of fitness evaluations). From **Figure 8**, we know that function f_8 approximately requires 300,000 FES to achieve the convergence, and that the convergence speed of function f_5 is relatively slow in the case of the above parameters. Therefore, in order to investigate the effect of the control parameter F on the convergence. Some experimental results are given in **Figures 13-18**. Firstly, the control parameter F is set to different values 0.4, 0.5, 0.6, 0.7 on functions f_1 and f_2, and the convergence curve is presented in **Figures 13** and **14**. From **Figures 13** and **14**, we can observe that GNO2DE can achieve the convergence for each value of the above control parameter F when the number of fitness evaluations is set to 100,000 FES, while the convergence speed is fastest when the value of the control parameter F is set to 0.5. For function f_5, we set the control parameter F to 0.5, 0.6, 0.7, and 0.8, respectively. The convergence graph is given in **Figure 15**. From **Figure 15**, it is clearly shown that the convergence speed is obviously fastest when the value of the control parameter F is set to 0.6. In addition, we also present the convergence graph of each function of f_8, f_{13}, and f_{20} in **Figures 16-18**, respectively. The control parameter F is set to 0.5, 0.6, 0.7, and 0.8. From these figures, we can institutively find that the convergence speed is relatively fastest when the value of the control parameter is set to 0.5.

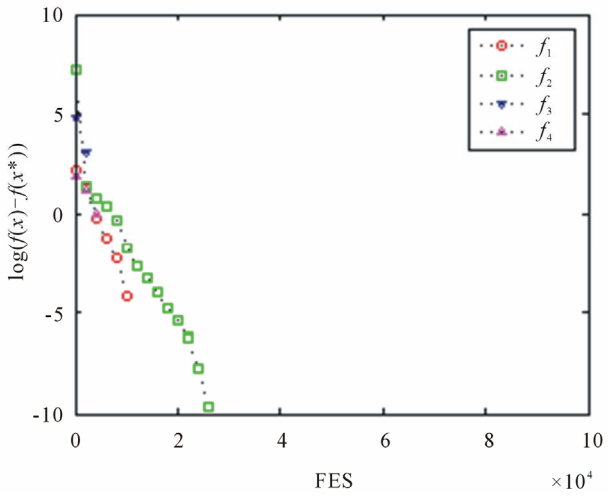

Figure 7. Convergence graph for functions f_1 - f_4.

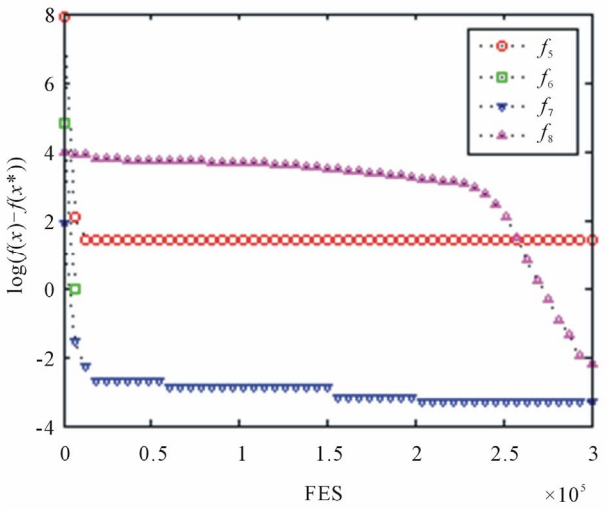

Figure 8. Convergence graph for functions f_5 - f_8.

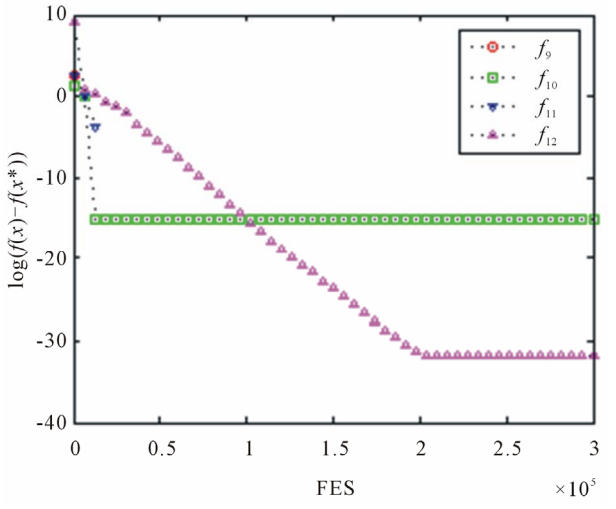

Figure 9. Convergence graph for functions f_9 - f_{12}.

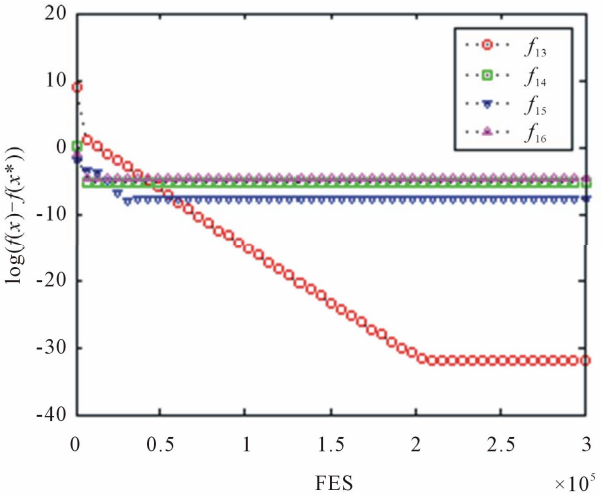

Figure 10. Convergence graph for functions f_{13} - f_{16}.

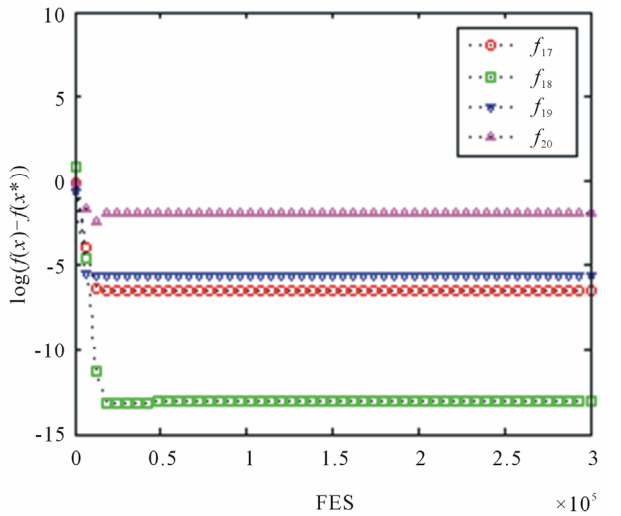

Figure 11. Convergence graph for functions f_{17} - f_{20}.

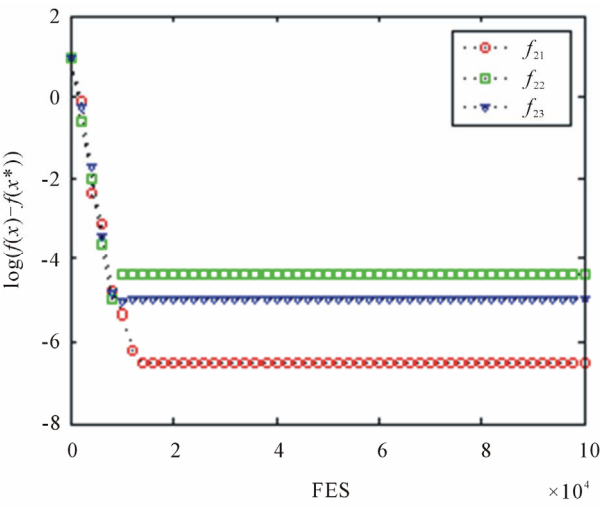

Figure 12. Convergence graph for functions f_{21} - f_{23}.

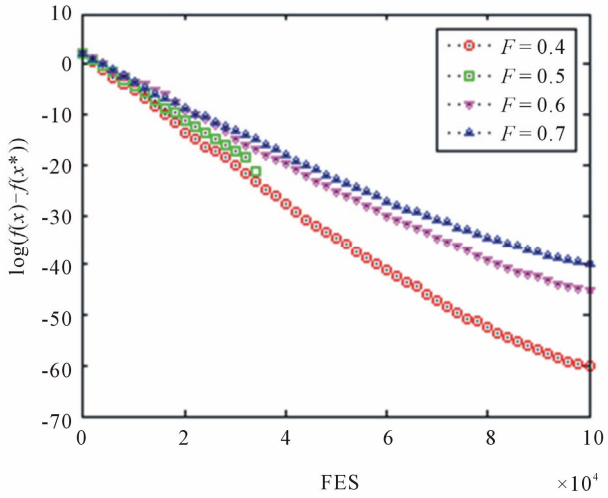

Figure 13. Convergence curve of f_1 for each *F* value.

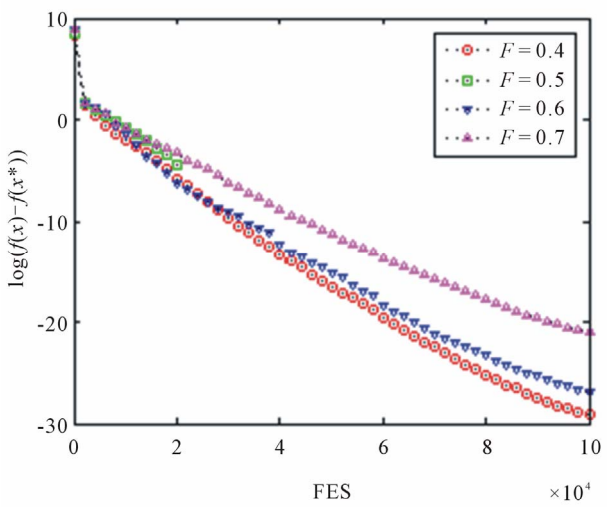

Figure 14. Convergence curve of f_2 for each *F* value.

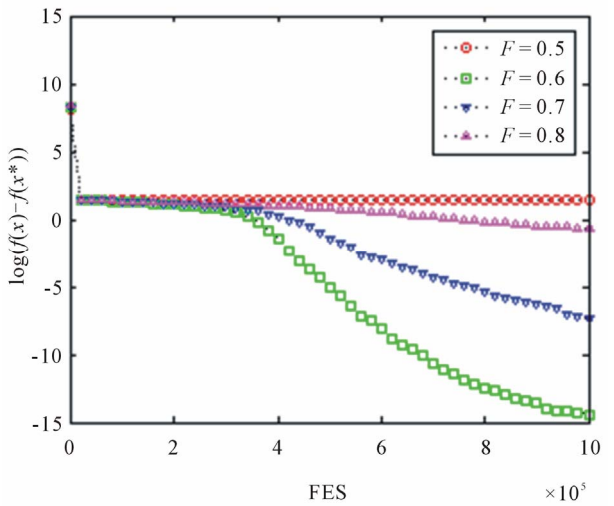

Figure 15. Convergence curve of f_5 for each *F* value.

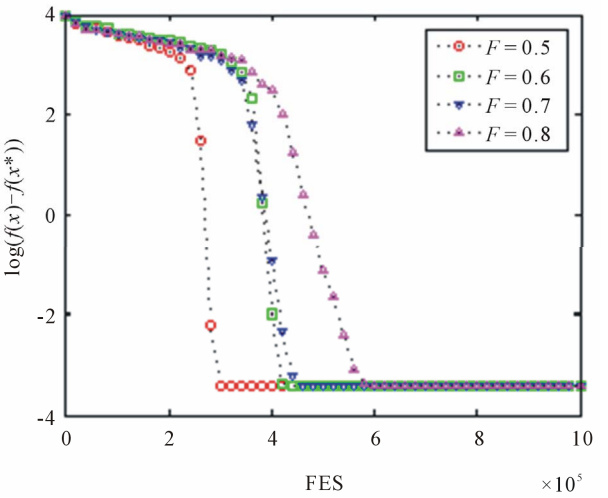

Figure 16. Convergence curve of f_8 for each *F* value.

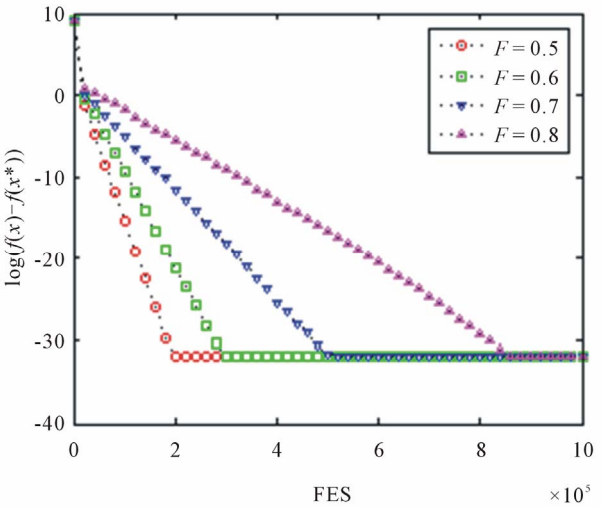

Figure 17. Convergence curve of f_{13} for each *F* value.

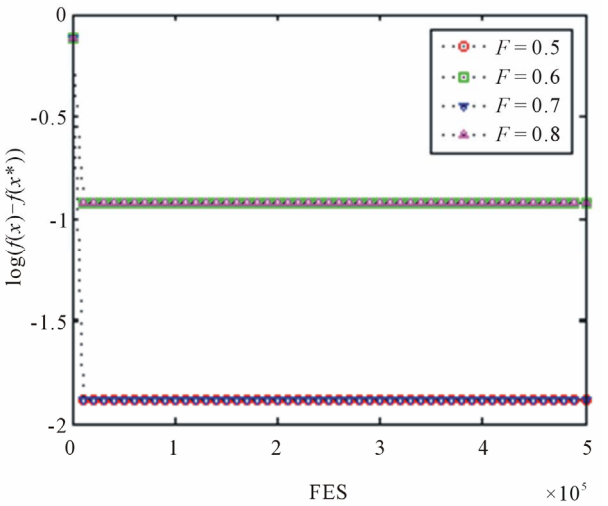

Figure 18. Convergence curve of f_{20} for each *F* value.

Therefore, for the most functions, GNO2DE can show good performance when the value of the control parameter F is set to 0.5 or 0.6. According to the above discussion and analysis, we set up the corresponding experimental parameters in **Tables 2-4**. **Table 2** presents the parameters used by GNO2DE, GNO2DE-A, and GNO2DE-B for functions of dimensionality 30 or lower, where GNO2DE-A and GNO2DE-B employ only DE schemes "DE/rand/1/bin" or "DE/current to best/2/bin", respectively. **Table 3** presents the parameters used by GNODE (without using the opposite point method) for functions of dimensionality 30 or lower. **Table 4** presents the parameter settings used by GNO2DE for functions of dimensionality 100.

4.3. Comparison of GNO2DE with GNODE

In this section, we compare GNO2DE with GNODE in terms of some performance indices according to the parameter settings presented in **Tables 2** and **3**. The experimental results are in detail summarized in **Tables 5** and **6**, and better results are highlighted in boldface. The optimized objective function values over 30 independent runs are arranged in ascending order and the 15th value in the list is called the median optimized function value.

According to **Tables 5** and **6**, we can find that GNO2DE can obtain the optima or near optima with certain precision for all test functions f_1 - f_{23} of dimensionality 30 or less. For each function of f_1 - f_4, f_6, f_7, f_9 - f_{11}, f_{14}, f_{15}, f_{17} - f_{19}, and f_{21} - f_{23}, the performance of GNO2DE is superior to or less worse than the performance of GNODE in terms of the min value (*i.e.*, the best result), the median value (*i.e.*, the median result), the max value (*i.e.*, the worst result), the mean value (*i.e.*, the mean result), and the std value (*i.e.*, the standard deviation result), on condition that while the FES of GNO2DE is essentially less than that of GNODE, although they are apparently set to the same FES 100,000. In addition, the global optimum of function f_{18} found by GNO2DE is $f_{18}(\boldsymbol{x})$ = 2.99999999999992, the corresponding $\boldsymbol{x}$ = (0.00000000061668, −0.99999999932877).

According to **Table 5**, for f_5, the performance of GNO2DE is obviously better than that of GNODE in terms of the max, mean, and std values, while the performance of GNODE is slightly better than that of GNO2DE in terms of the min, median values. For f_8, the median, max, mean, and std values of GNODE are better than those of GNO2DE, while the min value of GNODE is approximate to that of GNO2DE. For f_{12}, the min, median, max, and mean values of GNO2DE are better than those of GNODE, while the std value of GNO2DE is worse than that of GNODE. The reason is that GNO2DE can't find the optimal solution in very few runs of 30 runs. For function f_{13}, the min, median, max, mean, and std values of GNO2DE are slightly worse than those of GNODE.

As shown in **Table 6**, for f_{16}, the min, median values of GNO2DE are similar to those of GNODE, while the max, mean, and std values of GNO2DE are worse than those of GNODE to some extent. This is because that GNO2DE can't obtain the min value in one or two runs of 30 runs. For f_{20}, the min, and max values obtained by GNO2DE are the same to those by GNODE, while the median value obtained by GNO2DE is worse than that obtained by GNODE. Accordingly, it also decides that the mean, and std values of GNO2DE are worse than those of GNODE. GNO2DE and GNODE all have a tendency to getting stuck in the local optima. The global optima of function f_{20} found by GNO2DE is $f_{20}(\boldsymbol{x})$ = −3.33539215295525, the corresponding
$\boldsymbol{x}$ = (0.20085810809731, 0.15013171771783, 0.47865329178970, 0.27652528463205, 0.31191293322300, 0.65702016661775).

In conclusion, the performance of GNO2DE is relatively stable and obviously better than or not worse than that of GNODE. The reason is that GNO2DE uses the opposite point method to provide another chance for finding a solution more close to the global numerical optimum, without increasing much time.

Table 2. Parameters used by GNO2DE, GNO2DE-A, and GNO2DE-B for functions of dimensionality 30 or lower.

f	F	NP	T	FES
f_1 - f_4, f_6, f_7, f_9 - f_{23}	0.5	100	500	1×10^5
f_5	0.6	100	5000	1×10^6
f_8	0.5	100	1500	3×10^5

Table 3. Parameters used by GNODE for functions of dimensionality 30 or lower.

f	F	NP	T	FES
f_1 - f_4, f_6, f_7, f_9 - f_{23}	0.5	100	1000	1×10^5
f_5	0.6	100	10000	1×10^6
f_8	0.5	100	3000	3×10^5

Table 4. Parameters used by GNO2DE for functions of dimensionality 100.

f	F	NP	T	FES
f_1 - f_4, f_6, f_7, f_9 - f_{23}	0.5	100	2500	5×10^5
f_5	0.6	100	25000	5×10^6
f_8	0.5	100	15000	3×10^6

Table 5. Comparison between GNO2DE, and GNODE on functions f_1 - f_{13} of dimensionality 30.

f	f_{min}	method	min	median	max	mean	std
f_1	0	GNO2DE	**0**	**0**	**5.9825755800e−52**	**2.5518554212e−53**	**1.096785e−52**
		GNODE	7.54155067179e−30	5.399029026467e−29	5.2374789498e−28	1.0300737442e−28	1.256367e−28
f_2	0	GNO2DE	**0**	**0**	**0**	**0**	**0**
		GNODE	6.4413392103e−14	1.311376646166e−13	3.18424035936e−13	1.483498934045e−13	6.397059e−14
f_3	0	GNO2DE	**0**	**0**	**0**	**0**	**0**
		GNODE	0.92997781533084	3.84094195096149	7.55970302545084	4.10611784328047	1.8274011203
f_4	0	GNO2DE	**0**	**0**	**1.16657303078e−23**	**3.888576769266e−25**	**2.129861e−24**
		GNODE	4.02934458353e−3	1.1827251953012e−1	1.41424788000055	2.1194941395447e−1	2.878915e−1
f_5	0	GNO2DE	1.47821832439e−16	4.935106365033e−15	**9.64243063489e−14**	**1.26582991612e−14**	**2.134871e−14**
		GNODE	**0**	**0**	3.98662385430093	0.26577492362006	1.0114388899
f_6	0	GNO2DE	**0**	**0**	**0**	**0**	**0**
		GNODE	0	0	0	0	0
f_7	0	GNO2DE	**1.27769804370e−4**	**1.30652285810e−3**	**2.96374746091e−3**	**1.31349553757e−3**	**7.341583e−4**
		GNODE	2.82761792847e−3	5.21770209218e−3	9.26910954128e−3	5.51405735988e−3	1.348797e−3
f_8	−12569.487	GNO2DE	−1.256948661814e+4	−1.256948605445e+4	−1.256926191933e+4	−1.256946879174e+4	5.11364128e−2
		GNODE	**−1.256948661817e+4**	**−1.256948661817e+4**	**−1.256948661817e+4**	**−1.256948661817e+4**	**1.8500855e−12**
f_9	0	GNO2DE	**0**	**0**	**0**	**0**	**0**
		GNODE	16.94654083083428	23.79300366957455	29.35614772393047	23.59480273338819	2.8085602265
f_{10}	0	GNO2DE	**8.881784197e−16**	**8.881784197e−16**	**8.881784197e−16**	**8.881784197e−16**	**0**
		GNODE	3.28626015289e−14	6.128431095931e−14	1.572075802869e−13	6.708707663468e−14	3.018161e−14
f_{11}	0	GNO2DE	**0**	**0**	**0**	**0**	**0**
		GNODE	0	0	1.477977675483e−2	1.06777005656e−3	3.404385e−3
f_{12}	0	GNO2DE	**5.000000000000e−14**	**3.480000000000e−12**	**1.0421500297161e−1**	**1.064608725538e−2**	3.161382e−2
		GNODE	15.7868982918711	15.7868982918711	15.7868982918711	15.7868982918711	**1.271143e−14**
f_{13}	0	GNO2DE	2.916333541909e−13	1.549222503733e−12	6.434821403064e−12	2.175719010455e−12	1.6271296e−12
		GNODE	**1.328113253296e−27**	**6.739922873415e−27**	**3.55322475119e−26**	**1.06114283139e−26**	**9.0125651e−27**

4.4. Comparison of GNO2DE with GNO2DE-A, and GNO2DE-B

In this section, we compare GNO2DE ("DE/rand/1/bin" and "DE/current to best/2/bin") with GNO2DE-A ("DE/rand/1/bin"), and GNO2DE-B ("DE/current to best/2/bin") in terms of the best result (*i.e.*, the min value), the mean result (*i.e.*, the mean value), and the standard deviation result (*i.e.*, the std value). The parameter settings of GNO2DE, GNO2DE-A, and GNO2DE-B are given in **Table 2**. The experimental results are in detail summarized in **Tables 7** and **8**, and better results are highlighted in boldface. The optimized objective function values over 30 independent runs are arranged in ascending order and the 15th value in the list is called the median optimized function value.

From **Tables 7** and **8**, it is clearly shown that for each function of f_6, f_9 - f_{11}, f_{14}, f_{17} - f_{23}, the min, mean, and std values of GNO2DE are similar to those of

Table 6. Comparison between GNO2DE, and GNODE on functions f_{14} - f_{23}.

f	f_{min}	method	min	median	max	mean	std
f_{14}	0.998	GNO2DE	**0.99800383779445**	**0.99800383779445**	**0.99800383779445**	**0.99800383779445**	1.749353e−16
		GNODE	0.99800383779445	0.99800383779445	0.99800383779445	0.99800383779445	**1.129203e−16**
f_{15}	3.075e−4	GNO2DE	**3.074859878056e−4**	**3.074859878056e−4**	**3.074859878056e−4**	**3.074859878056e−4**	1.318374e−18
		GNODE	3.074859878056e−4	3.074859878056e−4	3.074859878056e−4	3.074859878056e−4	**1.055790e−19**
f_{16}	−1.0316	GNO2DE	**−1.03162845348988**	**−1.03162845348988**	−1.03162683969173	−1.03162838522757	3.024594e−7
		GNODE	−1.03162845348988	−1.03162845348988	**−1.03162845348988**	**−1.03162845348988**	**6.775215e−16**
f_{17}	0.397887	GNO2DE	**0.39788735772974**	**0.39788735772974**	**0.39788735772974**	**0.39788735772974**	**0**
		GNODE	0.39788735772974	0.39788735772974	0.39788735772974	0.39788735772974	0
f_{18}	3	GNO2DE	**2.99999999999992**	**2.99999999999992**	**2.99999999999992**	**2.99999999999992**	**1.953227e−15**
		GNODE	2.99999999999992	2.99999999999992	2.99999999999992	2.99999999999992	1.989449e−15
f_{19}	−3.86278	GNO2DE	**−3.86277751253760**	**−3.86277751253760**	**−3.86277751253760**	**−3.86277751253760**	**2.258405e−15**
		GNODE	−3.86277751253760	−3.86277751253760	−3.86277751253760	−3.86277751253760	2.258405e−15
f_{20}	−3.32237	GNO2DE	**−3.33539215295525**	−3.20321215577107	**−3.20321215577107**	−3.25167815473861	**6.478571e−2**
		GNODE	−3.33539215295525	**−3.33539215295525**	−3.20321215577107	**−3.27811415417544**	6.661963e−2
f_{21}	−10.1532	GNO2DE	**−10.15319967905823**	**−10.15319967905823**	**−10.15319967905823**	**−10.15319967905822**	**7.226896e−15**
		GNODE	−10.15319967905823	−10.15319967905823	−5.10077214033199	−9.81637117647648	1.281841951
f_{22}	−10.4029	GNO2DE	**−10.40294056681867**	**−10.40294056681867**	**−10.40294056681866**	**−10.40294056681866**	**1.615983e−15**
		GNODE	−10.40294056681867	−10.40294056681867	−10.40294056681866	−10.40294056681866	1.714009e−15
f_{23}	−10.5364	GNO2DE	**−10.53640981669205**	**−10.53640981669205**	**−10.53640981669205**	**−10.53640981669205**	**1.776357e−15**
		GNODE	−10.53640981669205	−10.53640981669205	−10.53640981669205	−10.53640981669205	1.806724e−15

GNO2DE-A, and GNO2DE-B, and three algorithms all can find the optimal solution.

Table 7 shows that for each function of f_1 - f_4, the optimal solution can be found by GNO2DE, GNO2DE-A, and GNO2DE-B, while the mean, std values of GNO2DE are slightly different from those of GNO2DE-A, and GNO2DE-B. For f_5, the mean, and std values of GNO2DE are obviously better than those of GNO2DE-A, and GNO2DE-B, while the min value of GNO2DE-A is worst among three algorithms. For f_7, the min value of GNO2DE-A is best among three algorithms, while its mean, and std values are worse or not better than those of GNO2DE, and GNO2DE-B. For f_8, the min, mean, and std values of GNO2DE-B are obviously worse than those of GNO2DE, and GNO2DE-A, while GNO2DE-A can obtained better mean, and std values than GNO2DE. For f_{12}, the min value of GNO2DE-A is worst among three algorithms, while the std value of GNO2DE is best among three algorithms. For f_{13}, the min value of GNO2DE-A is worst among three algorithms, while the mean, and std values of GNO2DE are best among three algorithms.

Table 8 shows that for f_{15}, the min, and mean values are approximate among three algorithms, while the std value of GNO2DE is best, that of GNO2DE-B is better, and that of GNO2DE-A is good. For f_{16}, the min, and mean values are similar among three algorithms, while the std value of GNO2DE-B is best, that of GNO2DE is better, and that of GNO2DE-A is good.

Therefore, from the above analysis, we know that the performance of GNO2DE is more stable than that of GNO2DE-A, and that of GNO2DE-B. This is because that GNO2DE employs two schemes "DE/bin/1/bin" and "DE/current to best/2/bin" to search the solution space. On the whole, GNO2DE can improve the search ability.

4.5. Comparison of GNO2DE with Some State-of-the-Art Algorithms

In this section, we compare GNO2DE with DE [3], ODE/2 [2], SOA [31], FEP [32], opt-IA [1], and CLPSO [15] in terms of the mean result *(i.e.*, the mean value), the standard deviation result (*i.e.*, the std value), and the

Table 7. Comparison between GNO2DE, GNO2DE-A, and GNO2DE-B on functions f_1 - f_{14} of dimensionality 30.

f	f_{min}	GNO2DE		GNO2DE-A		GNO2DE-B	
		min	mean ± std	min	mean ± std	min	mean ± std
f_1	0	**0**	2.55e−53 ± 1.1e−52	0	**0 ± 0**	0	4.2e−59 ± 1.6e−58
f_2	0	**0**	**0 ± 0**	0	0 ± 0	0	2.1e−34 ± 1.2e−33
f_3	0	**0**	**0 ± 0**	0	0 ± 0	0	2.8e−54 ± 1.5e−53
f_4	0	**0**	3.89e−25 ± 2.13e−24	0	**0 ± 0**	0	1.3e−23 ± 3.26e−23
f_5	0	1.4782e−16	**1.2658e−14 ± 2.1e−14**	4.3857	5.414 ± 4.4e−1	**0.0000**	4.6295 ± 5.0103
f_6	0	**0**	**0 ± 0**	0	0 ± 0	0	0 ± 0
f_7	0	1.2777e−4	1.3135e−3 ± 7.3e−4	**5.3502e−5**	1.3537e−3 ± 1.2e−3	1.3892e−4	**9.3933e−4 ± 4.2e−4**
f_8	−12569.487	**−12569.487**	−12569.469 ± 5.2e−2	−12569.487	**−12569.487 ± 2.3e−5**	−12230.228	−11423.429 ± 3.7e+2
f_9	0	**0**	**0 ± 0**	0	0 ± 0	0	0 ± 0
f_{10}	0	**8.8818e−16**	**8.8818e−16 ± 0**	8.8818e−16	8.8818e−16 ± 0	8.8818e−16	8.8818e−16 ± 0
f_{11}	0	**0**	**0 ± 0**	0	0 ± 0	0	0 ± 0
f_{12}	0	5.0000e−14	**1.0646e−2 ± 3.16e−2**	1.9663e−5	8.4895e−2 ± 1.25e−1	0	2.0875e−2 ± 1.13e−1
f_{13}	0	2.9163e−13	**2.1757e−12 ± 1.6e−12**	1.8552e−5	9.3345e−4 ± 5.87e−4	0	1.7522e−3 ± 4.8e−3

Table 8. Comparison between GNO2DE, GNO2DE-A, and GNO2DE-B on functions f_{14} - f_{23}.

f	f_{min}	GNO2DE		GNO2DE-A		GNO2DE-B	
		min	mean ± std	min	mean ± std	min	mean ± std
f_{14}	0.998	**0.9980038**	**0.9980038** ± 1.7e−16	0.9980038	0.9980038 ± 2.0e−16	0.9980038	0.9980038 ± **1.6e−16**
f_{15}	3.075e−4	**3.07486e−4**	**3.07486e−4 ± 1.3e−18**	3.07486e−4	3.07486e−4 ± 1.3e−10	3.07486e−4	3.07486e−4 ± 2.5e−16
f_{16}	−1.0316	**−1.031628**	−1.031628 ± 3.0e−7	−1.031628	−1.031626 ± 9.9e−6	−1.031628	**−1.031628 ± 5.1e−16**
f_{17}	0.397887	**0.397887**	**0.397887 ± 0**	0.397887	0.397887 ± 0	0.397887	0.397887 ± 0
f_{18}	3	**3.0000**	3.0000 ± 1.95e−15	3.0000	**3.0000 ± 1.7e−15**	3.0000	3.0000 ± 2.03e−15
f_{19}	−3.86278	**−3.862778**	**−3.862778 ± 2.3e−15**	−3.862778	−3.862778 ± 2.3e−15	−3.862778	−3.862778 ± 2.3e−15
f_{20}	−3.32237	**−3.33539**	−3.25168 ± 6.48e−2	−3.33539	−3.23846 ± **5.945e−2**	−3.33539	**−3.29029** ± 6.28e−2
f_{21}	−10.1532	**−10.1532**	**−10.1532 ± 7.2e−15**	−10.1532	−10.1532 ± 7.2e−15	−10.1532	−10.1532 ± 7.2e−15
f_{22}	−10.4029	**−10.4029**	−10.4029 ± 1.6e−15	−10.4029	**−10.4029 ± 1.04e−15**	−10.4029	−10.4029 ± 1.6e−15
f_{23}	−10.5364	**−10.5364**	**−10.5364 ± 1.8e−15**	−10.5364	−10.5364 ± 1.8e−15	−10.5364	−10.5364 ± 1.8e−15

number of fitness evaluations (*i.e.*, the FES value). The statistical results are summarized in **Tables 9** and **10**, and better results are highlighted in boldface. The experimental results of DE, SOA, and CLPSO are taken from [31], and the experimental results of ODE/2, FEP are taken from [2]. The optimized objective function values of 30 runs are arranged in ascending order and the 15th value in the list is called the median optimized function value. **Table 9** clearly shows that the mean, std values of GNO2DE are obviously superior to those of DE, ODE/2, SOA, FEP, opt-IA, and CLPSO on f_2, f_3, f_9 - f_{11}, while GNO2DE uses the least FES 100,000 among these methods, that the mean, std values of SOA are slightly better than those of GNO2DE, ODE/2, DE, etc. on f_1,

Table 9. Comparison between GNO2DE, DE, ODE/2, SOA, FEP, opt-IA, and CLPSO on functions f_1 - f_{14} of dimensionality 30.

f	item	GNO2DE	DE	ODE/2	SOA	FEP	opt-IA		CLPSO
							$\alpha = e^{(-\rho * f)}$	$\alpha = \left(\frac{1}{\rho}\right)e^{(-f)}$	
	mean	2.5519e−53	3.74e−13	2.06e−23	**1.02e−76**	5.7e−4	9.23e−12	1.7e−8	2.73e−12
f_1	std	1.0968e−52	3.94e−13	1.83e−23	**6.51e−76**	1.3e−4	2.44e−11	3.5e−15	1.68e−12
	FES	**100,000**	150,000	150,000	150,000	150,000	150,000	150,000	150,000
	mean	**0**	3.74e−9	1.43e−18	4.22e−63	8.1e−3	0.0	7.1e−8	3.82e−9
f_2	std	**0**	2.20e−9	8.11e−19	8.25e−63	7.7e−4	0.0	0.0	1.73e−9
	FES	**100,000**	200,000	200, 000	200,000	200,000	200,000	200,000	200,000
	mean	**0**	1.85e−10	5.25e−27	4.26e−25	1.6e−2	0.0	1.9e−10	4.20e−1
f_3	std	**0**	1.49e−10	9.66e−27	2.15e−24	1.4e−2	0.0	2.63e−10	3.62e−1
	FES	**100,000**	500,000	500,000	500,000	500,000	500,000	500,000	500,000
	mean	3.8886e−25	3.10e−2	2.72e−15	**1.02e−48**	0.3	1.0e−2	4.1e−2	2.05e−3
f_4	std	2.1299e−24	8.70e−2	9.30e−15	**2.46e−48**	0.5	5.3e−3	5.3e−2	1.25e−3
	FES	**100,000**	500,000	500,000	500,000	500,000	500,000	500,000	500,000
	mean	1.2658e−14	3.47e−31	**0**	2.54e+1	5.06	3.02	28.4	3.63e+1
f_5	std	2.1349e−14	2.45e−30	**0**	7.87e−1	5.87	12.2	0.42	3.12e+1
	FES	1000,000	2000,000	**428,776**	2000,000	2000,000	2000,000	2000,000	2000,000
	mean	0	0	**0**	0	0	0,2	0.0	0
f_6	std	0	0	**0**	0	0	0.44	0.0	0
	FES	100,000	150,000	**22,640**	150,000	150,000	150,000	150,000	150,000
	mean	1.3135e−3	4.66e−3	1.45e−3	**1.08e−4**	7.6e−3	3.0e−3	3.9e−3	2.98e−3
f_7	std	7.3416e−4	1.30e−3	4.20e−4	**6.44e−5**	2.6e−3	1.2e−3	1.3e−3	9.72e−4
	FES	**100,000**	300,000	300,000	300,000	300,000	300,000	300,000	300,000
	mean	−12569.4688	−11234	**−12569.4866**	−10126	−12554.5	−12508.38	−12568.27	−12271
f_8	std	5.1136e−2	455.5	**0**	669.5	52.6	155.54	0.23	177.8
	FES	300,000	900,000	**90, 381**	900,000	900,000	900,000	900,000	900,000
	mean	**0**	8.10e+1	0	0	4.6e−2	19.98	2.66	1.34e−9
f_9	std	**0**	3.23e+1	0	0	1.2e−2	7.66	2.39	8.57e−10
	FES	**100,000**	500,000	127,666	500,000	500,000	500,000	500,000	500,000
	mean	**8.8818e−16**	1.71e−7	4.67e−13	−4.44e−15	1.8e−2	18.98	1.1e−4	6.81e−6
f_{10}	std	**0**	7.66e−8	1.86e−13	0	2.1e−3	0.35	3.1e−5	1.94e−6
	FES	**100,000**	150,000	150,000	150,000	150,000	150,000	150,000	150,000
	mean	**0**	4.44e−4	0	0	1.6e−2	7.7e−2	4.55e−2	2.96e−4
f_{11}	std	**0**	1.77e−3	0	0	2.2e−2	8.63e−2	4.46e−2	1.46e−3
	FES	**100,000**	200,000	109, 853	200,000	200,000	200,000	200,000	200,000
	mean	1.0646e−2	3.67e−14	**6.73e−26**	1.28e−2	9.2e−6	0.137	3.1e−2	4.80e−11
f_{12}	std	3.1614e−2	4.07e−14	**9.27e−26**	7.62e−3	3.6e−6	0.23	5.7e−2	3.96e−11
	FES	**100,000**	150,000	150,000	150,000	150,000	150,000	150,000	150,000
	mean	2.1757e−12	2.91e−13	**4.37e−24**	1.89e−1	1.6e−4	1.51	3.20	6.42e−10
f_{13}	std	1.6271e−12	2.88e−13	**3.67e−24**	1.30e−1	7.3e−5	0.10	0.13	4.46e−10
	FES	**100, 000**	150,000	150,000	150,000	150,000	150,000	150,000	150,000

Table 10. Comparison between GNO2DE, DE, ODE/2, SOA, FEP, opt-IA, and CLPSO on functions f_{14} - f_{23}.

f	item	GNO2DE	DE	ODE/2	SOA	FEP	opt-IA		CLPSO
							$\alpha = e^{(-\rho^* f)}$	$\alpha = \left(\frac{1}{\rho}\right) e^{(-f)}$	
	mean	**0.9980038**	0.998	0.998	1.199	1.22	1.02	1.21	0.998
f_{14}	std	1.7494e−16	2.88e−16	**0**	5.30e−1	0.56	7.1e-2	0.54	5.63e−10
	FES	100,000	10,000	**9552**	10,000	10,000	10,000	10,000	10,000
	mean	**3.07486e−4**	4.7231e−2	3.08e−4	3.0749e−4	5.0e−4	7.1e−4	7.7e−3	5.3715e−4
f_{15}	std	1.3184e−18	3.55e−4	**0**	1.58e−9	3.2e−4	1.3e−4	1.4e−2	6.99e−5
	FES	100, 000	400, 000	**32,430**	400, 000	400, 000	400, 000	400, 000	400, 000
	mean	**−1.031628**	−1.0316	−1.03163	−1.0316	−1.031	−1.03158	−1.02	−1.0316
f_{16}	std	3.0246e−7	6.77e−13	**0**	6.73e−6	4.9e−7	1.5e−4	1.1e−2	8.50e−14
	FES	100,000	10,000	**10,000**	10,000	10,000	10,000	10,000	10,000
	mean	**0.397887**	0.39789	0.39789	0.39838	0.398	0.398	0.450	0.39789
f_{17}	std	**0**	1.14e−8	2.01e−10	5.14e−4	1.5e−7	2.0e−4	0.21	1.08e−13
	FES	100,000	10,000	10,000	10,000	10,000	10,000	10,000	**10,000**
	mean	**3.0000**	3	3.00	3.0001	3.02	3.0	3.0	3
f_{18}	std	1.9532e−15	3.31e−15	**0**	1.17e−4	0.11	0.0	0.0	5.54e−13
	FES	100,000	10,000	**10,000**	10,000	10,000	10,000	10,000	10,000
	mean	**−3.8627775**	−3.8628	−3.86278	−3.8621	−3.86	−3.72	−3.72	−3.8628
f_{19}	std	2.2584e−15	**1.97e−15**	2.68e−15	6.69e−4	1.4e−5	1.1e−4	1.1e−2	6.07e−12
	FES	100,000	**10,000**	10,000	10,000	10,000	10,000	10,000	10,000
	mean	−3.251678	−3.215	**−3.322**	−3.298	−3.27	−3.31	−3.31	−3.274
f_{20}	std	6.4786e−2	3.6e−2	**1.13e−12**	4.5e−2	5.9e−2	7.4e−2	5.9e−3	5.9e−2
	FES	100, 000	20, 000	**20,000**	20, 000	20,000	20,000	20,000	20,000
	mean	**−10.1531997**	−10.15	−10.1532	−9.67	−5.52	−9.11	−5.36	−9.57
f_{21}	std	**7.2269e−15**	4.67e−6	1.04e−6	4.96e−1	1.59	1.82	2.20	4.28e−1
	FES	100,000	10,000	**10,000**	10,000	10,000	10,000	10,000	10,000
	mean	**−10.40294**	−10.40	−10.40294	−9.79	−5.52	−9.86	−5.34	−9.40
f_{22}	std	**1.61598e−15**	2.07e−7	2.49e−8	4.48e−1	2.12	1.88	2.11	1.12
	FES	100,000	10,000	**10,000**	10,000	10,000	10,000	10,000	10,000
	mean	**−10.5364098**	−10.54	−10.53641	−9.72	−6.57	−9.96	−6.03	−9.47
f_{23}	std	**1.7764e−15**	3.21e−6	2.35e−8	4.72e−1	3.14	1.46	2.66	1.25
	FES	100,000	10,000	**10,000**	10,000	10,000	10,000	10,000	10,000

f_4, f_7, while the 100,000 FES of GNO2DE is least among these methods, and that the mean, std, FES values of GNO2DE are better than DE, ODE/2, FEP, opt-IA, and CLPSO. For f_5, f_6, f_8, the mean, std, and FES values of ODE/2 are not worse than or better than those of other methods. For f_5, compared with DE, GNO2DE is slightly worse in terms of the mean, std values of, while the 200,000 FES of DE is twice of that of

GNO2DE, and the performance of GNO2DE is clearly better than that of SOA, FEP, opt-IA, and CLPSO. For f_6, the mean, std values of GNO2DE are similar to those of DE, SOA, FEP, opt-IA, and CLPSO, while the 100,000 FES used by GNO2DE is least among these methods. For f_8, the mean, std, FES values of GNO2DE are obviously better than those of DE, SOA, FEP, opt-IA, and CLPSO. For f_{12}, f_{13}, the min, std values of ODE/2 are better than those of other methods, while the 100,000 FES used by GNO2DE is least among these methods.

As shown in **Table 10**, the mean value of GNO2DE and SOA on f_{15} is approximate and are better than other methods, while ODE/2 has the least std, and FES values. For f_{16} - f_{18}, all methods have the similar per- formance. For f_{19}, GNO2DE, DE, ODE/2, SOA, CLPSO have the similar performance and are better than FEP. For f_{20}, the performance of ODE/2 and FEP is better than other methods. For f_{21} - f_{23}, the perform- ance of GNO2DE, DE, and ODE/2 is approximate and are better than other methods.

In sum, the mean and standard deviation results of GNO2DE are not worse than or superior to DE, ODE/2, SOA, FEP, opt-IA, and CLPSO on a test set of benchmark functions. GNO2DE uses the opposite point method, employs two DE schemes "DE/rand/1/bin" and "DE/current to best/2/bin", and introduces non-uniform crossover rate. These techniques are beneficial to enhancing the performance of GNO2DE.

4.6. Experimental Results of 100-Dimensional Functions

In this section, the statistical results of GNO2DE on 100-dimensional functions are given in **Table 11**. The parameter setup is used in **Table 4**. The optimized objecttive function values over 30 independent runs are arranged in ascending order and the 15th value in the list is called the median optimized function value. **Table 11** clearly shows that GNO2DE can find the optimum or near optimum of each 100-dimensional function of f_1 - f_{13}, and that GNO2DE can obtain the stable performance of each function of f_1 - f_7, f_9 - f_{11}, while it performs slightly worse on f_8, f_{12}, f_{13}. Therefore, when used for solving high dimensional global numerical optimization problems, NGO2DE also performs well.

5. Conclusion and Future Work

This paper introduces an opposition-based DE algorithm for global numerical optimization (GNO2DE). GNO2DE uses the method of opposition-based learning to utilize the existing search spaces to improve the convergence speed, employs adaptive DE schemes and non-uniform crossover to enhance the adaptive search ability. Numerical results show that GNO2DE outperforms some state-of-the-art algorithms. However, there are still some possible things to do in the future: 1) further, to improve the self-adaptation of the control parameters such as the scaling factor *F*; 2) to test higher dimensional global nu-

Table 11. Experimental Results of 100-dimensional functions f_1 - f_{13}.

f	f_{min}	min	median	max	mean	std	FES
f_1	0	0	0	0	0	0	5×10^5
f_2	0	0	0	0	0	0	5×10^5
f_3	0	0	0	0	0	0	5×10^5
f_4	0	0	0	0	0	0	5×10^5
f_5	0	3.497698e−19	7.195169e−18	5.756444e−15	4.662977e−16	1.318697e−15	5×10^6
f_6	0	0	0	0	0	0	5×10^5
f_7	0	0	0	0	0	0	5×10^5
f_8	−41898.29	−41898.288727	−41898.288727	−41779.850393	−41882.496949	40.949569	3×10^6
f_9	0	0	0	0	0	0	5×10^5
f_{10}	0	8.881784197e−16	8.881784197e−16	8.881784197e−16	8.881784197e−16	0	5×10^5
f_{11}	0	0	0	0	0	0	5×10^5
f_{12}	0	0.000000000000	7.334999e−2	3.530528e−1	1.226082e−1	1.371314e−1	5×10^5
f_{13}	0	0.000000000000	1.098737e−2	9.888265e−2	2.072484e−2	2.975114e−2	5×10^5

merical optimization problems; 3) to introduce some local search and heuristic techniques to speed up the convergence and escape from the local optima, etc.

REFERENCES

[1] V. Cutello, G. Narzisi, G. Nicosia and M. Pavone, "An Immunological Algorithm for Global Numerical Optimization," *Artificial Evolution*: 7*th International Conference*, *Evolution Artificielle*, Lecture Notes in Computer Science Vol. 3871, 2006, pp. 284-295. doi:10.1007/11740698_25

[2] W. Gong, Z. Cai and L. Jiang, "Enhancing the Performance of Differential Evolution Using Orthogonal Design Method," *Applied Mathematics and Computation*, Vol. 206, No. 1, 2008, pp. 56-69. doi:10.1016/j.amc.2008.08.053

[3] R. Storn and K. Price, "Differential Evolution—A Simple and Efficient Heuristic for Global Optimization over Continuous Spaces," *Journal of Global Optimization*, Vol. 11, No. 4, 1997, pp. 341-359. doi:10.1023/A:1008202821328

[4] J. Sun, Q. Zhang and E. P. K. Tsang, "DE/EDA: A New Evolutionary Algorithm for Global Optimization," *Information Sciences*, Vol. 169, No. 3-4, 2005, pp. 249-262.

[5] M. M. Ali, C. Storey and A. Torn, "Application of Some Recent Stochastic Global Optimization Algorithms to Practical Problems," TUCS Technical Report No. 47, Turku Centre for Computer Science, Turku, 1996.

[6] H. P. Schwefel, "Numerical Optimization of Computer Models," John Wiley & Sons, Chichester, 1981.

[7] J. H. Holland, "Adaptation in Natural and Artificial Systems," University of Michigan Press, Ann Arbor, 1975.

[8] I. Rechenberg, "Evolution Strategy: Optimization of Technical Systems by Means of Biological Evolution," Fromman-Holzboog, Stuttgart, 1973.

[9] J. R. Koza, "Genetic Programming: On the Programming of Computers by Means of Natural Selection," The MIT Press, Cambridge, 1992.

[10] D. B. Fogel, "Applying Evolutionary Programming to Selected Traveling Salesman Problems," *Cybernetics and Systems*, Vol. 24, No. 1, 1993, pp. 27-36. doi:10.1080/01969729308961697

[11] K. E. Parsopoulos and M. N. Vrahatis, "Recent Approaches to Global Optimization Problems through Particle Swarm Optimization," *Natural Computing*, Vol. 1, No. 2-3, 2002, pp. 235-306. doi:10.1023/A:1016568309421

[12] J. Kennedy and R. C. Eberhart, "Particle Swarm Optimization," *Proceedings of the* 1995 *IEEE International Conference on Neural Networks*, Vol. 4, Perth, 27 November-1 December 1995, pp. 1942-1948. doi:10.1109/ICNN.1995.488968

[13] D. Karaboğa and S. Ökdem, "A Simple and Global Optimization Algorithm for Engineering Problems: Differential Evolution Algorithm," *Turk Journal of Electrical Engineering*, Vol. 12, No. 1, 2004, pp. 53-60.

[14] J. Vesterstrom and R. Thomsen, "A Comparative Study of Differential Evolution, Particle Swarm Optimization, and Evolutionary Algorithms on Numerical Benchmark Problems," 2004 *IEEE Congress on Evolutionary Computation*, Vol. 2, Portland, 19-23 June 2004, pp. 1980-1987.

[15] J. J. Liang, A. K. Qin, P. N. Suganthan and S. Baskar, "Comprehensive Learning Particle Swarm Optimizer for Global Optimization of Multi-Modal Functions," *IEEE Transactions on Evolutionary Computation*, Vol. 10, No. 3, 2006, pp. 281-295. doi:10.1109/TEVC.2005.857610

[16] R. Storn and K. Price, "Differential Evolution—A Simple and Efficient Adaptive Scheme for Global Optimization over Continuous Spaces," Technical Report TR-95-012, International Computer Science Institute, Berkeley, 1995.

[17] K. Price, R. Storn and J. Lampinen, "Differential Evolution: A Practical Approach to Global Optimization," Springer-Verlag, Berlin, 2005.

[18] J. Brest, S. Greiner, B. Boskovic, M. Mernik and V. Zumer, "Self-Adapting Control Parameters in Differential Evolution: A Comparative Study on Numerical Benchmark Problems," *IEEE Transactions on Evolutionary Computation*, Vol. 10, No. 6, 2006, pp. 646-657. doi:10.1109/TEVC.2006.872133

[19] A. K. Qin and P. N. Suganthan, "Self-Adaptive Differential Evolution Algorithm for Numerical Optimization," *Proceedings of the* 2005 *IEEE Congress on Evolutionary Computation*, Vol. 2, 2005, pp. 1785-1791. doi:10.1109/CEC.2005.1554904

[20] S. Das, A. Abraham, U. K. Chakraborty and A. Konar, "Differential Evolution Using a Neighborhood-Based Mutation Operator," *IEEE Transactions on Evolutionary Computation*, Vol. 13, No. 3, 2009, pp. 526-553. doi:10.1109/TEVC.2008.2009457

[21] S. Rahnamayan and G. G. Wang, "Solving Large Scale Optimization Problems by Opposition-Based Differential Evolution (ODE)," *WSEAS Transactions on Computers*, Vol. 7, No. 10, 2008, pp. 1792-1804.

[22] S. Rahnamayan, H. R. Tizhoosh and M. M. A. Salama, "Opposition-Based Differential Evolution," *IEEE Transactions on Evolutionary Computation*, Vol. 12, No. 1, 2008, pp. 64-79. doi:10.1109/TEVC.2007.894200

[23] S. Rahnamayan, H. R. Tizhoosh and M. M. A. Salama, "Opposition versus Randomness in Soft Computing Techniques," *Elsevier Journal on Applied Soft Computing*, Vol. 8, No. 2, 2008, pp. 906-918. doi:10.1016/j.asoc.2007.07.010

[24] H. R. Tizhoosh, "Opposition-Based Reinforcement Learning," *Journal of Advanced Computational Intelligence and Intelligent Informatics*, Vol. 10, No. 4, 2006, pp. 578-585.

[25] H. A. Abbass, R. Sarker and C. Newton, "PDE: A Pareto-frontier Differential Evolution Approach for Multi-Objective Optimization Problems," 2001 *IEEE Congress on Evolutionary Computation*, Vol. 2, Seoul, 27-30 May 2001, pp. 971- 978.

[26] M. Ali, M. Pant and V. P. Singh, "Two Modified Differential Evolution Algorithms and Their Applications to

Engineering Design Problems," *World Journal of Modelling and Simulation*, Vol. 6, No. 1, 2010, pp.72-80.

[27] Z. Y. Yang, K. Tang and X. Yao, "Self-Adaptive Differential Evolution with Neighborhood Search," 2008 *Congress on Evolutionary Computation*, Hong Kong, 1-6 June 2008, pp. 1110-1116.

[28] Z. Michalewicz, "Genetic Algorithms + Data Structures = Evolution Programs," 3rd Edition, Springer, Berlin, 1996.

[29] K. Zielinski, P. Weitkemper, R. Laur and K.-D. Kammeyer, "Examination of Stopping Criteria for Differential Evolution Based on a Power Allocation Problem," *Proceedings of the* 10*th International Conference on Optimization of Electrical and Electronic Equipment*, Vol. 3, Brasov, 18-19 May 2006, pp. 149-156.

[30] Y. Ao and H. Chi, "An Adaptive Differential Evolution Algorithm to Solve Constrained Optimization Problems in Engineering Design," *Engineering*, Vol. 2, No. 1, 2010, pp. 65-77. doi:10.4236/eng.2010.21009

[31] C. Dai, W. Chen, Y. Song and Y. Zhu, "Seeker Optimization Algorithm: A Novel Stochastic Search Algorithm for Global Numerical Optimization," *Journal of Systems Engineering and Electronics*, Vol. 21, No. 2, 2010, pp. 300-311.

[32] X. Yao, Y. Liu and G. Lin, "Evolutionary Programming Made Faster," *IEEE Transactions on Evolutionary Computation*, Vol. 3, No. 2, 1999, pp. 82-102. doi:10.1109/4235.771163

[33] A.-R. Hedar and M. Fukushima, "Directed Evolutionary Programming: Towards an Improved Performance of Evolutionary Programming," 2006 *IEEE Congress on Evolutionary Computation*, Vancouver, 11 September 2006, pp. 1521-1528.

Appendix

$$f_{12}: y_i = 1+\frac{1}{4}\left(x_i+1\right), \quad f_{12}-f_{13}: u\left(x_i,a,k,m\right)=\begin{cases} k\left(x_i-a\right)^m, & x_i > a; \\ 0, & -a \le x_i \le a; \\ k\left(-x_i-a\right)^m, & x_i < -a. \end{cases}$$

$$f_{14}: a=\begin{bmatrix} -32 & -32 \\ -16 & -32 \\ 0 & -32 \\ 16 & -32 \\ 32 & -32 \\ -32 & -16 \\ -16 & -16 \\ 0 & -16 \\ 16 & -16 \\ 32 & -16 \\ -32 & 0 \\ -16 & 0 \\ 0 & 0 \\ 16 & 0 \\ 32 & 0 \\ -32 & 16 \\ -16 & 16 \\ 0 & 16 \\ 16 & 16 \\ 32 & 16 \\ -32 & 32 \\ -16 & 32 \\ 0 & 32 \\ 16 & 32 \\ 32 & 32 \end{bmatrix}^T, \quad f_{15}: a=\begin{bmatrix} 0.1957 \\ 0.1947 \\ 0.1735 \\ 0.1600 \\ 0.0844 \\ 0.0627 \\ 0.0456 \\ 0.0342 \\ 0.0323 \\ 0.0235 \\ 0.0246 \end{bmatrix}^T, \quad b=\begin{bmatrix} 4 \\ 2 \\ 1 \\ \frac{1}{2} \\ \frac{1}{4} \\ \frac{1}{6} \\ \frac{1}{8} \\ \frac{1}{10} \\ \frac{1}{12} \\ \frac{1}{14} \\ \frac{1}{16} \end{bmatrix}^T$$

$$f_{19}: a=\begin{bmatrix}3.0 & 10 & 30\\ 0.1 & 10 & 35\\ 3.0 & 10 & 30\\ 0.1 & 10 & 35\end{bmatrix},\ p=10^{-4}\begin{bmatrix}6890 & 1170 & 2673\\ 4699 & 4387 & 7470\\ 1091 & 8732 & 5547\\ 381 & 5743 & 8828\end{bmatrix},\ c=[1,1.2,3,3.2]^T$$

$$f_{20}: a=\begin{bmatrix}10 & 3 & 17 & 3.05 & 1.7 & 8\\ 0.05 & 10 & 17 & 0.1 & 8 & 14\\ 3 & 3.5 & 1.7 & 10 & 17 & 8\\ 17 & 8 & 0.05 & 10 & 0.1 & 14\end{bmatrix},\ c=\begin{bmatrix}1\\ 1.2\\ 3\\ 3.2\end{bmatrix},\ p=10^{-4}\begin{bmatrix}1312 & 1696 & 5569 & 124 & 8283 & 5886\\ 2329 & 4135 & 8307 & 3736 & 1004 & 9991\\ 2348 & 1451 & 3522 & 2883 & 3047 & 6650\\ 4047 & 8828 & 8732 & 5743 & 1091 & 381\end{bmatrix}$$

$$f_{21}-f_{23}: a=\begin{bmatrix}4.0 & 4.0 & 4.0 & 4.0\\ 1.0 & 1.0 & 1.0 & 1.0\\ 8.0 & 8.0 & 8.0 & 8.0\\ 6.0 & 6.0 & 6.0 & 6.0\\ 3.0 & 7.0 & 3.0 & 7.0\\ 2.0 & 9.0 & 2.0 & 9.0\\ 5.0 & 5.0 & 3.0 & 3.0\\ 8.0 & 1.0 & 8.0 & 1.0\\ 6.0 & 2.0 & 6.0 & 2.0\\ 7.0 & 3.6 & 7.0 & 3.6\end{bmatrix},\ c=\begin{bmatrix}0.1\\ 0.2\\ 0.2\\ 0.4\\ 0.4\\ 0.6\\ 0.3\\ 0.7\\ 0.5\\ 0.5\end{bmatrix}^T$$

