J Comb Optim (2016) 31:546–576
DOI 10.1007/s10878-014-9773-6

Modified differential evolution with self-adaptive parameters method

Xiangtao Li · Minghao Yin

Published online: 22 July 2014

Abstract The differential evolution algorithm (DE) is a simple and effective global optimization algorithm. It has been successfully applied to solve a wide range of real-world optimization problem. In this paper, the proposed algorithm uses two mutation rules based on the rand and best individuals among the entire population. In order to balance the exploitation and exploration of the algorithm, two new rules are combined through a probability rule. Then, self-adaptive parameter setting is introduced as uniformly random numbers to enhance the diversity of the population based on the relative success number of the proposed two new parameters in a previous period. In other aspects, our algorithm has a very simple structure and thus it is easy to implement. To verify the performance of MDE, 16 benchmark functions chosen from literature are employed. The results show that the proposed MDE algorithm clearly outperforms the standard differential evolution algorithm with six different parameter settings. Compared with some evolution algorithms (ODE, OXDE, SaDE, JADE, jDE, CoDE, CLPSO, CMA-ES, GL-25, AFEP, MSAEP and ENAEP) from literature, experimental results indicate that the proposed algorithm performs better than, or at least comparable to state-of-the-art approaches from literature when considering the quality of the solution obtained.

Keywords Differential evolution · Success rate · Self-adaptive · Numerical optimization · Evolutionary algorithms

X. Li · M. Yin (✉)
College of Computer Science, Northeast Normal University, Changchun 130117, People's Republic of China
e-mail: Minghao.Yin1@gmail.com

X. Li
e-mail: lixt314@nenu.edu.cn

1 Introduction

Many real-world problems may be formulated as optimization problems with variables in continuous domains. In the past decade, we have viewed different kinds of evolutionary algorithms advanced to solve optimization problems, such as genetic algorithm (GA), particle swarm optimization algorithm (PSO), estimation of distribution algorithms (EDA), ant colony optimization (ACO), simulated annealing (SA), biogeography based optimization (BBO), differential evolution (DE), artificial bee colony (ABC), and cuckoo search algorithm (CS) (Suman 2004; Horn et al. 1994; Zhang and Muhlenbein 2004; Clerc and Kennedy 2002; Dorigo et al. 1996; Simon 2008; Storn and Price 1997; Yang and Deb 2009).

Recently, differential evolution algorithm (Storn and Price 1997) has been proposed as a simple and powerful population-based stochastic optimization method, which is originally motivated by the mechanism of the natural selection. This algorithm searches solutions using three basic operators: mutation, crossover and greedy selection. Mutation is used to generate a mutation vector by adding differential vectors obtained from the difference of several randomly chosen parameter vectors to the parent vector. After that, crossover operation generates the trial vector by combining the parameters of the mutation vector with the parameters of a parent vector selected from the population. Finally, according to the fitness value, selection operation determines which of the vectors should be chosen for the next generation by implementing a one-to-one completion between the generated trail vectors and the corresponding parent vectors. In order to accelerate the convergence speed and avoid trapping in the local optima, several variations of DE have been proposed to enhance the performance of the standard DE recently. Moreover, DE has been proved to be quite efficient when solving real-world problems. Similar with other evolutionary algorithms, DE also has many disadvantages. For example, while the global exploration ability is considered adequate, its local exploitation ability is regarded weak and its convergence velocity is too low. In the other aspects, the performance of the DE algorithm is sensitive to the mutation strategy and respective control parameters such as the population size, crossover rate and scale factor. The best setting for the control parameter is different for different problems. Therefore, in order to successfully solve a complex problem, the parameter setting and the mutation strategy are very important (Das and Suganthan 2011).

In fact, the performance of the conventional DE algorithm highly depends on the mutation and crossover operations. Being fascinated by the prospect and potential of DE, recently, many researches are working on the improvement of DE, and many variants of the new algorithm are presented. A brief overview of these algorithms is proposed in this section.

Brest et al. (2006) proposed a self adaptive parameter setting in differential evolution algorithm in order to avoid the manual parameter setting of F and CR. The parameter control technique is based on the self adaptation of two parameters associated with the evolutionary process. The main goal here is to product a flexible DE, in terms of control parameters F and CR. The result shows that the algorithm with self adaptive control parameter setting is better than, or at least comparable to, the standard DE algorithm and other evolutionary algorithms from literature.

Rahnamayan et al. (2008) proposed the opposition based Differential evolution (ODE) employing the logic based on the opposition points in order to enhance search properties of DE and test a wide portion of the decision space. The ODE algorithm consists of a DE framework and two opposition based components: the first after the initial sampling and the second after the survivor selection scheme.

Liu and Lampinen (2005) introduced a new version of the differential evolution algorithm with control parameters, called the fuzzy adaptive differential evolution algorithm. The algorithm uses the fuzzy logic controllers to adapt the parameters.

Qin et al. (2009) proposed a self adaptive DE algorithm (SaDE), in which both trail vector generation strategies and their associated control parameter values were gradually self-adaptive by learning from their previous experiences in generating promising solutions. This method does not use any particular learning strategy, nor any specific setting for the control parameters F and CR. From the results, we can conclude that SaDE is more effective in obtaining better quality solution, which are more stable with the relatively smaller standard deviation and higher success rate.

Zhang and Sanderson (2009) proposed a novel algorithm, named JADE, to improve the performance of the differential evolution algorithm by proposing a novel mutation strategy "DE/current-to-pbest/1" and then the self adaptive parameters were used to update the parameters. Experiment results show that JADE is better than, or at least comparable to other algorithms in terms of the solutions.

Ghosh et al. (2011) proposed an effective adaptation technique which suggested a novel automatic tuning method for the scale factor and crossover rate of population members in DE. This technique is proposed based on their objective function values. Comparison with the best-known and expensive variants of DE over fourteen well-known numerical benchmarks and a real-life engineering problem reflects the superiority of proposed parameter tuning scheme in terms of the accuracy, convergence speed, and robustness.

Das et al. (2009) described a family of improved variants of the DE/target-to-best/1/bin scheme, which utilized the concept of the neighborhood of each population member. In order to balance the exploration and exploitation of the algorithm, a new hybrid mutation strategy is proposed. The local mutation model was the explorative mutation operator and the global mutation model is the exploitative mutation operator. Experiment results showed that the algorithm was better than other algorithms on a test suite of 24 test problems and two real-life problems.

Wang et al. (2011a) proposed a new composite differential evolution algorithm, which was improved by combining several effective trial vector generation strategies with some suitable control parameter settings. This algorithm used three trail generator methods and three different parameter settings. CoDE has been tested on all the CEC2005 test problems. Experimental results showed that the CoDE algorithm was very competitive.

In other aspects, some studies on hybrid algorithms for DE found that the performance of hybrid DE algorithms have a better solution quality with in a given number of objective function evaluations. Therefore, there has been an increasing interest in combining DE with other algorithms. Gong et al. (2010) proposed a hybrid DE with BBO namely DE/BBO, for the global numerical optimization problem. The algorithm combines the exploration of DE with the exploitation of BBO effectively.

The experimental results show that the algorithm can obtain the high quality of the final solutions and the high convergence rate. Sun et al. (2004) proposed a combination of DE algorithms and the estimation of distribution algorithm (EDA), in which new promising solution was created by DE/EDA. This algorithm used a probability model to determine promising regions in order to focus the search process on those areas. The presented experimental results showed that the DE/EDA outperformed DE and EDA. Noman and Iba (2008) proposed a crossover-based adaptive local search operation for enhancing the performance of standard differential evolution algorithm; this algorithm combined the DE with fittest individual refinement (FIR). The FIR scheme accelerated DE by applying a fixed-length crossover-based in the neighborhood of the best solution in each generation. Gong et al. (2008) proposed an improved version of DE, namely orthogonal based DE. This algorithm employed the two level orthogonal crossovers to improve the performance of DE. The results indicated that orthogonal based DE is able to find the optimal or close-to-optimal in all cased. Omran et al. (2007) proposed the barebones differential evolution that is a new, almost parameter-free optimization algorithm that is a hybrid of the barebones particle swarm optimization and differential evolution. DE was used to mutate for each individual, the attractor associated with that particle, defined as a weighted average of its personal and neighborhood best positions. Neri and Tirronen (2009) proposed the scale factor local search differential evolution. This algorithm was a differential evolution based memetic algorithm which employs, within a self adaptive scheme, two local search algorithms. These local search algorithms aimed at detecting a value of the scale factor corresponding to an offspring with a high performance. A statistical analysis of the optimization results has been included in order to compare the results in terms of final solution detected and convergence speed. Yang et al. (2008) proposed the neighborhood search differential evolution. In this algorithm, the scale factor was adjusted by the sampling value from probability distributions, and the mutation was updated by a logic inspired by evolutionary programming.

Practically, based on the analysis of the above introduction, it can easy to see that the control parameters and the new mutation strategy are the major modifications. However, compared with these two parts, the new mutation strategy seems little to enhance the local search ability of DE or to overcome the convergence rate of the algorithms. Then, proposing the new control parameters and the new mutation model are still an open challenge direction of research. Therefore, in this paper, we propose a new differential evolution algorithm which uses a new self adaptive crossover rate depending on the success rate of the crossover probability. In order to balance the exploration and exploitation of the algorithm, a probability rule is used to combine two different mutation rules to enhance the diversity of the population and the convergence rate of the algorithm. In other aspect, our algorithm has a very simple structure and thus it is easy to implement. To verify the performance of MDE, 16 benchmark functions chosen from literature are employed. Compared with other evolution algorithms from literature, experimental results indicate that the proposed algorithm performs better than, or at least comparable to state-of-the-art approaches from literature when considering the quality of the solution obtained.

2 Basic algorithm

2.1 Differential evolution algorithm

Differential Evolution (DE) is a fairly novel population based search heuristic which is simple to implement and requires little parameter tuning compared to other search heuristics in continuous space. Different from other algorithms, DE uses distance and direction information from the current population to guide the search process. The crucial idea behind DE is a scheme for producing trial vectors according to the manipulation of target vector and difference vector. If the trail vector yields a worse fitness than a predetermined population member, the newly trail vector will be accepted and compared in the following generation. Sometimes, when handling the single problem and multi-objective problem, the algorithm is superior to other algorithms, such as genetic algorithm and particle swarm optimization. The logic of standard differential evolution algorithm can be described as follows:

Algorithm 1. Algorithm description of DE algorithm

Input: *NP* is the population size. *G* is the number of the iteration

Output: The best solutions.

1: **begin**

2: Set the generation counter *G*=0; and randomly initialize a population of *NP* individuals X_i . Initialize the parameter *F*, *CR*

3: Evaluate the fitness for each individual in *P*.

4: **while** stopping criteria is not satisfied do

5: **for** i= 1 to *NP* **do**

6: select randomly $a \neq b \neq c \neq d \neq i$

7: **for** j=1 to *D* **do**

8: $j_{rand} = \lfloor rand(0,1) * D \rfloor$

9: **If** rand (0, 1) $\leq$ CR or j== j_{rand} **then**

10: $u_{i,j} = x_{a,j} + F \times (x_{b,j} - x_{c,j})$

11: /* the mutation strategies*/

12: **Else**

13: $u_{i,j} = x_{i,j}$

14: **end if**

15: **end for**

16: **end for**

17: **for** i=1 to *NP* **do**

18: Evaluate the offspring u_i

19: **If** u_i is better than X_i **then**

20: $X_i = u_i$

21: **end if**

22: **end for**

23: Memorize the best solution achieved so far

24: **end while**

25: **end**

As can be seen in pseudo-code, DE can be summarized into three major steps: mutation, crossover and selection. In the beginning of the algorithm, the algorithm begins with a randomly initiated population which generates $NP*D$ matrix with uniform probability distribution random values. We can generate the jth component of the ith vector as

$$x_{j,i,0} = x_{j,\min} + rand_{i,j}[0, 1] \cdot (x_{j,\max} - x_{j,\min}) \tag{1}$$

where $rand_{i,j}[0, 1]$ is a uniformly distribution random number between 0 and 1. $i = 1, \ldots, NP$ and $j = 1, \ldots, D$. $x_{j,\max}$ and $x_{j,\min}$ are the upper bound and lower bound of the jth column, respectively.

After initialization, mutation vectors $V_{i,G}$ are generated according to each population member or target vector $X_{i,G}$ in current population. In the standard DE algorithm, five different mutation strategies can be used with one or two different crossover methods, which are listed in the followings:
"DE/rand/1/bin"

$$V_{i,G} = X_{r_1,G} + F \cdot (X_{r_2,G} - X_{r_3,G})$$

"DE/best/1/bin"

$$V_{i,G} = X_{best,G} + F \cdot (X_{r_1,G} - X_{r_2,G})$$

"DE/current-to-best/1/bin"

$$V_{i,G} = X_{i,G} + F \cdot (X_{best,G} - X_{i,G}) + F \cdot (X_{r_1,G} - X_{r_2,G})$$

"DE/best/2/bin"

$$V_{i,G} = X_{best,G} + F \cdot (X_{r_1,G} - X_{r_2,G}) + F \cdot (X_{r_3,G} - X_{r_4,G})$$

"DE/rand/2/bin"

$$V_{i,G} = X_{r_1,G} + F \cdot (X_{r_2,G} - X_{r_3,G}) + F \cdot (X_{r_4,G} - X_{r_5,G}) \tag{2}$$

where $r_1, r_2, r_3, r_4, r_5 \in [1, \ldots, NP]$ are randomly chosen integers, and $r_1 \neq r_2 \neq r_3 \neq r_4 \neq r_5 \neq i$. F is a mutation control parameter which affects the disturbance added by the weight of different vectors. $X_{best,G}$ is the best individual with

the best fitness in the current population at generation G. In this paper, we use the "DE/rand/1/bin" mutation method to generate the offspring vector in the basic DE algorithm.

In the crossover operation, a recombination of the offspring vector $V_{i,G}$ and the parent vector $X_{i,G}$ produces a trail vector $U_{i,G} = [u_{1,i,G}, u_{2,i,G}, u_{3,i,G}, \ldots, u_{D,i,G}]$. Usually the binomial crossover is accepted, which is defined as follows:

$$u_{j,i,G} = \begin{cases} v_{j,i,G}, (rand_j[0,1] \leq CR) \, or \, (j = j_{rand}) \\ x_{j,i,G}, otherwise \end{cases} \tag{3}$$

where $j = [1, \ldots, D]$; $rand_{i,j}[0, 1]$ is a random number between (0,1); $j_{rand} = [1, \ldots, D]$ is the randomly chosen index, CR is the crossover rate and $v_{j,i,G}$ is the difference vector of the jth particle in the ith dimension at the Gth iteration, and $u_{j,i,G}$ denotes the trail vector of the jth individual in the ith dimension at the Gth iteration.

Selection operation is used to choose the next population (i.e. $G = G + 1$) between the trail population and the target population. The selection operation can be described as follows:

$$\begin{aligned} X_{i,G+1} &= U_{i,G}, \quad \text{If } f(U_{i,G}) \leq f(X_{i,G}) \\ &= x_{i,G}, \quad \text{If } f(U_{i,G}) > f(X_{i,G}) \end{aligned} \tag{4}$$

3 Modified differential evolution algorithm

In order to apply the differential evolution algorithm to more complex optimization problems, the further improvement in performance is necessary. In this paper, we introduce an efficient algorithm named modified differential evolution algorithm (MDE) to get the global optimization solution of high quality for the numerical optimization problems. In this section, we will use two mutation rules: DE/rand/2/bin and DE/rand to pbest/1/bin (Zhang and Sanderson 2009). These rules are used alternately through a probability rule. In the standard DE, scale factor F and crossover rate CR are set to the fixed constant for the process of the algorithm. However, it is easy to make the algorithm trap into the local solution and lead to convergence premature of the DE algorithm. Hence, in this section, to enable all solutions to get rid of the stagnation easily, an adaptive strategy of modifying the scale factor, crossover rate and the relative success number of the two new proposed parameters in a previous period is proposed. Overall, in this paper, three modifications are proposed in order to balance the exploration and exploitation of the DE algorithm.

3.1 Modified mutation strategy

Global exploration and local exploitation are two important aspects in developing an effective evolutionary algorithm. The first is to find every region of the search space and the second denotes the ability to converge to the optimal solution as fast as possible. The mutation strategy plays a vital role in the search capability and the convergence

rate. As mentioned above, there are many strategies proposed in DE (Yang and Deb 2009), each one has its own characterize. However, to our best knowledge, different optimization problems require different mutation strategies depending on the nature of the problems and available computation resources. DE/rand/2/bin is the mutation strategy that has been the most successful and widely used scheme in the literature. From this mutation strategy, it can be seen that five vectors are selected at random from the current population. Hence, it can conclude that the basic DE/rand/2/bin mutation strategy can maintain the population diversity and global search ability. However, it suffers for weak local exploitation ability, and its convergence rate is still lower when the optimal solution is reached. In Zhang and Sanderson (2009), this paper indicates that DE/current-to pbest/1/bin can maintain the diversity of the population and the convergence rate. In this strategy, an optional external archive is used and it can utilize historical data to provide information of the progress direction. The archive provides the information about the progress direction and it can enhance the diversity of the population. In this paper, because we don't use the external archive, the random vector will be added to the DE/current-to pbest/1/bin. The DE/rand-to pbest/1/bin can be described as follows:
"DE/rand-to-best/1/bin"

$$V_{i,G} = X_{r_1,G} + F \cdot (X^p_{best,G} - X_{r_1,G}) + F \cdot (X_{r_2,G} - X_{r_3,G}) \tag{5}$$

where $X^p_{best,G}$ is randomly chosen as one of the top 100p% in the current population, and $p \in [0, 1]$. This process explores the region according to the $X^p_{best,G}$. Moreover, it can maintain the diversity of the population and enhance the convergence rate of the algorithm at the same time. From the new mutation strategy, we can find that this new strategy has two benefits. Firstly, the randomly vectors are instead of the current individual in the original mutation strategy in Zhang and Sanderson (2009). It can enhance the diversity of the population. Secondly, the *pbest* individual is selected as the new mutation process that exploits the feasible region according to $X^p_{best,G}$. Consequently, this new mutation strategy has the better local search ability and it can enhance the diversity of the population. In this paper, this new mutation strategy is combined into the original DE algorithm and it is integrated into the basic mutation strategy DE/rand/2/bin through some probability rules. The framework of this new mutation strategy can be described as follows:

$$\begin{aligned}
&If \text{ rand} \leq \omega \text{ then} \\
&\quad V_{i,G} = X_{r_1,G} + F \cdot (X^p_{best,G} - X_{r_1,G}) + F \cdot (X_{r_2,G} - X_{r_3,G}) \\
&else \\
&\quad V_{i,G} = X_{r_1,G} + rand \cdot (X_{r_2,G} - X_{r_3,G}) + F \cdot (X_{r_4,G} - X_{r_5,G}) \\
&end\ if
\end{aligned} \tag{6}$$

As can be seen in formula 6, *rand* returns a real number between 0 and 1 with uniform random probability. ω is the probability value which determines to select the mutation rule. It can observe that for each individual, only one of the two strategies can be used for the process of generating the trail vector. Therefore, the value of ω is

very important. In this paper, we consider three different schemes for the selection and adaptation of ω to determine the mutation rule. We can describe them in the following:

(1) Random probability rule: in this scheme, the value of the ω of each individual is made to vary as a uniformly distributed random number in (0, 1), i.e. $\omega \in [0, 1]$. Such a choice may enhance the diversity of the population. However, it will make the convergence speed slower.
(2) Linear increment rule: ω is linearly increased from 0 to 1:

$$\omega = \frac{\mathrm{G}}{\mathrm{Gmax}} \tag{7}$$

Where *rand* is a real number between 0 and 1 with a uniform random probability distribution and *G* is the current generation. *Gmax* is the maximum generation number. Based on the modified search strategy, it can find that one of the two strategies is used for producing the current individual relative to a uniformly distributed random value within the range (0, 1). For each individual, if the random real number is smaller than (G/Gmax), then the DE/rand-to pbest/1/bin is used. Otherwise, the DE/rand/2/bin is used. For these rules, the probability of selecting one of these two new search strategies is a function of the generation number. The value of (G/Gmax) increases zero to one in order to balance exploration and the exploitation. In fact, in the beginning, two different strategies are selected to produce the offspring. From the (G/Gmax), the probability of selecting the DE/rand/2/bin is greater than the probability of the DE/rand-to pbest/1/bin. This process causes exploration. Then, as the generation increases, the two new search methods may be used with the same probability. This process directs the search. In the last, two search strategies are still used. However, in the opposite, the probability of selecting the DE/rand-to-pbest/1/bin is greater than the probability of the DE/rand/2/bin. Finally, it enhances the exploitation. Hence, based on the linear decreasing probability rule and two new search methods, the algorithm can balance the exploitation and exploration in the search space.

(3) Exponential increment rule: the value of ω increases from zero to one in an exponential fashion inspired by the paper (Das et al. 2009):

$$\omega = exp\left(\frac{G}{G_{\max}} \cdot \ln(2)\right) - 1 \tag{8}$$

Compared with the linear increment rule, this scheme increases the number of running the DE/rand/2/bin and reduces the number of running the DE/rand-to pbest/1/bin. In other words, this scheme is slower in the exploitation and faster in the exploration. The method of selecting the mutation rule is the same as the linear increment rule.

3.2 Randomized scale factor

The scale factor is a very important control parameter that controls the amplification of the differential variation between two individuals. In the original DE algorithm, the value of F is kept as a constant value. Small values of F lead to the premature

convergence, and high values of F slow down the search. To our knowledge, there is no optimal value of the F that can solve all complex benchmark problems. However, from the formula 6, we can find that if the value of F is always the same value, and the diversity of the population is lost as all the individuals are calculated by the same scale factor. Hence, the scale factor must be a positive value for the search process. In order to solve this drawback of keeping the F constant, we generate the value of F as two Gaussian distributions. Gaussian distribution is used because it gets most of the numbers within unity due to its short tail property. Moreover, the random numbers generated are not bound within any limit, this is because larger values of scale parameter 'F' will assist the solution space to easily escape from large plateaus or suboptimal peaks, thereby minimizing the chances to trap to local optima. It can be described as follows:

$$\begin{aligned} &If \ \text{rand}() \leq \text{rand}() \ \text{then} \\ &\quad F = |\text{Gaussian}(0.3,0.3)| \\ &else \\ &\quad F = |\text{Gaussian}(0.7,0.3)| \\ &end\ if \end{aligned} \tag{9}$$

where $F = |\text{Gaussian}(0.3,0.3)|$ is a random number with the range (0.6, 1.2) and $F = |\text{Gaussian}(0.7,0.3)|$ is a random number with the range (0.2, 1.6). This range ensures both the exploitation tendency and exploration ability.

3.3 Self adaptive crossover rate

The crossover operation aims to construct an offspring by mixing components of the current element and of that generated by mutation operator. The crossover rate reflects the probability with which the trail individual inherits the actual individual's genes. Small values of CR in exploratory move parallel to a small number of axes of the search space, while large values of CR move at angles to the search spaces' axes. Additionally, at the early stage of the search, the diversity of the population is large and the variance of the population is large. Hence, the CR needs to a small value in order to avoid exceeding the level of diversity and this can reduce the convergence rate. Then, after some generations, the population will become similar. In this stage, in order to advance the diversity and increase the convergence speed, the value of CR must be a large value. The paper (Montgomery and Chen 2010) also suggests that many existing adaptive DE techniques will be likely unable to find and exploit both low and high values of CR will avoid the value which adversely affect algorithm performances. Moreover, for different problems, they need different value of parameters; some problems need to enhance the diversity and the convergence speed. In other aspects, some problems need to smaller value to avoid the exceeding level of diversity that may result in premature convergence and slow the convergence rate. Therefore, based on the above analysis and discussions, and in order to balance the exploration and exploitation of the algorithm, in this paper, we propose a statistical learning strategy which selects the evolution method for each individual according to the relative success ratio of selecting one of two new parameters in the previous periods. Two new parameters

settings can be described as follows:

$$(1)\, CR_1 \in [0.05, 0.15];\ (2)\, CR_2 \in [0.85, 1] \tag{10}$$

Based on two new parameter settings, the first one can increase the possibility stagnation and slow down the search process. The second one can advance diversity and increase the convergence speed. In the beginning of the algorithm, we generate the value of parameter CR using the first one for each individual. Then we design a new label l to store the success ratio of two new parameters. In each generation, we generate the CR according to the two new parameters. If the CR is generated by using the first one, we will set the label $l_i^2 = 1$, otherwise if the CR is generated by using the second one, we will set the label $l_i^2 = 2$. Then after generating the offspring individuals, we calculate the individual and compare it with the previous individuals. If the CR is generated by using the first one with the better solution, we will set the label $l_i^1 = 1$, otherwise, we will set the label $l_i^1 = 2$. And then, we calculate the number of the success ratio. According to the success ratio, we can generate the new CR value. The method can be described as follows:

Algorithm 2. Parameter settings

Begin

for i=1 to *NP* $CR(i) = CR_1$; $l_i^1 = 0$; $l_i^2 = 0$; end

while G < Gmax then

for i=1 to *NP*

if G ==1

if rand<0.5

if (rand<=0.05) $CR(i) = CR_1$; l_i^2 =1; end

else

if (rand<=0.05) $CR(i) = CR_2$; l_i^2 =2; end

end

end

if G >1 if sucess1>sucess2 $p \in [0.7,1]$; else $p \in [0.1,0.4]$; end if; end

if rand<p

if (rand<=0.5) $CR(i) = CR_1$;end

else

if (rand<=0.5) $CR(i) = CR_2$;end

end

end

success1=0;success2=0;

for i=1 to *NP*

if l_i^2 ==1 if l_i^1 ==1 success1=success1+1; end; end

if l_i^2 ==2 if l_i^1 ==1 success2=success2+1; end; end

end

End

From the procedure, we can find that the parameter success1 and success2 are important for the algorithm. The sucess1 denotes the success number of using the first parameter setting in a learning period. The success2 denotes the success number of using the second parameter setting in a learning period. For the evolutionary process, if the individual using the first parameter setting is superior to the previous generation, the l_i^1 will be set one, otherwise it will be set zero. In the final of every generation, the summation of label l_i^1 for different parameter settings will be calculated. Then, we compare success1 and success2. If the sucess1 is larger than the success2, it denotes the first parameter setting can perform better than second parameter setting. We will add the probability of selecting the first parameter setting. Otherwise, it represents that the second parameter setting is very efficient. We also reduce the probability of selecting the first parameter setting, and then add the probability of selecting the second parameter setting.

3.4 Boundary constraints

The modified differential evolution algorithm assumes that the whole population should be in an isolated and finite space. During the searching process, if there are some individuals that may move out of bounds of the space, the original algorithm will stop them on the boundary. In other words, the individual will be assigned a boundary value. The disadvantage is that if there are too many individuals on the boundary, and especially when there exists some local minima on the boundary, the algorithm will lose its population diversity to some extent. In order to tackle this problem, we proposed the following repair rule:

$$x_i = \begin{cases} 2^* x_{\min} - x_i & \text{if} \quad x_i < x_{\min} \\ 2^* x_{\max} - x_i & \text{if} \quad x_i > x_{\max} \\ x_i & \text{otherwise} \end{cases} \tag{11}$$

where $x_{\max}$ and $x_{\min}$ are the upper bound and lower bound of each dimension for every individual, respectively.

3.5 Modified differential evolution with self adaptive parameter setting

In this section, we will propose a modified differential evolution with self adaptive parameter settings. Firstly, we use a new mutation rule based on the DE/rand/2/bin

and DE/rand-to-pbest/1/bin to generate the trail vector. The new mutation rule is used through the probability model. The proposed mutation rule maintains the diversity of the population and enhances the diversity of the population. Secondly, both operations diversify the population and improve the convergence speed, and the parameter adaptation automatically updates the control parameter according to the success ration of the two parameter settings in the learning period. In addition, the MDE has a very simple structure and thus it is easy to implement and not enhance any complexity. This method can overcome the lack of the exploitation of the original DE algorithm. The Algorithm 3 can be described as follows:

Algorithm 3. Algorithm description of MDE algorithm

Input: NP is the population size. G is the number of the iteration

Output: The best solutions.

1: **begin**

2: Set the generation counter G=0; and randomly initialize a population of *NP* individuals X_i . Initialize the parameter *F*, *CR*.

3: Evaluate the fitness for each individual in *P*.

4: **while** stopping criteria is not satisfied **do**

Calculate the value of parameter setting ⇐Algorithm 2

5: **for** i= 1 to *NP* **do**

6: select randomly $a \neq b \neq c \neq d \neq i$

7: **for** j=1 to *D* **do**

8: $j_{rand} = \lfloor rand(0,1) * D \rfloor$

9: **If** rand (0, 1) ≤CR or j== j_{rand} **then**

10: new proposed mutation rule ⇐Formulate (6)

11: /* the mutation strategies*/

12: **Else**

13: $u_{i,j} = x_{i,j}$

14: **end if**

15: **end for**

16: **end for**

17: **for** *i*=1 to *NP* **do**

18: Evaluate the offspring u_i

19: **If** u_i is better than X_i **then**

20: $X_i = u_i$

21: l_i^1=1; ⇐set the success number

Else

22 l_i^1=0; ⇐set the failure number

23: **end if**

24: **end for**

25: Calculate the success ratio

26: Memorize the best solution achieved so far

27: **end while**

28: **end**

4 Performance evaluation

To evaluate the performance of our algorithm, we apply it to 16 standards benchmark functions (Li et al. 2013; Mallipeddi et al. 2010). These functions have been widely used in the literature. The first four functions are unimodal functions. The following 12 functions are multimodal test functions. Among them, for the function f03, the generalized Rosenbrock's function is a multimodal function when D>3. For these functions, the number of the local minima increases exotically with the problem dimension. So far, these problems have been widely used as benchmarks for researches with different methods by many researchers. Definitions of the benchmark problems are described as follows:

(1) Shifted sphere function $f_1(x) = \sum_{i=1}^{D} z_i^2, z = \mathrm{X} - \mathrm{O}, o = [o_1, o_2, \ldots, o_D]$: the shift global optimum.

(2) Shifted Schwefel's problem 1.2 $f_2(x) = \sum_{i=1}^{D} \left(\sum_{j=1}^{i} z_i\right)^2, z = \mathrm{X} - \mathrm{O}, \quad o = [o_1, o_2, \ldots, o_D]$: the shift global optimum.

(3) Rosebrock's function $f_3(x) = \sum_{i=1}^{D-1} [100(x_{i+1} - x_i^2)^2 + (x_i - 1)^2]$,

(4) Shifted Schwefel's problem 1.2 with noise in fitness $f_4(x) = \sum_{i=1}^{D} \left(\sum_{j=1}^{i} z_i\right)^2 (1 + 0.4\,|N(0,1)|), z = \mathrm{X} - \mathrm{O}, \quad o = [o_1, o_2, \ldots, o_D]$: the shift global optimum.

(5) Shifted Ackley's function $f_5(x) = -20\exp(-0.2\sqrt{\frac{1}{D}\sum_{i=1}^{D} z_i^2}) - \exp(\frac{1}{D}\sum_{i=1}^{D} \cos 2\pi z_i), z = \mathrm{X} - \mathrm{O}, \quad o = [o_1, o_2, \ldots, o_D]$: the shift global optimum. +20+e

(6) Shifted rotated Ackley's function $f_5(x) = -20\exp(-0.2\sqrt{\frac{1}{D}\sum_{i=1}^{D} z_i^2}) - \exp(\frac{1}{D}\sum_{i=1}^{D} \cos 2\pi z_i), z = M(\mathrm{X}-\mathrm{O}), \quad cond(M) = 1, o = [o_1, o_2, \ldots, o_D]$: the shift global optimum. +20+e

(7) Shifted Griewank's function $f_7(x) = \frac{1}{400}\sum_{i=1}^{D} z_i^2 - \prod_{i=1}^{D} \cos(\frac{z_i}{\sqrt{i}}) + 1, z = \mathrm{X} - \mathrm{O}, \quad o = [o_1, o_2, \ldots, o_D]$: the shift global optimum.

(8) Shifted rotated Griewank's function $f_8(x) = \frac{1}{400}\sum_{i=1}^{D} z_i^2 - \prod_{i=1}^{D} \cos(\frac{z_i}{\sqrt{i}}) + 1, z = M(\mathrm{X} - \mathrm{O}), cond(M) = 3, \quad o = [o_1, o_2, \ldots, o_D]$: the shift global optimum.

(9) Shifted Rastrigin's function $f_9(x) = \sum_{i=1}^{D} [z_i^2 - 10\cos(2\pi z_i) + 10], z = \mathrm{X} - \mathrm{O}, o = [o_1, o_2, \ldots, o_D]$: the shift global optimum.

(10) Shifted rotated Rastrigin's function $f_{10}(x) = \sum_{i=1}^{D} [z_i^2 - 10\cos(2\pi z_i) + 10], z = M(\mathrm{X}-\mathrm{O}), cond(M) = 2, o = [o_1, o_2, \ldots, o_D]$: the shift global optimum.

(11) Shifted rotated Rastrigin's function

$$f_{10}(x) = \sum_{i=1}^{D} [z_i^2 - 10\cos(2\pi z_i) + 10],$$
$$y_i = \begin{cases} z_i & |z_i| < 1/2 \\ round(z_i)/2 & |z_i| < 1/2 \end{cases} \text{ for } i = 1, 2, \ldots, D$$

$z = \mathrm{X} - \mathrm{O}$, $o = [o_1, o_2, \ldots, o_D]$: the shift global optimum.

(12) Schwefel's function

$$f_{12}(x) = 418.9828 \times D - \sum_{i=1}^{D} x_i \sin(|x_i|^{1/2}).$$

(13) Penalty 1's function

$$f_{13}(x) = \frac{\pi}{D}\{10\sin^2(\pi y_i) + \sum_{i=1}^{D-1} (y_i - 1)^2[1 + 10\sin^2(\pi y_i + 1)] + (yD - 1)^2 + \sum_{i=1}^{D} u(x_i, 10, 100, 4)\}$$
$$y_i = 1 + \frac{x_i + 1}{4} \quad u(x_i, a, k, m) = \begin{cases} k(x_i - a)^m & x_i > a \\ 0 & -a < x_i < a \\ k(-x_i - a)^m & x_i < -a \end{cases}$$

(14) Penalty 2's function

$$f_{14}(x) = 0.1\{10\sin^2(\pi y_i) + \sum_{i=1}^{D-1} (y_i - 1)^2[1 + 10\sin^2(\pi y_i + 1)] + (yD - 1)^2\} + \sum_{i=1}^{D} u(x_i, 10, 100, 4)$$

(15) Composition function 1 (CF1) The function $f_{15}(x)$ (CF1) is composed by using 10 sphere functions; the global optimum is easy to find the global solution.

(16) Composition function 6 (CF6)

The function $f_{16}(x)$(CF6) is composed by using 10 different benchmark functions, i.e. 2 rotated Rastrigin's functions, 2 rotated Weierstress functions, 2 rotated Griewank's functions, 2 rotated Ackley's functions and 2 rotated Sphere functions.

Based on these functions, the shift and rotated functions make the optimum is very difficult to be found. Where $\vec{o}$ denotes the position of the shifted optima. M is a rotation matrix and $cond(M)$ is the condition number of the matrix. The global optimum, search ranges and initialization ranges of the test functions are presented in Table 1.

Table 1 Global optimum, search ranges and initialization ranges of the test functions

Test function	Dimenison	Global optimum x*	Range	Initialize range
f01	10 and 30	o	$(-100, 100)^D$	$(-100, 100)^D$
f02		o	$(-100, 100)^D$	$(-100, 100)^D$
f03		(1,1,…,1)	$(-100, 100)^D$	$(-100, 100)^D$
f04		o	$(-100, 100)^D$	$(-100, 100)^D$
f05		o	$(-32, 32)^D$	$(-32, 32)^D$
f06		o	$(-32, 32)^D$	$(-32, 32)^D$
f07		o	R	$(0, 600)^D$
f08		o	R	$(0, 600)^D$
f09		o	$(-5, 5)^D$	$(-5, 5)^D$
f10		o	$(-5, 5)^D$	$(-5, 5)^D$
f11		(420.96,…, 420.96)	$(-500, 500)^D$	$(-500, 500)^D$
f12		(420.96,…, 420.96)	$(-500, 500)^D$	$(-500, 500)^D$
f13		o	$(-50, 50)^D$	$(-50, 50)^D$
f14		o	$(-50, 50)^D$	$(-50, 50)^D$
f15		o	$(-5, 5)^D$	$(-5, 5)^D$
f16		o	$(-5, 5)^D$	$(-5, 5)^D$

4.1 Experimental setup

In order to evaluate the effectiveness and efficiency of MDE, we have chosen a suitable set of value and don't have made any effort in finding the best parameter settings because of this algorithm is a self adaptive method. In this experiment, we set the number of individuals in all algorithms to be 50. The algorithm is coded in MATLAB 7.9, and experiments are made on a Pentium 3.2 GHz Processor with 4.0 GB of memory. The above benchmark functions f1 to f16 are tested in 10-dimension and 30-dimension. The maximum number of function evaluations is set to 100,000 for 10D problems and 300,000 for 30D problems. For all test functions, the algorithms carry out 50 independent runs. The real numbers +, ≈, - denote that the modified differential evolution algorithm is superior to, equal to or inferior to other algorithms.

4.2 Effect of different ω value

In this section, we compared with three different probability rules including the random probability rule, linear increment rule and exponential increment rule. The experiment results are listed in Table 2. From Table 2, it is interesting to see that there is always one or more version of MDE that outperforms the standard DE with the DE/rand/1/bin scheme. This reflects the effectiveness of the incorporation of the hybrid mutation rule in DE. From the results, we also can find that the time-varying ω is better than

Table 2 Comparison between different ω on 30D problems

F	Standard DE	MDE with random probability rule	MDE with linear increment rule	MDE with Exponential increment rule
	Mean ± SD	Mean ± SD	Mean ± SD	Mean ± SD
$f01$	2.60e–028 ± 1.52e–028	**0 ± 0**	**0 ± 0**	**0 ± 0**
$f02$	6.59e–013 ± 8.40e–013	**2.84e–026 ± 2.84e–026**	4.17e–022 ± 6.52e–022	9.42e–020 ± 2.28e–019
$f03$	29.5149 ± 42.2659	0.2657 ± 1.0293	1.98e–005 ± 7.66e–005	**8.71e–008 ± 1.67e–007**
$f04$	**1.04e–004 ± 3.01e–004**	5.21e–004 ± 9.96e–004	2.27e–03 ± 4.40e–03	0.0054 ± 0.0110
$f05$	7.11e–015 ± 0	**3.55e–015 ± 0**	**3.55e–015 ± 0**	**3.55e–015 ± 0**
$f06$	**3.55e–015 ± 0**	**3.55e–015 ± 0**	**3.55e–015 ± 0**	**3.55e–015 ± 0**
$f07$	8.21e–004 ± 0.0032	**0 ± 0**	**0 ± 0**	**0 ± 0**
$f08$	0.0034 ± 0.0062	0.0015 ± 0.0031	**0 ± 0**	**0 ± 0**
$f09$	17.1133 ± 4.1401	0.0663 ± 0.2568	**0 ± 0**	**0 ± 0**
$f10$	98.2747 ± 72.3087	45.5690 ± 8.8609	**40.45 ± 10.79**	46.9619 ± 14.0214
$f11$	18.7315 ± 2.6894	**0 ± 0**	**0 ± 0**	**0 ± 0**
$f12$	272.4089 ± 199.9484	**0 ± 0**	**0 ± 0**	**0 ± 0**
$f13$	1.57e–032 ± 2.83e–048	**1.57e–032 ± 2.83e–048**	**1.57e–032 ± 0**	**1.57e–032 ± 2.83e–048**
$f14$	1.35e–032 ± 2.83e–048	**1.35e–032 ± 2.83e–048**	**1.350e–032 ± 0**	**1.35e–032 ± 2.83e–048**
$f15$	5.47e–031 ± 7.50e–031	**0 ± 0**	**0 ± 0**	**0 ± 0**
$f16$	15.7213 ± 35.9622	**3.7006 ± 1.0469**	3.978 ± 0.932	3.7334 ± 1.2484

Better results are highlighted in bold

the random probability rule. It notes that the random probability rule is also better than the original algorithm. Compared with two increment rules, the results show that the linear increment rule is better than the exponential increment rule. Therefore, in this paper, we use the linear increment rule as the final model to form a judicious tradeoff between the exploration and exploitation which is the key for the success of the MDE.

4.3 Comparison of the original DE algorithm with different parameters

In this section, we compare our method with six different DE/rand/1/bin including DE/rand/1/bin with (F = 0.8, CR = 0.9), DE/rand/1/bin with (F = 0.9, CR = 0.1), DE/rand/1/bin with (F = 0.9, CR = 0.9), DE/rand/1/bin with (F = 0.5, CR = 0.3), DE/rand/1/bin with (F = 0.5, CR = 0.9), and DE/rand/1/bin with (F = 0.7, CR = 0.3). The results (mean, standard deviation values) of this experiment are presented in Tables 3 and 4 for 10D and 30D, respectively. The "−" denotes our algorithm is better than other algorithms. And the "+" and "≈" denote our algorithm is inferior to, equal

Table 3 Comparison between MDE and various classical DE methods on 10D problems

F	DE/rand/1/bin $F = 0.8, CR = 0.9$ Mean ± SD	DE/rand/1/bin $F = 0.9, CR = 0.1$ Mean ± SD	DE/rand/1/bin $F = 0.9, CR = 0.9$ Mean ± SD	DE/rand/1/bin $F = 0.5, CR = 0.3$ Mean ± SD	DE/rand/1/bin $F = 0.5, CR = 0.9$ Mean ± SD	DE/rand/1/bin $F = 0.7, CR = 0.3$ Mean ± SD	This work Mean ± SD
$f01$	3.45e–023 ± 3.76e–023—	0 ± 0 ≈	8.57e–013 ± 8.54e–013—	0 ± 0 ≈	0 ± 0 ≈	0 ± 0 ≈	0 ± 0
$f02$	4.57e–012 ± 4.94e–012—	0.7673 ± 0.3235—	2.58e–005 ± 1.48e–005—	7.00e–009 ± 9.98e–009—	1.01e–029 ± 3.91e–029—	8.63e–005 ± 5.57e–005—	0 ± 0
$f03$	2.55e–009 ± 3.02e–009—	1.7311 ± 1.4310—	50.3274 ± 165.4850—	1.6618 ± 1.9698—	2.5387 ± 1.4883—	1.1234 ± 1.2597—	0 ± 0
$f04$	2.97e–010 ± 4.44e–010—	13.3875 ± 5.3033—	4.98e–004 ± 3.35e–004—	1.25e–006 ± 9.96e–007—	1.68e–029 ± 4.54e–029—	0.0045 ± 0.0026—	1.51e–029 ± 3.24e–029
$f05$	3.29e–012 ± 2.21e–012—	0 ± 0 ≈	7.8052 ± 9.5527—	0 ± 0 ≈	2.37e–016 ± 9.17e–016—	0 ± 0 ≈	0 ± 0
$f06$	7.54e–012 ± 4.11e–012—	1.81e–004 ± 6.87e–004—	8.91e–007 ± 4.62e–007—	3.55e–015 ± 0—	3.32e–015 ± 9.17e–016—	3.32e–015 ± 9.17e–016—	2.37e–015 ± 1.73e–015
$f07$	0.2755 ± 0.1699—	0 ± 0 ≈	0.2446 ± 0.1725—	0 ± 0 ≈	0.0190 ± 0.0154—	4.93e–004 ± 0.0019—	0 ± 0
$f08$	0.3406 ± 0.1525—	0.1231 ± 0.0436—	0.3121 ± 0.2361—	0.1603 ± 0.0377—	0.0299 ± 0.0270+	0.2211 ± 0.0528—	0.03988 ± 0.02896
$f09$	8.2057 ± 7.4506—	0 ± 0 ≈	10.3972 ± 5.9058—	0 ± 0 ≈	0.6633 ± 0.8952—	0 ± 0 ≈	0 ± 0
$f10$	23.8553 ± 10.9704—	14.3834 ± 3.4985—	16.8279 ± 11.5128—	17.7759 ± 3.0316—	10.0204 ± 6.4865—	18.8827 ± 3.0676—	6.3014 ± 3.7543
$f11$	7.3295 ± 3.4151—	0 ± 0 ≈	7.0614 ± 2.6958—	0 ± 0 ≈	3.1846 ± 3.2602—	0 ± 0 ≈	0 ± 0
$f12$	0 ± 0 ≈	0 ± 0 ≈	262.0187 ± 314.8240—	0 ± 0 ≈	0 ± 0 ≈	0 ± 0 ≈	0 ± 0
$f13$	9.43e–023 ± 1.30e–022—	4.71e–032 ± 1.13e–047≈	7.04e–013 ± 6.87e–013—	4.71e–032 ± 1.13e–047≈	4.71e–032 ± 1.13e–047≈	4.71e–032 ± 1.13e–047≈	4.71e–032 ± 0
$f14$	4.00e–023 ± 3.57e–023—	1.35e–032 ± 2.83e–048≈	8.21e–013 ± 1.09e–012—	1.35e–032 ± 2.83e–048≈	1.35e–032 ± 2.83e–048≈	1.35e–032 ± 2.83e–048≈	1.35e–032 ± 0
$f15$	6.08e–025 ± 6.01e–025—	0.0237 ± 0.0376—	2.19e–012 ± 3.48e–012—	3.6182 ± 9.8109—	0 ± 0 ≈	6.05e–016 ± 1.54e–015—	0 ± 0
$f16$	0.0856 ± 0.3314+	5.8893 ± 1.7025—	0.1018 ± 0.3943+	1.8167 ± 1.5593—	13.33 ± 35.18—	3.2044 ± 1.9639—	1.4334 ± 0.3422
—	14	8	15	8	10	9	
+	1	0	1	0	1	0	
≈	1	8	0	8	5	7	

Table 4 Comparison between MDE and various classical DE methods on 30D problems

F	DE/rand/1/bin $F = 0.8, CR = 0.9$ Mean ± SD	DE/rand/1/bin $F = 0.9, CR = 0.1$ Mean ± SD	DE/rand/1/bin $F = 0.9, CR = 0.9$ Mean ± SD	DE/rand/1/bin $F = 0.5, CR = 0.3$ Mean ± SD	DE/rand/1/bin $F = 0.5, CR = 0.9$ Mean ± SD	DE/rand/1/bin $F = 0.7, CR = 0.3$ Mean ± SD	This work Mean ± SD
$f01$	5.97e–008 ± 3.79e–008—	0 ± 0 ≈	0.0340 ± 0.0245—	0 ± 0 ≈	2.60e–028 ± 1.52e–028—	0 ± 0 ≈	0 ± 0
$f02$	20.3211 ± 12.4195—	3.25e+003 ± 482.9848—	740.8497 ± 385.4690—	1.27e+003 ± 338.2845—	6.59e–013 ± 8.40e–013—	5.07e+003 ± 739.3445—	4.17e–022 ± 6.52e–022
$f03$	28.1397 ± 21.0561—	29.2710 ± 6.2039—	381.2765 ± 555.0719—	25.4648 ± 14.1615—	29.5149 ± 42.2659—	22.7173 ± 2.1067—	1.98e–005 ± 7.66e–005
$f04$	237.8955 ± 107.4926—	1.17e+004 ± 2.91e+003—	5.58e+003 ± 2.88e+003—	3.82e+003 ± 1.03e+003—	1.04e–004 ± 3.01e–004+	8.85e+003 ± 1.65e+003—	2.27e–03 ± 4.40e–03
$f05$	8.37e–005 ± 3.48e–005—	7.11e–015 ± 0—	18.9269 ± 5.0686—	4.26e–015 ± 1.47e–015—	7.11e–015 ± 0—	6.87e–015 ± 9.17e–016—	3.55e–015 ± 0
$f06$	1.53e–004 ± 7.57e–005—	0.1584 ± 0.1111—	0.0700 ± 0.0392—	3.55e–015±0 ≈	3.55e–015±0 ≈	6.16e–015 ± 1.63e–015—	3.55e–015 ± 0
$f07$	4.95e–004 ± 0.0019—	0 ± 0 ≈	0.0888 ± 0.1018—	0 ± 0 ≈	8.21e–004 ± 0.0032—	0 ± 0 ≈	0 ± 0
$f08$	0.0020 ± 0.0021—	0.0825 ± 0.0282—	0.7164 ± 0.2255—	5.52e–007 ± 1.98e–006—	0.0034 ± 0.0062—	0.0569 ± 0.0508—	0 ± 0
$f09$	34.6460 ± 23.2772—	0 ± 0 ≈	59.0989 ± 22.1190—	25.2083 ± 6.3012—	17.1133 ± 4.1401—	49.4920 ± 5.2209—	0 ± 0
$f10$	211.9034 ± 21.8161—	154.1508 ± 10.6286—	220.2144 ± 20.0484—	182.7177 ± 6.6364—	98.2747 ± 72.3087—	197.2288 ± 8.7948—	40.45 ± 10.79
$f11$	38.8497 ± 7.6680—	0 ± 0 ≈	52.5146 ± 20.4332—	27.8388 ± 2.6704—	18.7315 ± 2.6894—	33.6042 ± 1.9870—	0 ± 0
$f12$	159.1146 ± 138.7869—	0 ± 0 ≈	1.23e+003 ± 825.317—8	0 ± 0 ≈	272.4089 ± 199.9484—	0 ± 0 ≈	0 ± 0
$f13$	2.48e–008 ± 2.89e–008—	1.57e–032 ± 2.83e–048≈	0.0027 ± 0.0037—	1.57e–032 ± 2.83e–048≈	1.57e–032 ± 2.83e–048≈	1.57e–032 ± 2.83e–048≈	1.57e–032 ± 0
$f14$	1.01e–007 ± 1.09e–007—	1.35e–032 ± 2.83e–048≈	0.0130 ± 0.0143—	1.35e–032 ± 2.83e–048≈	1.35e–032 ± 2.83e–048≈	1.35e–032 ± 2.83e–048≈	1.350e–032 ± 0
$f15$	5.17e–009 ± 3.71e–009—	0.0316 ± 0.1182—	0.0026 ± 0.0023—	4.19e–026 ± 1.62e–025—	5.47e–031 ± 7.50e–031—	3.01e–021 ± 1.17e–020—	0 ± 0
$f16$	8.4209 ± 8.4064—	12.9927 ± 2.7601—	106.7107 ± 325.5164—	11.9400 ± 1.5035—	15.7213 ± 35.9622—	15.2235 ± 2.2931—	3.978 ± 0.932
—	16	9	16	10	12	11	
+	0	0	0	0	1	0	
≈	0	7	0	6	3	5	

to other algorithms. This comparison is between these DE algorithms with different parameter settings with our algorithm MDE. In Tables 3 and 4, we can find that no single parameter setting is adopted for all problems. For 10D problem presented in Table 3, it can be observed that MDE is inferior to, equal to and superior to the differential evolution with different parameter settings in 11, 29, and 56 problems, respectively out of the total 96 cases. Therefore, the MDE is always either better or equal. Moreover, as can be seen in this table, we can find the DE/rand/1/bin with (F = 0.9, CR = 0.1) and DE/rand/1/bin with (F = 0.5, CR = 0.3) are better than other classical DE algorithms. For the 30D problems presented in Table 4, it can observed that MDE is inferior to, equal to and superior to the different methods in 1, 21, and 74 problems, respectively out of the total 96 cases. From this table, the results of MDE are statistically significantly better as compared to all other algorithms considered here. Obviously, it can be concluded that MDE is superior to its entire algorithm in all these 16 test functions in terms of average and standard deviation values. In this table, for the function f04, the DE/rand/1/bin with (F = 0.5, CR = 0.9) can give the best solution than the MDE algorithm. For the 10D and 30D problem, we can find that the performance of all these algorithms is very similar. The significant difference is that the performance of MDE is not affected in a worse way along with the growth of the search space dimensionality. Therefore, it can conclude that MDE is better than the DE algorithm with different parameter settings. This result indicates that MDE has the greater robustness than DE with different parameter settings on the majority of functions.

4.4 Comparison of MDE with state-of-art DE variants

In this section, we compare MDE with six other DE variants including ODE (Rahnamayan et al. 2008), OXDE (Wang et al. 2011b), SaDE (Qin et al. 2009), JADE (Zhang and Sanderson 2009), jDE (Brest et al. 2006), and CoDE (Wang et al. 2011a). The mean and standard deviation values are presented in Tables 5 and 6 for 10D and 30D problems, respectively. To compare the performance of different algorithms, the results are labeled by the "−", "+" and "≈". Out of the five algorithms used for comparison, the MDE is statistically better than other algorithms that use the label "−". From the results, it can be seen that MDE can perform better than other algorithms in most of the cases. For the 10D problems in Table 5, among these algorithms used for comparison, OXDE is weaker in performance. But, it can provide the best solution for the function f16 compared with other algorithms. For the unimodal function f01–f04, MDE can obtain the two better solutions than other algorithms except the f03. For f04, ODE and JADE can give the better solution than other algorithms. For multimodal functions f08–f16 with many local minima, the final results are more important because of this function can reflect the algorithm's ability to escape from the poor local optima and obtain the near-global optimum. The MDE provides better solutions for some functions except f06, f08, f10, and f16. For these four functions, the optimal solution can be obtained by the ODE, SaDE, JADE, jDE and CoDE. For f06, SaDE can give the better results. For f08 and f10, JADE gives the best solution. All in all, it can be observed that MDE is inferior to, equal to, superior to compared algorithm in 10, 41,

Table 5 Comparison between MDE and various state-of-the-art DE methods on 10D problems

	ODE	OXDE	SaDE	JADE	*j*DE	CoDE	This work
	Mean ± SD	Mean ± SD	Mean ± SD	Mean ± SD	Mean ± SD	Mean ± SD	Mean ± SD
$f01$	0 ± 0 ≈	4.53e–026 ± 4.13e–026—	0 ± 0 ≈	0 ± 0 ≈	0 ± 0 ≈	0 ± 0 ≈	0 ± 0
$f02$	1.68e–029 ± 4.54e–029—	2.44e–011 ± 2.50e–011—	1.51e–029 ± 4.01e–029—	0 ± 0 ≈	2.19e–029 ± 6.17e–029—	0 ± 0 ≈	0 ± 0
$f03$	4.7798 ± 1.5191—	9.16e–010 ± 1.05e–009—	1.1696 ± 1.0968—	1.5074 ± 7.3029—	0.2658 ± 1.0114—	0.1329 ± 0.7278—	0 ± 0
$f04$	0 ± 0+	6.63e–010 ± 5.96e–010—	1.72e–029 ± 5.63e–029—	0 ± 0+	2.83e–029 ± 5.07e–029—	5.65e–028 ± 7.05e–028—	1.51e–029 ± 3.24e–029
$f05$	0 ± 0 ≈	1.78e–013 ± 1.20e–013—	0 ± 0 ≈	0 ± 0 ≈	0 ± 0 ≈	0 ± 0 ≈	0 ± 0
$f06$	3.32e–015 ± 9.17e–016—	2.16e–013 ± 1.39e–013—	1.78e–015 ± 1.81e–015+	3.37e–015 ± 8.48e–016—	3.43e–015 ± 6.49e–016—	3.43e–015 ± 6.49e–016—	2.37e–015 ± 1.73e–015
$f07$	0.0422 ± 0.0418—	0.0899 ± 0.0545—	0 ± 0 ≈	1.72e–004 ± 0.0017—	0 ± 0 ≈	0 ± 0 ≈	0 ± 0
$f08$	0.1082 ± 0.1291—	0.1547 ± 0.1622—	0.0206 ± 0.02035—	0.00683 ± 0.00626+	0.0197 ± 0.0165+	0.0458 ± 0.0267—	0.03988 ± 0.02896
$f09$	2.1338 ± 2.0578—	0.0102 ± 0.0396—	0 ± 0 ≈	0 ± 0 ≈	0 ± 0 ≈	0 ± 0 ≈	0 ± 0
$f10$	12.2709 ± 7.7603—	12.3755 ± 10.6854—	6.4009 ± 2.5290—	4.3731 ± 1.4421+	7.0237 ± 2.2727—	7.7275 ± 2.5692—	6.3014 ± 3.7543
$f11$	2.1393 ± 1.4079—	2.8620 ± 1.3078—	0 ± 0 ≈	0 ± 0 ≈	0 ± 0 ≈	0 ± 0 ≈	0 ± 0
$f12$	0 ± 0 ≈	0 ± 0 ≈	0 ± 0 ≈	42.6378 ± 70.4622—	0 ± 0 ≈	0 ± 0 ≈	0 ± 0
$f13$	4.71e–032 ± 1.13e–047≈	4.15e–026 ± 5.30e–026—	4.71e–032 ± 2.23e–047≈	4.71e–032 ± 4.40e–047≈	4.71e–032 ± 2.23e–047≈	4.7116e–032 ± 2.23e–047≈	4.71e–032 ± 0
$f14$	1.35e–032 ± 2.83e–048≈	7.38e–026 ± 9.81e–026—	1.35e–032 ± 5.57e–048≈	1.35e–032 ± 2.48e–047≈	1.35e–032 ± 5.57e–048≈	1.35e–032 ± 5.57e–048≈	1.35e–032 ± 0
$f15$	6.6667 ± 25.8199—	6.6667 ± 25.8199—	0 ± 0 ≈	40 ± 69.92—	0 ± 0 ≈	0 ± 0 ≈	0 ± 0
$f16$	0.0794 ± 0.3076+	2.02e–017 ± 7.83e–017+	10.1574 ± 31.5713—	50.1191 ± 52.5802—	0.5207 ± 0.8691+	0.3351 ± 0.7493+	1.4334 ± 0.3422
—	9	14	6	6	5	5	
+	2	1	1	3	2	1	
≈	5	1	9	7	9	10	

Table 6 Comparison between MDE and various state-of-the-art DE methods on 30D problems

F	ODE Mean ± SD	OXDE Mean ± SD	SaDE Mean ± SD	JADE Mean ± SD	*j*DE Mean ± SD	CoDE Mean ± SD	This work Mean ± SD
*f*01	9.3879 ± 35.5812—	1.51e–025 ± 2.06e–025—	0 ± 0 ≈	0 ± 0 ≈	0 ± 0 ≈	0 ± 0 ≈	0 ± 0
*f*02	1.28e–008 ± 2.64e–008—	1.8740 ± 1.1593—	8.32e–006 ± 2.29e–005—	6.55e–028 ± 5.13e–028+	1.07e–010 ± 1.70e–010—	2.27e–015 ± 3.55e–015—	4.17e–022 ± 6.52e–022
*f*03	116.5952 ± 163.0865—	0.5398 ± 1.4095—	25.0378 ± 24.9568—	0.839 ± 3.793—	0.9873 ± 1.7467—	0.0797 ± 0.5638—	1.98e–005 ± 7.66e–005
*f*04	8.69e–007 ± 2.74e–006+	49.4513 ± 31.0137—	1.69e+002 ± 2.26e+002—	1.124 ± 7.953—	0.3654 ± 1.2438—	0.0139 ± 0.0588—	2.27e–03 ± 4.40e–03
*f*05	0.0823 ± 0.2001—	9.14e–014 ± 4.50e–014—	0.1316 ± 0.3432—	3.91e–015 ± 1.08e–015—	4.50e–015 ± 1.60e–015—	3.55e–015 ± 0≈	3.55e–015 ± 0
*f*06	0.1413 ± 0.5474—	1.08e–013 ± 4.11e–014—	0.3135 ± 0.5452—	3.80e–015 ± 1.17e–015—	4.14e–015 ± 1.35e–015—	3.55e–015 ± 0≈	3.55e–015 ± 0
*f*07	0.0119 ± 0.0194—	0 ± 0 ≈	0.0037 ± 0.0067—	1.48e–03 ± 3.89e–03—	0 ± 0 ≈	0 ± 0 ≈	0 ± 0
*f*08	0.7573 ± 1.8846—	0.0023 ± 0.0041—	0.01442 ± 0.01562—	7.19e–03 ± 7.75e–03—	0.0046 ± 0.0064—	0.0028 ± 0.0045—	0 ± 0
*f*09	21.0237 ± 5.3272—	9.2216 ± 2.6498—	0.0663 ± 0.2524—	0 ± 0 ≈	0 ± 0 ≈	0.0398 ± 0.1970—	0 ± 0
*f*10	57.7531 ± 20.9002—	162.4719 ± 40.3121—	50.8091 ± 15.0074—	29.14 ± 6.30+	39.44 ± 6.27+	39.16 ± 12.13+	40.45 ± 10.79
*f*11	32.7157 ± 10.1377—	22.4718 ± 2.9756—	0.400 ± 0.6747—	0 ± 0 ≈	0 ± 0 ≈	0.2000 ± 0.4518—	0 ± 0
*f*12	1.06e+003 ± 453.0494—	0 ± 0 ≈	0 ± 0 ≈	1.26e+002 ± 1.05e+002—	7.8959 ± 30.0488—	0 ± 0 ≈	0 ± 0
*f*13	7.41e–009 ± 2.85e–008—	1.36e–026 ± 2.40e–026—	1.571e–032 ± 5.57e–048≈	1.571e–032 ± 5.529e–048≈	0.0035 ± 0.0189—	0.0021 ± 0.0147—	1.571e–032 ± 0
*f*14	3.20e–005 ± 1.24e–004—	9.77e–026 ± 9.85e–026—	3.66e–004 ± 0.0020—	2.20e–004 ± 1.55e–03—	1.350e–032 ± 5.57e–048≈	1.35e–032 ± 1.11e–047≈	1.350e–032 ± 0
*f*15	7.4665 ± 25.7844—	1.44e–027 ± 1.35e–027—	1.05e–031 ± 2.35e–031—	40 ± 96.61—	10 ± 31.6228—	0 ± 0 ≈	0 ± 0
*f*16	7.3843 ± 3.8551—	3.0490 ± 0.9107+	3.3905 ± 1.5466+	30.792 ± 43.368—	1.8138 ± 0.7747+	3.8751 ± 1.3006+	3.978 ± 0.932
—	*15*	*13*	12	10	9	8	
+	1	1	1	2	2	1	
≈	0	2	3	4	5	7	

Table 7 Comparison between MDE with CLPSO, CMA-ES, and GL-25 algorithms on 10D problems

F	CLPSO	CMA-ES	GL-25	This work
	Mean ± SD	Mean ± SD	Mean ± SD	Mean ± SD
$f01$	0 ± 0 ≈	8.8984e–28 ± 3.1124e–28—	0 ± 0 ≈	0 ± 0
$f02$	0.089715 ± 0.067926—	5.7606e–27 ± 1.5281e–27—	9.9131e–27 ± 3.0613e–26—	0 ± 0
$f03$	0.70985 ± 0.60485—	0.39866 ± 1.2607—	2.8128 ± 0.75363—	0 ± 0
$f04$	3.0674 ± 2.2645—	23310 ± 25957—	8.4632e–08 ± 1.3773e–07—	1.51e–029 ± 3.24e–029
$f05$	5.3291e–15 ± 1.8724e–15—	17.847 ± 6.2714—	7.1054e–16 ± 2.2469e–15—	0 ± 0
$f06$	3.4493e–07 ± 5.7139e–07—	19.415 ± 0.29694—	3.5527e–15 ± 0≈	2.37e–015 ± 1.73e–015
$f07$	1.9938e–07 ± 2.5081e–07—	0.013542 ± 0.009995—	0.019196 ± 0.020952—	0 ± 0
$f08$	0.10353 ± 0.045341—	0.013302 ± 0.007802+	0.014768 ± 0.013268+	0.03988 ± 0.02896
$f09$	0 ± 0 ≈	78.103 ± 55.786—	4.6685 ± 2.0002—	0 ± 0
$f10$	9.8605 ± 1.9166—	48.055 ± 99.516—	17.747 ± 8.2943—	6.3014 ± 3.7543
$f11$	0 ± 0 ≈	142.3 ± 58.275—	6.276 ± 1.1247—	0 ± 0
$f12$	0 ± 0 ≈	1862.9 ± 449.79—	144.1 ± 154.61—	0 ± 0
$f13$	4.7116e–32 ± 1.5472e–46—	2.7752e–31 ± 7.6565e–32—	4.8665e–32 ± 4.8981e–33≈	4.71e–032 ± 0
$f14$	1.8675e–32 ± 9.8744e–33—	0.0021975 ± 0.0046327—	3.2493e–32 ± 6.0068e–32≈	1.35e–032 ± 0
$f15$	0.022459 ± 0.068249—	190 ± 218.33—	0 ± 0 ≈	0 ± 0
$f16$	5.8892 ± 3.9436—	210.51 ± 184.82—	100 ± 4.737e–15—	1.4334 ± 0.3422
—	12	15	11	
+	0	1	0	
≈	4	0	5	

and 45 cases, respectively out of the 96 cases. For the dimensional 30, the experimental results of 50 independent runs are summarized in Table 6. From the Table 6, we can find that MDE is able to find the optimal or near-optimal solution with small deviations for these 12 functions including f02, f04, f10 and f16. For the f02 and f10, JADE can give the best solutions. For f04, ODE can perform better than other algorithms. jDE can perform better than MDE in the f16 function. In general, the performance of MDE is highly competitive with other algorithms, especially for the high-dimensional problems.

4.5 Comparison of MDE with other state-of-art algorithm variants

In order to evaluate the effectiveness and efficiency of MDE, we compare its performance with CLPSO (Liang et al. 2006), CMA-ES (Hansen and Ostermeier 2001),

Table 8 Comparison between MDE with CLPSO, CMA-ES, and GL-25 algorithms on 30D problems

F	CLPSO	CMA-ES	GL-25	This work
	Mean ± SD	Mean ± SD	Mean ± SD	Mean ± SD
$f01$	**0 ± 0**≈	1.6722e–25 ± 3.5518e–26—	7.4614e–24 ± 2.3588e–23—	0 ± 0
$f02$	837.52 ± 172.78—	5.6794e–25 ± 1.1122e–25+	32.234 ± 29.284—	4.17e–022 ± 6.52e–022
$f03$	4.1396 ± 3.6828—	1.196 ± 1.9257—	20.683 ± 1.0665—	1.98e–005 ± 7.66e–005
$f04$	7831.9 ± 1242.6—	32523 ± 23762—	504.85 ± 266.3—	2.27e–03 ± 4.40e–03
$f05$	1.8829e–14 ± 3.3704e–15—	19.762 ± 0.084447—	2.3448e–14 ± 1.233e–14—	3.55e–015 ± 0
$f06$	0.0030707 ± 0.0036104—	19.488 ± 0.1449—	9.2939e–13 ± 2.6891e–12—	3.55e–015 ± 0
$f07$	2.2204e–17 ± 7.0217e–17—	2.2204e–17 ± 7.0217e–17—	5.218e–16 ± 5.3129e–16—	0 ± 0
$f08$	0.033509 ± 0.018325—	0.0027111 ± 0.0044163—	1.2812e–07 ± 2.5833e–07—	0 ± 0
$f09$	0 ± 0 ≈	495.48 ± 90.483—	26.119 ± 10.932—	0 ± 0
$f10$	100.54 ± 15.333—	42.783 ± 10.103—	126.91 ± 68.241—	40.45 ± 10.79
$f11$	0 ± 0 ≈	421.6 ± 105.18—	35.386 ± 10.769—	0 ± 0
$f12$	1.819e–13 ± 5.7521e–13—	5223.7 ± 857.16—	3585.3 ± 1224.8—	0 ± 0
$f13$	2.1565e–31 ± 2.363e–31—	2.8538e–29 ± 5.3004e–30—	2.771e–32 ± 1.5949e–32—	1.57e–032 ± 0
$f14$	1.6892e–30 ± 1.1956e–30—	0.0042998 ± 0.0074428—	5.3084e–31 ± 6.0552e–31—	1.350e–032 ± 0
$f15$	1.0428e–06 ± 2.9459e–06—	40 ± 69.921—	0 ± 0 ≈	0 ± 0
$f16$	7.6027 ± 1.4511—	65.973 ± 155.58—	3.1989 ± 0.68806+	3.978 ± 0.932
—	13	15	14	
+	0	1	1	
≈	3	0	1	

and GL-25 (Garcia-Martinez et al. 2008). Liang et al. proposed a new particle swarm optimization-CLPSO. A particle used the personal historical best information of all the particles to update its velocity. Hansen and Ostermeier proposed a very efficient and famous evolution strategy. Garcia-Martinez et al. proposed a hybrid real-coded genetic algorithm which combines the global and local search. Each method was run 50 times on each test function. Table 7 summarizes the experimental results for 10 dimensional. As can be seen in Table 7, MDE significantly outperforms CLPSO, CMA-ES, and GL-25. MDE performs better than CLPSO, CMA-ES, and GL-25 on 12, 15, and 11 out of 16 test function, respectively. CLPSO is superior to, equal to MDE on 5 test functions. CMA-ES performs superior to MDE on one test functions. GL-25 is superior to, equal to MDE on 4 test functions. For the 30D, the results are shown in Table 8 in terms of the mean, standard deviation of the solutions obtained in the 50 independent runs by each algorithm. From the Table 8, we can find that the MDE provides better solutions than other algorithms. From the results, it can observed that MDE is inferior

Table 9 Comparison between MDE and different EP on 10D problems

F	AFEP	MSAEP	ENAEP	This work
	Mean ± SD	Mean ± SD	Mean ± SD	Mean ± SD
$f01$	4.9835e–14 ± 4.6744e–14—	4.1988e–18 ± 3.2888e–18—	6.6745e–18 ± 6.4698e–18—	0 ± 0
$f02$	3.8504e–11 ± 3.1914e–11—	4.08e–14 ± 3.6915e–14—	5.1726e–14 ± 3.9589e–14—	0 ± 0
$f03$	19.318 ± 30.205—	11.189 ± 21.353—	2.7468 ± 0.94971—	0 ± 0
$f04$	5.6113e–11 ± 4.6843e–11—	4.8504e–13 ± 6.4947e–13—	6.3632e–13 ± 1.3852e–12—	1.51e–029 ± 3.24e–029
$f05$	9.8406e–08 ± 5.0821e–08—	20.015 ± 0.016352—	1.1098e–09 ± 2.1645e–10—	0 ± 0
$f06$	8.9152e–08 ± 5.3185e–08—	8.8765e–10 ± 4.8945e–10—	1.1168e–09 ± 3.8896e–10—	2.37e–015 ± 1.73e–015
$f07$	0.050678 ± 0.029513—	0.043309 ± 0.023988—	0.040379 ± 0.019135—	0 ± 0
$f08$	0.04775 ± 0.030614—	0.030037 ± 0.016838+	0.040835 ± 0.017835—	0.03988 ± 0.02896
$f09$	1.194 ± 0.62927—	986.22 ± 3.2797e–10—	2.7859 ± 1.2231—	0 ± 0
$f10$	12.24 ± 3.8678—	983.78 ± 1.195—	6.6805 ± 2.917—	6.3014 ± 3.7543
$f11$	0.1 ± 0.31623—	971 ± 0.003965—	1.4211e–15 ± 3.4322e–15—	0 ± 0
$f12$	142.13 ± 145.59—	274.47 ± 133.45—	248.72 ± 87.391—	0 ± 0
$f13$	7.1647e–16 ± 7.191e–16—	4.8126e–20 ± 6.086e–20—	6.2273e–20 ± 4.6085e–20—	4.71e–032 ± 0
$f14$	3.6349e–15 ± 4.7699e–15—	1.6907e–19 ± 1.5616e–19—	2.3595e–19 ± 1.9697e–19—	1.35e–032 ± 0
$f15$	1.8264e–15 ± 1.6628e–15—	20 ± 42.164—	10 ± 31.623—	0 ± 0
$f16$	0.67731 ± 0.71864+	2.7547e–19 ± 4.9295e–19+	0.60196 ± 0.78696+	1.4334 ± 0.3422
—	15	14	15	
+	1	2	1	
≈	0	0	0	

to, equal to, superior to compared algorithms in 1, 4, and 43 cases, respectively out of the 48 cases. The results demonstrate good performance of MDE in solving these problems.

4.6 Comparison of MDE, AFEP, MSAEP and ENAEP

In the experiment, we compare the performance of MDE and AFEP, MSAEP and ENAEP (Mallipeddi et al. 2010). AFEP is an adaptive evolutionary programming which presents empirical studies carried out to evaluate the performance of different constraint handling methods on constrained real-parameter optimization. MSAEP presents a mixed strategy evolutionary programming algorithm that employs the Gaussian, Cauchy, Lévy, and single-point mutation operators. The results of the

Table 10 Comparison between MDE and different EP on 30D problems

F	AFEP	MSAEP	ENAEP	This work
	Mean ± SD	Mean ± SD	Mean ± SD	Mean ± SD
$f01$	9.8721e–16 ± 6.5416e–16—	8.9186e–15 ± 2.6125e–14—	1.8348e–20 ± 9.5662e–21—	0 ± 0
$f02$	0.13609 ± 0.3052—	0.060647 ± 0.11101—	0.066752 ± 0.16543—	4.17e–022 ± 6.52e–022
$f03$	100.69 ± 137.63—	213.06 ± 297.38—	79.735 ± 106.63—	1.98e–005 ± 7.66e–005
$f04$	468.42 ± 627.1—	816.09 ± 381.36—	880.5 ± 757.85—	2.27e–03 ± 4.40e–03
$f05$	5.9476e–09 ± 2.6802e–09—	20.004 ± 0.0081259—	2.6294e–11 ± 1.1745e–11—	3.55e–015 ± 0
$f06$	8.5292e–09 ± 2.4153e–09—	7.7875e–07 ± 2.4531e–06—	4.5934e–11 ± 2.6546e–11—	3.55e–015 ± 0
$f07$	0.012074 ± 0.014616—	0.010837 ± 0.014735—	0.01545 ± 0.023332—	0 ± 0
$f08$	0.0063988 ± 0.0086851—	0.0034512 ± 0.0064906—	0.010097 ± 0.0072799—	0 ± 0
$f09$	5.5718 ± 2.0009—	2230.8 ± 1.08—	8.0592 ± 4.4853—	0 ± 0
$f10$	78.674 ± 62.834—	2212.7 ± 17.328—	45.745 ± 11.565—	40.45 ± 10.79
$f11$	1.3 ± 1.1595—	2197.5 ± 1.9579—	1.5 ± 1.4337—	0 ± 0
$f12$	823.97 ± 304.77—	1223.7 ± 235.96—	1282 ± 305.3—	0 ± 0
$f13$	1.0679e–18 ± 7.5803e–19—	1.3953e–21 ± 1.0908e–21—	2.1418e–22 ± 2.1796e–22—	1.57e–032 ± 0
$f14$	5.1166e–17 ± 6.0938e–17—	3.109e–21 ± 1.8625e–21—	0.0010987 ± 0.0034745—	1.350e–032 ± 0
$f15$	1.1568e–17 ± 8.8581e–18—	20 ± 44.721—	20 ± 42.164—	0 ± 0
$f16$	7.1059 ± 1.9406—	40 ± 89.443—	7.4433 ± 2.9433—	3.978 ± 0.932
—	16	16	16	
+	0	0	0	
≈	0	0	0	

MSAEP show that the mixed strategy performs equally well or better than the best of the four pure strategies, for all of the benchmark problems. ENAEP enables us to benefit from different mutation operators with different parameter values whenever they are effective during different stages of the search process. Experimental results show that the algorithm is very effective. The experimental results are given in Tables 9 and 10 for different algorithms at dimension 10D and 30D. For 10D, the performance of MDE is significantly superior to all evolutionary algorithms for all functions except f08 and f16 according to the experimental results is shown in Table 9. For f08 and f16, MSAEP slightly performs better than MDE. All in all, from the results, it can be observed that MDE is inferior to, equal to, superior to the compared algorithms on 4, 0, and 44 cases. For the 30D, as can be seen in Table 10, we can conclude that the performance of all other algorithms is very similar to that on the 10-deminsional benchmark functions. The significant difference is that the performance of MDE is not affected in a worse way with the increase of the dimensional. Hence, we can also

Table 11 Comparison between MDE and MDE with different version on 30D problem

F	Standard DE	MDE1	MDE2	MDE
	Mean ± SD	Mean ± SD	Mean ± SD	Mean ± SD
$f01$	2.60e–028 ± 1.52e–028—	0 ± 0 ≈	2.06e–021 ± 3.57e–021—	0 ± 0
$f02$	6.59e–013 ± 8.40e–013—	6.84e–012 ± 7.57e–012—	0.0463 ± 0.0793—	4.17e–022 ± 6.52e–022
$f03$	29.5149 ± 42.2659—	21.0188 ± 24.9765—	87.4198 ± 64.8129—	1.98e–005 ± 7.66e–005
$f04$	1.04e–004 ± 3.01e–004+	5.2688 ± 9.7792—	5.0942 ± 8.2905—	2.27e–03 ± 4.40e–03
$f05$	7.11e–015 ± 0—	3.55e–015 ± 0≈	2.79e–007 ± 4.81e–007—	3.55e–015 ± 0
$f06$	3.55e–015 ± 0≈	3.55e–015 ± 0≈	3.55e–015 ± 0≈	3.55e–015 ± 0
$f07$	8.21e–004 ± 0.0032—	0 ± 0 ≈	1.07e–015 ± 1.86e–015—	0 ± 0
$f08$	0.0034 ± 0.0062—	0.0046 ± 0.0078—	0.0090 ± 0.0086—	0 ± 0
$f09$	17.1133 ± 4.1401—	0 ± 0 ≈	12.9444 ± 6.9497—	0 ± 0
$f10$	98.2747 ± 72.3087—	54.4619 ± 14.8664—	48.4795 ± 18.1950—	40.45 ± 10.79
$f11$	18.7315 ± 2.6894—	0.0666 ± 0.2581—	23.4451 ± 1.5538—	0 ± 0
$f12$	272.4089 ± 199.9484—	0 ± 0 ≈	1.97e+002 ± 1.37e+002—	0 ± 0
$f13$	1.57e–032 ± 2.83e–048≈	1.57e–032 ± 2.83e–048≈	0.0607 ± 0.0590—	1.57e–032 ± 0
$f14$	1.35e–032 ± 2.83e–048≈	1.35e–032 ± 2.83e–048≈	0.0036 ± 0.00634—	1.350e–032 ± 0
$f15$	5.47e–031 ± 7.50e–031—	0 ± 0 ≈	1.80e–015 ± 2.55e–015	0 ± 0
$f16$	15.7213 ± 35.9622—	4.5797 ± 1.022—	3.5267 ± 0.8883	3.978 ± 0.932
—	12	7	15	
+	1	0	1	
≈	3	9	0	

conclude that the self adaptive method and the new search method can enhance the ability to accelerate DE, and especially for the higher dimensionality.

4.7 A parametric study on MDE

In this section, in order to investigate the impact of the problem modifications, some experiments are conducted. Three different versions of MDE algorithm have been tested and compared against the proposed ones.

Version 1: to study the effect of the proposed modifications for the self adaptive parameter with the basic mutation DE/rand/1/bin strategy. This version denotes the MDE1.
Version 2: to study the effect of the proposed modifications for the new mutation rule with the basic parameter setting (F = 0.5; CR = 0.9). This version denotes the MDE2.

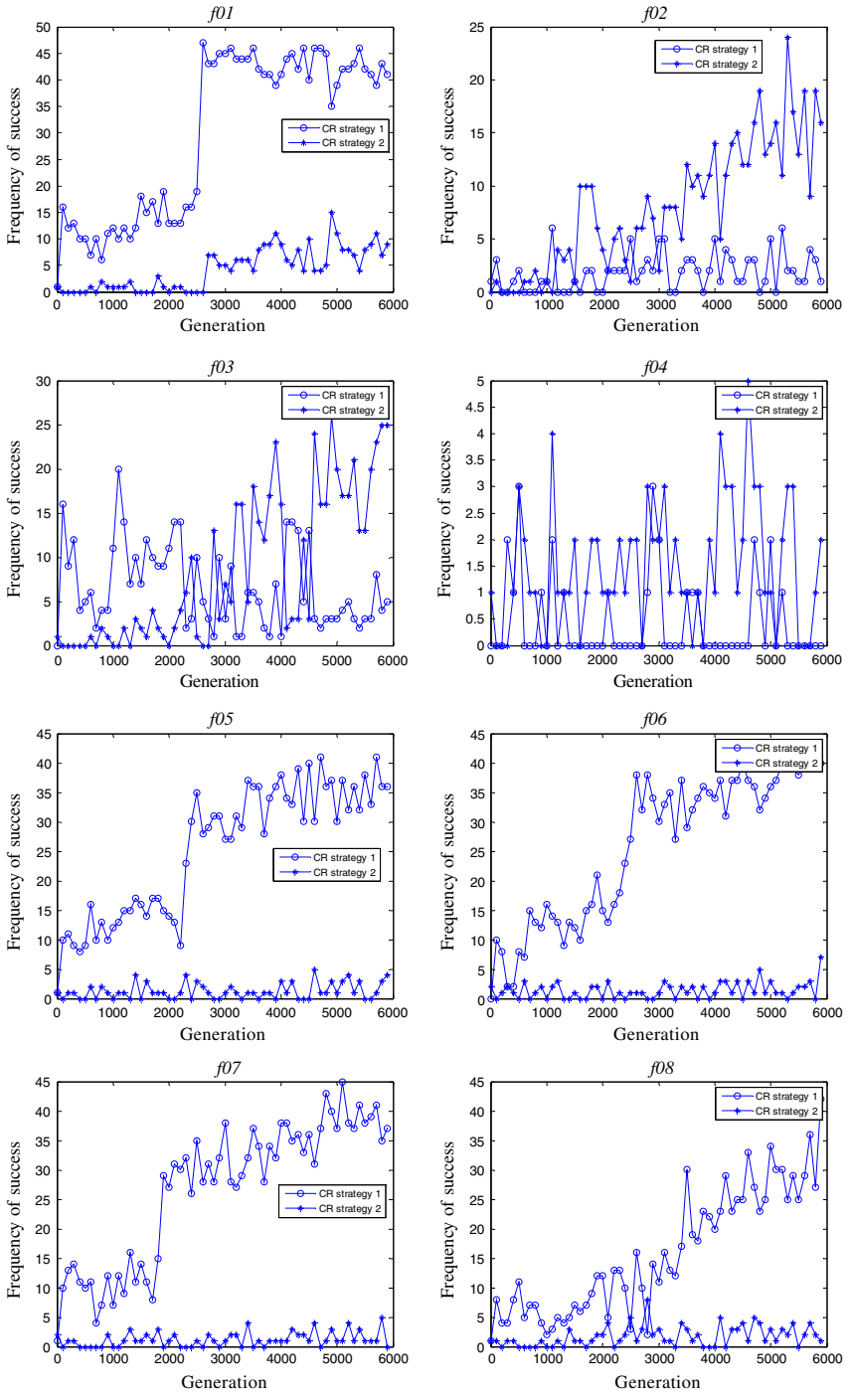

Fig. 1 The success rate of one of the two parameter settings for *f01-f08*

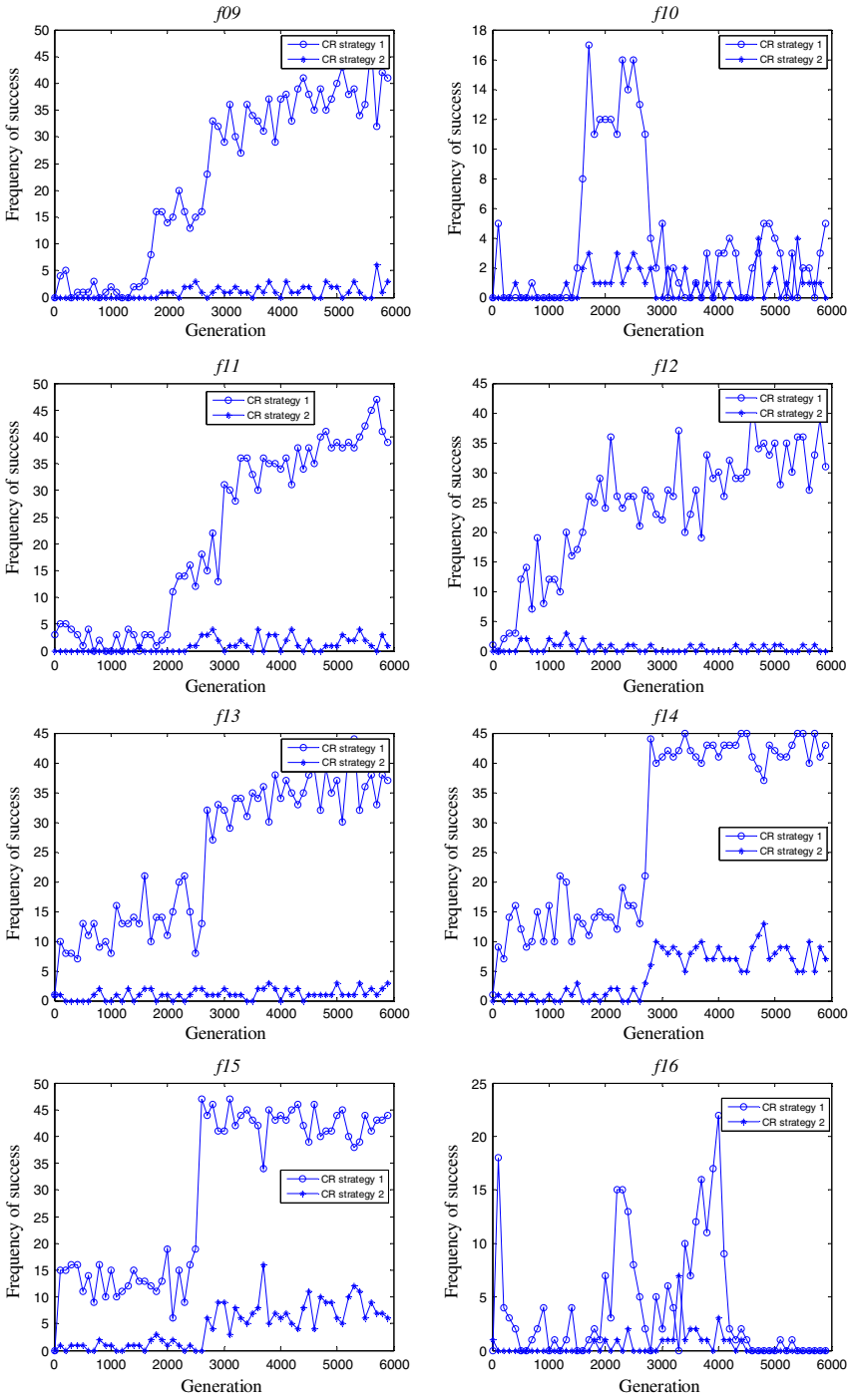

Fig. 2 The success rate of one of the two parameter settings for *f09-f16*

Version 3: modified differential evolution algorithm with self adaptive parameters method

In order to evaluate the final solution quality produced by all algorithms, the performance of these three different versions of MDE algorithms are investigated based on the 30D functions. The results of the MDE algorithm against its versions and the standard DE algorithm are summarized in Table 11. As can be seen in Table 11, we can find that the self adaptive parameter settings can enhance the standard DE effectively. Then the new mutation rule can enhance the results based on the multimodal functions. Accordingly, the main benefits of the proposed modification can balance between the exploration and exploitation ability through the evolutionary process. Figures 1 and 2 show the success rate of one of the two parameter settings for all functions. As can be seen in figures, the value of CR1 performs better than the functions f01, f05–f09, f11 and f13–f16. For f02, f03, and f12, CR2 is the winner compared with the CR1. For f04 and f10, they have the similar effect for the final results. All in all, the self adaptive method is very suitable for solving these global problems.

5 Conclusions

The performance of DE depends on the selected mutation strategy and its associated parameter values. In this paper, we propose a modified differential evolution with the self adaptive parameter settings for solving unconstrained global real-parameter optimization problems in the continuous domain. In our paper, the proposed algorithm proposes two mutation rules based on the rand and best individuals among the entire population. In order to balance the exploitation and exploration of the algorithm, the new rules are combined with the two mutation strategies through a probability rule. Then, a self adaptive parameter is introduced as uniform random numbers to enhance the diversity of the population based on the relative success ratio of the new proposed parameter setting in a previous period. In other aspect, our algorithm has a very simple structure and thus it is easy to implement. To verify the performance of MDE, 16 benchmark functions chosen from literature are employed. The results show that the proposed MDE algorithm clearly outperforms the standard DE with six different parameter settings. Compared with some evolution algorithms from the literature, we find our algorithm is superior to or at least highly competitive with these algorithms.

In this paper, we only consider the global optimization. The algorithm can be extended to solve other problems such as constrained optimization problems.

References

Brest J, Greiner S, Boskovic B, Mernik M, Zumer V (2006) Self adapting control parameters in differential evolution: a comparativestudy on numerical benchmark problems. IEEE Trans Evolut Comput 10(6):646–657

Clerc M, Kennedy J (2002) The particle swarm-explosion, stability, and convergence in a multidimensional complex space. IEEE Trans Evol Comput 6:58–73

Das S, Suganthan PN (2011) Differential evolution: a survey of the atate-of-the-art. IEEE Trans. Evolut Comput 15(1):4–31
Das S, Abraham A, Chakraborty UK, Konar A (2009) Differential evolution using a neighborhood based mutation operator. IEEE Trans Evol Comput 13(3):526–553
Dorigo M, Maniezzo V, Colorni A (1996) The ant system: optimization by a colony of cooperating agents. IEEE Trans Syst Man Cybern Part B 26(1):29–41
Garcia-Martinez C, Lozano M, Herrera F, Molina D, Sanchez AM (2008) Global and local real-coded genetic algorithms based on parent-centric crossover operators. Eur J Oper Res 185:1088–1113
Ghosh A, Das S, Chowdhury A, Giri R (2011) An improved differential evolution algorithm with fitness-based adaptation of the control parameters. Inf Sci 181(18):3749–3765
Gong WY, Cai ZH, Jiang LX (2008) Enhancing the performance of differential evolution using orthogonal design method. Appl Math Comput 206(1):56–69
Gong W, Cai Z, Ling CX (2010) DE/BBO: a hybrid differential evolution with biogeography-based optimization for global numerical optimization. Soft Comput. 15(4):645–665
Hansen N, Ostermeier A (2001) Completely derandomized self adaptation in evolution strategies. Evol Comput 9(2):159–195
Horn J, Nafpliotis N, Goldberg DE (1994) A niched Pareto genetic algorithm for multiobjective optimization. Evol Comput 1:82–87
Li XT, Wang JN, Yin MH (2013) Enhancing the performance of cuckoo search algorithm using orthogonal learning method. Neural Comput Appl. doi:10.1007/s00521-013-1354-6
Liang JJ, Qin AK, Suganthan PN, Baskar S (2006) Comprehensive learning particle swarm optimizer for global optimization of multimodal functions. IEEE Trans Evol Comput 10(3):281–295
Liu J, Lampinen J (2005) A fuzzy adaptive differential evolution algorithm. Soft Comput Fusion Found Methodol Appl 9(6):448–462
Mallipeddi R, Mallipeddi S, Suganthan PN (2010) Ensemble strategies with adaptive evolutionary programming. Inf Sci 180(9):1571–1581
Montgomery J, Chen S (2010) An analysis of the operation of differential evolution at high and low crossover rates. In: IEEE Congress on Evolutionary Computation (CEC), IEEE, pp 1–8
Neri F, Tirronen V (2009) Scale factor local search in differential evolution. Memetic Comput J 1(2):153–171
Noman N, Iba H (2008) Accelerating differential evolution using an adaptive local search. IEEE Trans Evol Comput 12(1):107–125
Omran MGH, Engelbrecht AP, Salman A (2007) Differential evolution based particle swarm optimization. IEEE Swarm Intel. Symp. (SIS 2007) 4:112–119
Qin AK, Huang VL, Suganthan PN (2009) Differential evolution algorithm with strategy adaptation for global numerical optimization. IEEE Trans Evolut Comput 13(2):398–417
Rahnamayan S, Tizhoosh HR, Salama MMA (2008) Opposition-based differential evolution. IEEE Transactions on Evolutionary Computation 12(1):64–79
Simon D (2008) Biogeography-based optimization. IEEE Trans Evol Comput 12(6):702–713
Storn R, Price K (1997) Differential evolution—a simple and efficient heuristic for global optimization over continuous space. J Global Optim 11:341–359
Suman B (2004) Study of simulated annealing based algorithms for multiobjective optimization of a constrained problem. Comput Chem Eng 8:1849–1871
Sun J, Zhang Q, Tsang E (2004) DE/EDA: a new evolutionary algorithm for global optimization. Inf Sci 169:249–262
Wang Y, Cai ZX, Zhang QF (2011a) Differential evolution with composite trail vector generation strategies and control parameters. IEEE Trans Evol Comput 15(1):55–66
Wang Y, Cai ZX, Zhang QF (2011b) Enhancing the search ability of differential evolution through orthogonal crossover. Inf Sci 18(1):153–177
Yang XS, Deb S (2009) Cuckoo search via Levy flights. In: World Congress on Nature & Biologically Inspired Computing (NaBIC 2009). IEEE Publication, USA, pp 210–214
Yang Z, He J, Yao X (2008) Making a difference to differential evolution. In: Michalewicz Z, Siarry P (eds) Advances in metaheuristics for hard optimization. Springer, Berlin, pp 397–414
Zhang Q, Muhlenbein H (2004) On the convergence of a class of estimation of distribution algorithms. IEEE Trans Evol Comput 8(2):127–136
Zhang J, Sanderson AC (2009) JADE: adaptive differential evolution with optional external archive. IEEE Trans Evolut Comput 13(5):945–958