Applied Soft Computing 47 (2016) 577–599

Contents lists available at ScienceDirect

Applied Soft Computing

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

A novel hybrid differential evolution algorithm with modified CoDE and JADE

Genghui Li, Qiuzhen Lin*, Laizhong Cui, Zhihua Du, Zhengping Liang, Jianyong Chen, Nan Lu, Zhong Ming

College of Computer Science and Software Engineering, Shenzhen University, Shenzhen, PR China

ARTICLE INFO

Article history:
Received 17 August 2015
Received in revised form 21 May 2016
Accepted 10 June 2016
Available online 18 June 2016

Keywords:
Differential evolution
Modified JADE
Modified CoDE
Hybrid framework
Global numerical optimization

ABSTRACT

JADE and CoDE are two well-known state-of-the-art DE algorithms for solving global optimization problems (GOPs). JADE is found to be suitable for solving unimodal and simple multimodal functions as an exploitation mutation strategy, *i.e.*"DE/current-to-*p*best/1", is employed, while CoDE is shown to fit for handling complicated multimodal functions due to its exploration mutation strategies, such as "DE/rand/1/bin", "DE/currant-to-rand/1", and "DE/rand/2/bin". To combine their merits for tackling different types of GOPs, a novel hybrid framework is designed based on the modified JADE (MJADE) and modified CoDE (MCoDE), named HMJCDE. Different from the simple combination of MJADE and MCoDE, they are operated alternatively according to the improvement rate of the fitness value. To assess the performance of HMJCDE, 30 benchmark problems taken from CEC2014 competition on real parameter optimization are employed. When compared with JADE, CoDE, other state-of-the-art DE variants and non-DE heuristic algorithms, HMJCDE performs better than all the competitors on most of test problems. Moreover, the sensitivity analysis of some parameters in HMJCDE is conducted and the effectiveness of our proposed hybrid framework is also justified experimentally.

1. Introduction

In many scientific researches and engineering applications, the problems to find a global optimal value, are often encountered. Such problems are generally termed global optimization problems (GOPs). Without loss of generality, GOPs that are specified for minimization with no other constraints can be mathematically defined as follows.

$$\min \quad f(X), \quad X = (x_1, \quad x_2, \quad \cdots, \quad x_D) \tag{1}$$

where $f \ : \ R^D \to R$ defines a real-valued objective function, and D is the number of the decision variables. The target of Eq. (1) is to find an optimal solution X^*, which should satisfy the condition that $f(X^*) \leq f(X)$ for $\forall X \in R^D$.

Due to the complex characteristics such as multimodality, highly non-linearity, non-differentiability, with steep and flat search regions in difficult optimization problems [1], conventional mathematical and analytical methods are not so suitable for solving such kind of optimization problems. Therefore, nature-inspired heuristic algorithms such as genetic algorithm (GA) [2], particle swarm optimization (PSO) [3–5], artificial bee colony (ABC) [6,7], artificial immune algorithm [8,9], harmony search (HS) [10] and differential evolution (DE) [11–13], are designed to tackle the difficult optimization problems. Especially, DE has been shown to be a very effective and efficient approach for dealing with such problems, which attracts a great attention of scientific researchers. Since the first DE algorithm was reported by Price and Storn [11] in 1997, one important discussion about DE is how to balance its capabilities of exploration and exploitation. On one hand, the excessive exploration can maintain the population diversity, but can easily lead to a slow convergence speed or cause the stagnation phenomenon. On the other hand, too much exploitation can result in premature convergence or easily fall into the local optimum. Over the past twenty years, a lot of research works have been conducted to alleviate this weakness, and thus, many state-of-the-art DE variants were developed for this purpose, including JADE [14] and CoDE [15]. In JADE, a greedy mutation strategy "DE/current-to-*p*best/1" with optional external archive and an adaptive parameter method were designed. Both of the best solution and the other good solutions are utilized in "DE/current-to-*p*best/1", and the external archive not only can provide the progress direction but also can improve the diversity of the population. The adaptive parameter method

* Corresponding author.
E-mail addresses: qiuzhlin@szu.edu.cn, qiuzhlin@qq.com (Q. Lin).

http://dx.doi.org/10.1016/j.asoc.2016.06.011
1568-4946/© 2016 Elsevier B.V. All rights reserved.

lets the scaling factor *F* and the crossover rate *CR* be generated following the Cauchy and Gaussian distributions respectively. Their expected mean values are adaptively updated using the successful values of *F* and *CR*. These approaches enable JADE to handle many kinds of GOPs. However, the greedy mutation strategy adopted in JADE makes it more suitable for solving unimodal and simple multimodal functions, as justified by the experimental results [15]. On the other hand, CoDE combines three well-studied trial vector generation strategies (*i.e.*, "DE/rand/1/bin", "DE/rand/2/bin", "DE/current-to-rand/1") with three control parameter settings (*i.e.*, [F = 1.0, CR = 0.9], [F = 1.0, CR = 0.1], [F = 0.8, CR = 0.2]) in a random way to generate trial vectors. These non-greedy DE strategies and the non-fixed parameters settings enable CoDE to fit for tackling multimodal functions. Due to the fact that a random combination of three well-known DE strategies and three common parameter settings in CoDE can further enhance the optimization performance, it also motivates us to study whether the two well-studied DE algorithms, i.e., JADE and CoDE, can be effectively mixed in order to combine their advantages when solving different kinds of GOPs. Therefore, in this paper, a novel hybrid framework is designed to combine the merits of modified JADE and modified CoDE. To respectively enhance the exploitation and exploration capabilities of original JADE and CoDE, Gaussian mutation is embedded into JADE and an external archive is introduced for CoDE. Besides that, the three control parameter settings in CoDE are replaced by using Cauchy and Gaussian distributions to generate the values of *F* and *CR* respectively. Briefly, the contributions of this paper can be summarized as follows.

(1) The original JADE is modified by the embedment of Gaussian mutation on the best individual, and consequently, the exploitation ability of original JADE is enhanced. Moreover, the adaptive parameter approach in original JADE is simply modified to improve its adaption. This modified JADE is called MJADE in this paper.
(2) The original CoDE is modified by the introduction of an external archive, with the aims of enhancing the population diversity and exploration ability of original CoDE. Besides that, Cauchy distribution and Gaussian distribution are respectively employed for the generation of *F* and *CR* according to the three control parameter settings in original CoDE. This modified CoDE is called MCoDE in this paper.
(3) A novel hybrid framework combining MJADE and MCoDE is accordingly proposed, namely HMJCDE. HMJCDE is designed to alternatively execute MJADE and MCoDE based on the improvement rate. Experimental results (presented in Section 5) justify that HMJCDE performs better than the single use of JADE or CoDE, and the simple combinations of MJADE and MCoDE.

To validate the advantage of our proposed algorithm, many experiments are conducted on 30 test functions derived from the special session on real-parameter optimization of CEC2014 [16]. When compared with nine DE variants, *i.e.*, CoDE [15], JADE [14], ODE [17], SaDE [18], sinDE [19], AuDE [20], DE-VNS [21], AEPD-JADE [22] and rank-jDE [23], and five non-DE heuristic algorithms, *i.e.*, EHS [10], CLPSO [3], TLBO [24], ABC [7] and CMA-ES [25], HMJCDE has demonstrated the superiority on most of test problems. Moreover, the parameter sensitivity of HMJCDE is analysed experimentally and HMJCDE is further compared with two other hybrid frameworks of MJADE and MCoDE to verify the advantage of our proposed hybrid framework.

The remainder of this paper is organized as follows. In Section 2, we briefly introduce the basic DE algorithm. Section 3 gives a review on the DE algorithms for solving GOPs. The details of HMJCDE are described in Section 4, which includes the introduction of MJADE, MCoDE, and the proposed hybrid framework. The experimental results are presented and discussed in Section 5, which compares HMJCDE with nine DE variants and five non-DE heuristic algorithms. Finally, Section 6 concludes this paper.

2. Basic differential evolution algorithm

Differential evolution is a branch of EA, which follows the similar evolutionary procedures of EA, including mutation, crossover and selection. Generally, the new candidate solutions (offspring) are produced through mutation and crossover operators. After that, the parents will be replaced by the offspring in selection procedure when a better fitness value is found by the offspring. The pseudo-code of basic DE algorithm is shown in **Algorithm 1** (Fig. 1), where *NP* is the population size, *F* is the mutation scaling factor, *CR* is the crossover rate, *D* is the number of decision variables, *G* is the generation time; $X_{r1}^G, X_{r2}^G, X_{r3}^G$ are the three distinct individuals randomly picked out from the population at *G* generation, which are not equal to the target vector X_i^G; $randint(1, \quad D)$ generates a uniformly distributed random integer number in [1, *D*] and *rand*(0, 1) returns a uniformly distributed random real number in [0, 1]; $X_j^{\min}$ and $X_j^{\max}$ are the lower and upper bounds of the *j*th variable respectively.

As described in **Algorithm 1**, after the initialization process to generate an initial population, DE turns into the evolutionary loop including the procedures of mutation, crossover, and selection, until the termination condition is met. The detail introductions of these three crucial operators are given as follows.

Mutation operator: DE adopts the mutation operator to generate the mutant vector (also called donor vector) $V_i^G = (V_{i,1}^G, \quad V_{i,2}^G, \quad \cdots, \quad V_{i,D}^G)$ with respect to each individual X_i^G (called target vector). Recently, there are many mutation schemes reported in DE [11,12]. In order to distinguish among these schemes, the notation "DE/*x*/*y*/*z*" is used, where *x* represents the vector (also called basic vector) to be perturbed, *y* denotes the number of difference vectors considered for perturbation and *z* is the crossover scheme (*e.g.*, *bin*: binomial crossover and *exp*: exponential crossover). Some well-known mutation strategies are listed as follows.

$$\text{DE/rand/1}: V_i^G = X_{r1}^G + F \cdot (X_{r2}^G - X_{r3}^G) \tag{2}$$

$$\text{DE/rand/2}: V_i^G = X_{r1}^G + F \cdot (X_{r2}^G - X_{r3}^G) + F \cdot (X_{r4}^G - X_{r5}^G) \tag{3}$$

$$\text{DE/best/1}: V_i^G = X_{best}^G + F \cdot (X_{r1}^G - X_{r2}^G) \tag{4}$$

$$\text{DE/rand-to-best/1}: V_i^G = X_{r1}^G + F \cdot (X_{best}^G - X_{r2}^G) + F \cdot (X_{r3}^G - X_{r4}^G) \tag{5}$$

$$\text{DE/current-to-rand/1}: V_i^G = X_i^G + F \cdot (X_{r1}^G - X_i^G) + F \cdot (X_{r2}^G - X_{r3}^G) \tag{6}$$

$$\text{DE/current-to-best/1}: V_i^G = X_i^G + F \cdot (X_{best}^G - X_i^G) + F \cdot (X_{r2}^G - X_{r3}^G) \tag{7}$$

where X_{best}^G is the best individual in current generation *G*, *r*1, *r*2, *r*3, *r*4 and *r*5 are randomly generated from [1, *NP*], and $r1 \neq r2 \neq r3 \neq r4 \neq r5 \neq i$. The parameter *F* is called the scaling factor controlling the mutation scale, which is generally restricted in the range (0, 1]. Generally, different mutation strategies have distinct characteristics, which have the advantages for solving certain kinds of problems [18].

Crossover operator: After mutation, a trial vector $U_i^G = (U_{i,1}^G, \quad U_{i,2}^G, \quad \cdots, \quad U_{i,D}^G)$ is obtained according to a binomial crossover operator on X_i^G and V_i^G, as follows.

Algorithm 1: basic DE algorithm

```
01: Generate a uniformly distributed random initial population including $NP$ solutions that contain
    $D$ variables according to $X_{i,j}^{0} = X_{j}^{\min} + rand(0,1) \cdot \left(X_{j}^{\max} - X_{j}^{\min}\right)$   $(i \in [1, NP], j \in [1, D])$
02: while termination condition is not satisfied
03:   for $i$=1 to $NP$
04:     Generate three random indexes $r1$, $r2$ and $r3$ with $r1 \neq r2 \neq r3 \neq i$ //mutation
05:     $V_{i}^{G} = X_{r1}^{G} + F \cdot \left(X_{r2}^{G} - X_{r3}^{G}\right)$ //end mutation
06:     $j_{rand} = randind(1, D)$ //crossover
07:     for $i$=1 to $D$
08:       if $rand(0,1) \le CR$ or $j == j_{rand}$
09:         $U_{i,j}^{G} = V_{i,j}^{G}$
10:       else
11:         $U_{i,j}^{G} = X_{i,j}^{G}$
12:       end if
13:     end for //end crossover
14:     if $f(U_{i}^{G}) \le f(X_{i}^{G})$ //selection
15:       $X_{i}^{G+1} = U_{i}^{G}$
16:     else
17:       $X_{i}^{G+1} = X_{i}^{G}$
18:     end if //end selection
19:   end for
20: end while
```

Fig. 1. The pseudo-code of the basic DE algorithm (with DE/rand/1/bin).

$$U_{i,j}^{G} = \begin{cases} V_{i,j}^{G} & \text{if } (rand_{i,j}(0,1) \le CR \text{ or } j == j_{rand}) \\ X_{i,j}^{G} & \text{otherwise} \end{cases} \tag{8}$$

where $rand_{i,j}(0, 1)$ is a uniformly distributed random real number in [0,1] and j_{rand} is a uniformly distributed random integer in [1,D]. The use of j_{rand} can make sure that at least one variable of trial vector is inherited from the mutant vector V_i^G. If the jth variable $U_{i,j}^G$ of the trial vector U_i^Gviolates the boundary constraints, it will be reset as follows.

$$U_{i,j}^{G} = \begin{cases} \min\{X_j^{\max}, \ 2X_j^{\min} - U_{i,j}^{G}\} & \text{if } U_{i,j}^{G} < X_j^{\min} \\ \max\{X_j^{\min}, \ 2X_j^{\max} - U_{i,j}^{G}\} & \text{if } U_{i,j}^{G} > X_j^{\max} \end{cases} \tag{9}$$

Selection operator: By simulating the rule of natural evolution, selection operator will keep the better one from the target vector X_i^G and the trial vector U_i^G. This chosen vector will survive and enter the next evolutionary generation. Without loss of generality, for a minimization problem, the vector with better fitness value (*i.e.*, lower objective value) is selected, as follows.

$$X_i^{G+1} = \begin{cases} U_i^G & \text{if } f(U_i^G) \le f(X_i^G) \\ X_i^G & \text{otherwise} \end{cases} \tag{10}$$

It is noted that when the trial vector U_i^G is better than or equal to the parent X_i^G, it is called a successful update. Otherwise, it is marked as an unsuccessful update.

3. Relevant works

Over the past twenty years, a lot of in-depth research works have been conducted on DE to achieve a better performance in solving GOPs. A brief survey on the improved DE approaches can be generally classified into four categories, *i.e.*, novel mutation and crossover operators in DE, parameters and strategies control in DE, general DE operators and the hybrid of DE with other methods. They are respectively introduced as follows.

3.1. Novel mutation and crossover operators in DE

Fan et al. [26] reported a novel trigonometric mutation strategy operator. In this strategy, the vector to be perturbed is not an individual randomly selected from the current population, but the central point of three randomly selected vectors. The perturbation parts are contributed together as the three legs of the triangle and three weight terms are adopted to guide the evolutionary direction towards a promising area. This approach is combined with "DE/rand/1/bin" to get a better balance between the convergence and the robustness. Kaelo and Ali [27] designed a hybrid DE strategy by utilizing the attraction-repulsion concept of electromagnetism-like algorithm. In this approach, the individuals with better fitness values attract others, while those with inferior fitness values repel others. Das et al. [28] proposed a novel mutation strategy based on the neighbourhoods (DEGL). In this method, the best one among the neighbours and the global best individual are cooperatively used to guide the evolutionary direction. Zhang et al. [14] presented a new DE mutation operator using the optional external archive (JADE), named "DE/current-to-*p*best/1". This approach exploits the beneficial information from the top ($p \times 100\% \times NP$) individuals in population (both the global best individual and other competitive individuals). Islam et al. [29] designed a new DE mutation operator, called "DE/current-to-gr_best/1". Similar to "DE/current-to-*p*best/1", this operator employs a group of good individuals randomly selected from the current population to guide the evolutionary direction collectively. Yu et al. [30] proposed a novel mutation strategy named "DE/lbest/1" (ADE). In this strategy, the evolution of each individual is guided by multiple local best individuals instead of the global best individual. The mutation operators proposed in [14,28–30] are the generalization of "DE/current-to-best/1" or "DE/best/1", and they utilize the information of the best individual and other good individuals to compensate the lack of the global exploration ability in "DE/current-to-best/1" and "DE/best/1". Unlike the multiple mutation strategies used in some state-of-the-art DE variants [12,15,18], Ji et al. [20] employed a single equation unifying multiple mutation strategies into one expression, called AuDE. Cui et al. [31] proposed an adaptive differential evolution algorithm with novel mutation strategies in multiple subpopulations (MPADE). This approach divides the parent population into three sub-populations based on the fitness values and then performs three novel DE strategies to take on the responsibility for either exploitation or exploration.

Islam et al. [29] proposed a novel *p*-best crossover, with a biased parent selection in a usual binomial crossover. This approach allows a mutant vector to exchange its components with an individual randomly selected from the p top-ranked individuals in current population. Wang et al. [32] reported an orthogonal crossover operator based on the orthogonal design. This method performs a systematic and rational search in a region defined by the parent solutions. The aim of this orthogonal design is to repair the weakness of the binomial crossover and exponential crossover that can only generate a trial vector in a hyper-rectangle defined by the mutant and target vectors. Guo et al. [33] designed an eigenvector-based crossover operator. This scheme makes the crossover rotationally invariant by utilizing the eigenvectors of the individuals' covariance matrix, and consequently, this operator significantly improves the optimization performance. Wang et al. [34] established an Eigen coordinate system with loose variable correlation for crossover operator. This approach exploits the statistical information based on the covariance matrix of some good solutions, so as to relieve the dependence of DE on the coordinate system and enhance the performance of DE in solving such kind of optimization problems with high variable correlation.

3.2. Parameters and strategies control in DE

In [35], Liu et al. proposed a fuzzy adaptive DE (FADE), which adopts fuzzy logic controllers to adjust the parameters F and CR. Due to the fact that the better control parameters having higher opportunity to produce superior individuals, Brest et al. [36] presented a self-adaptation scheme called jDE. In this strategy, both F and CR are also encoded into the individuals, and updated relying on two pre-defined parameters τ_1 and τ_2. In some reported approaches, such as JADE [14], MDE-*p*best [29], SHADE [37] and L-SHADE [38], F and CR are adaptively produced from Cauchy distribution (*i.e.*, $randc(F_m,\quad 0.1)$) and Gaussian distribution (*i.e.*, $randn(CR_m,\quad 0.1)$) respectively. The expected mean parameters F_m and CR_m are updated according to the successful F and CR values that are able to effectively produce the better offspring. Yu et al. designed a novel two-level adaptive parameter control scheme for DE [30]. The first step is to compute the population-level parameters F_P and CR_P for the whole population based on the exploration and exploitation statuses. These statuses are estimated by measuring the population distribution. The second step is to calculate the individual-level parameters F_i and CR_i for each individual, by adjusting F_P and CR_P according to the individual's fitness value and its distance to the global best individual. Draa et al. [19] reported a new Sinusoidal DE (SinDE) by using new sinusoidal formulas to automatically adjust the values of F and CR. The authors claimed that SinDE is especially effective for solving multimodal and composition functions. Sarker et al. [39] designed a new mechanism to dynamically select the best combination of parameter settings (*i.e.* scaling factor F, crossover rate CR and population size NP) for a specific problem in each single run. The experimental results demonstrated its superior performance over the state-of-the-art DE variants.

Qin et al. [18] proposed a self-adaptive DE algorithm (SaDE). In this approach, the DE mutation strategies are gradually self-adapted by learning from their previous search experiences. Mallipeddi et al. [12,40,41] designed the ensemble differential evolution algorithms (EPSDE). In these algorithms, a pool of distinct DE mutation strategies, crossover strategies and a pool of associated values for the control parameters not only coexist throughout the evolutionary process, but also compete with each other to produce offspring. Wang et al. [15] put forward a composite DE (CoDE) algorithm. This approach combines three well-studied DE strategies with three control parameters settings in a random way to produce the trial vector. Recently, they also used DE to solve constrained optimization problems [42] and multimodal optimization problems [43]. Yang et al. [44] introduced a self-adaptive clustering-based DE with composite trial vector generation strategies (SaCoCDE). This scheme divides the evolved population into different subsets using a clustering algorithm, and then mutates each subset using certain kind of DE mutation strategy, which exploits the cluster center and the best solution in the current population. Elsayed et al. [45] introduced a novel framework by using multiple search operators in each generation. In this framework, a self-adaptive multi-operator generic algorithm (SAMO-GA) and a self-adaptive multi-operators differential evolution (SAMO-DE) were proposed. Then, SAMO-DE was further extended in [46,47]. Recently, the concept of training and testing was further used in SAMO-GA to find a set of suitable parameters for different operators [48]. Additionally, Elsayed et al. [49] designed a novel algorithmic framework, by integrating a mix of four different DE mutation strategies, two crossover operators and two constraint handing techniques for tackling constrained optimization problem (COPs). The experimental results show that the proposed algorithm is better than other state-of-the-art algorithms.

3.3. General DE operators

Recently, more and more general techniques have been reported, which can be applied to different DE algorithms. Epitropakis et al. [50] proposed a novel DE operator based on the proximity characteristics among individuals (namely Pro DE). This scheme assigns each individual a selection probability, which is inversely proportional to its distance to the mutant individual. Gong and Cai [23] designed a ranking-based DE operator (rank-DE). This approach selects some parents for DE mutation operator based on their rankings in the current population. Generally, a higher ranking will get a larger selection probability. It is further extended by Gong et al. [51] to tackle constraint optimization problems, enabling DE to achieve feasible solutions fast. Cai and Wang [52] exploits the information of neighbouring individuals (NDi-DE), with the aim of exploring the regions of minima and accelerating the convergence speed. This algorithm employs the directional information provided by the neighbours to prevent the individuals from entering an undesired region, and resultantly guide the mutant individual towards a promising area. Guo et al. [53] reported a successful-parent-selecting framework (SPSDE). This framework maintains the successful solutions in an archive, and then chooses the parents from this archive when the times that a solution is not continuously updated are larger than the unacceptable times. In this way, it can effectively help DE to escape from the stagnation status.

3.4. The hybrid of DE with other methods

DE has been hybridized with other nature-inspired heuristic algorithms, such as PSO [54], bacterial foraging [55], biogeography-based optimization [56], artificial bee colony algorithm [57] and artificial glowworm swarm algorithm [58]. These hybrid algorithms can effectively combine the merits of different approaches, and resultantly improve the comprehensive optimization performance. However, to the best of our knowledge, there is no such study to mix two state-of-the-art DE variants in order to combine their distinct advantages. This fact encourages us to study whether the performance of DE can be further improved by mixing two state-of-the-art DE variants, and consequently, a novel hybrid framework is designed in this paper, with the combination of MJADE and MCoDE.

4. The proposed algorithm HMJCDE

In this section, the details of our algorithm HMJCDE are introduced. In order to clearly describe our algorithm, the algorithmic

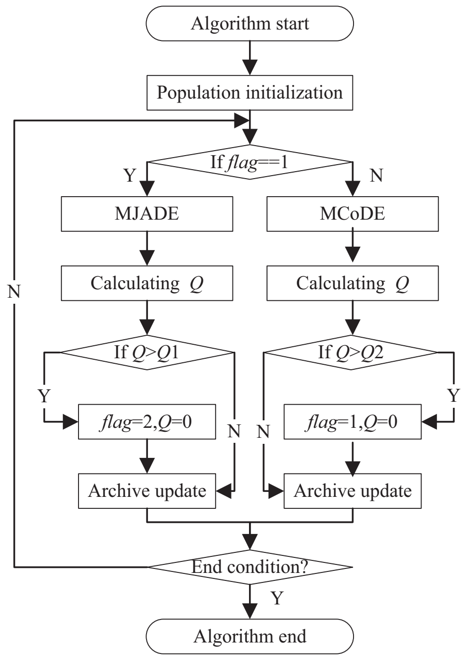

Fig. 2. The algorithmic flowchart of HMJCDE.

```
Algorithm 2: MJADE
01: Generate a uniformly distributed random population
02: Set $F_m = 0.5, CR_m = 0.5$, $p$=0.05, $count$(1:$NP$)=0, $m$=30 ,$c$=0.1
03: while termination condition is not satisfied
04:     $S_F = \varnothing, S_{CR} = \varnothing$
05:     for $i$=1 to $NP$
06:         $F_i = randc(F_m, 0.1)$,  $CR_i = randn(CR_m, 0.1)$
07:         if $count(i)<=m$
08:             $V_i^G = X_i^G + F_i \cdot (X_{pbest}^G - X_i^G) + F_i \cdot (X_{r1}^G - X_{r2}^G)$
09:         else
10:             $V_i^G = X_{best}^G + F_i \cdot randn(1, D)$
11:         end if
12:         Execute crossover operator as described in lines 06-13 of Algorithm 1
13:         if  $f(U_i^G) > f(X_i^G)$      // unsuccessful update
14:             $X_i^{G+1} = X_i^G$  , $count(i)$=$count(i)$+1
15:         else                           //successful update
16:             $X_i^{G+1} = V_i^G$,  $X_i^G \to A$,  $F_i \to S_F$,  $CR_i \to S_{CR}$, $count(i)$=0
17:         end if
18:     end for
19:     if  $S_F \neq \varnothing$  &&  $S_{CR} \neq \varnothing$
20:         $F_m = (1-c) \cdot F_m + c \cdot mean_L(S_F)$, $CR_m = (1-c) \cdot CR_m + c \cdot mean_A(S_{CR})$
21:     else
22:         $F_m = (1-c) \cdot F_m + c \cdot rand(0,1)$,  $CR_m = (1-c) \cdot CR_m + c \cdot rand(0,1)$
23:     end if
24: end while
```

Fig. 3. The pseudo-code of MJADE.

framework of HMJCDE is demonstrated in Fig. 2. After the initialization, MCoDE and MJADE are executed alternatively in each generation, which is controlled by the parameter *flag*. If *flag* = 1, MJADE will be operated. Otherwise, MCoDE will be executed. At the final steps of MJADE (or MCoDE), the parameter Q will be recalculated and it represents the number of the unsuccessful generations (the case that the improvement rate is lower than the specified parameter ε). The definition of the unsuccessful generation will be given in Subsection 4.3. If the number of unsuccessful generation Q is larger than Q1 (for MJADE) or Q2 (for MCoDE), the parameter *flag* will be updated and Q is reset to 0. At the end of each generation, the archive will be renewed. The proposed main components, including MCoDE, MJADE and archive update, are respectively introduced in the following subsections. At last, the pseudo-code of the complete algorithm HMJCDE is also depicted for further clarification.

4.1. Modified JADE: MJADE

In JADE, a new mutation strategy "DE/current-to-*p*best/1" with external archive is used, as defined by

$$V_i^G = X_i^G + F \cdot (X_{pbest}^G - X_i^G) + F \cdot (X_{r1}^G - X_{r2}^G) \tag{11}$$

where X_{pbest}^G is randomly selected from the top ($p \times 100\% \times NP$) individuals in current population P with $p \in (0, 1]$, X_{r1}^G is randomly selected from the current population P, and X_{r2}^G is randomly selected from the union $P \cup A$ (A denotes the external archive that maintains the target vector removed in selection process).

Based on "DE/current-to-*p*best/1" mutation strategy, any of top ($p \times 100\% \times NP$) individuals can be randomly chosen to guide other individuals, and the archive can maintain the diversity and provide additional information about the promising evolutionary direction. The parameter p is a user specified parameter. As revealed by our experiments in Section 5, JADE emphasizes exploitation when p is small (*e.g.*,$p \in (0, 0.2)$) and it is suitable for solving unimodal and simply multimodal functions. Thus, p is set to 0.05 in this paper. In order to further enhance the exploitation ability of JADE, Gaussian mutation is introduced into JADE as follows.

$$V_i^G = \begin{cases} X_i^G + F_i \cdot (X_{pbest}^G - X_i^G) + F_i \cdot (X_{r1}^G - X_{r2}^G) & \text{if } count(i) <= m \\ X_{best}^G + F_i \cdot randn(1, D) & \text{otherwise} \end{cases} \tag{12}$$

where X_{best}^G is the best individual in current population, *count*(i) denotes the number of recently consecutive unsuccessful updates of the target individual X_i^G, m is a user specified parameter, $randn(1, \ D)$ produces a vector of D random numbers from the standard normal distribution. The modification in Eq. (12) is verified by our experiments to improve the performance of original JADE on some of test functions.

In JADE, the scaling factor F_i and crossover rate CR_i for each individual X_i^G are independently produced according to Cauchy distribution and Gaussian distribution respectively, as follows.

$$F_i = randc(F_m, 0.1) \tag{13}$$

$$CR_i = randn(CR_m, 0.1) \tag{14}$$

where F_m and CR_m are initialized to 0.5. F_i is truncated to 1 in the case of $F_i > 1$, while F_i is regenerated in the case of $F_i < 0$. CR_i is truncated to 1 when $CR_i > 1$, while CR_i is truncated to 0 when $CR_i < 0$. In each generation, all successful F_i and CR_i are saved in the set S_F and S_{CR}. At the end of each generation, if S_F and S_{CR} are not empty, F_m and CR_m will be updated as follows

$$F_m = (1 - c) \cdot F_m + c \cdot mean_L(S_F) \tag{15}$$

$$CR_m = (1 - c) \cdot CR_m + c \cdot mean_A(S_{CR}) \tag{16}$$

where c is a positive constant in [0, 1], $mean_A(\cdot)$ is the usual arithmetic mean, and $mean_L(\cdot)$ is defined by

$$mean_L(S_F) = \sum\nolimits_{F \in S_F} F^2 / \sum\nolimits_{F \in S_F} F \tag{17}$$

Note that when the sets S_F and S_{CR} are empty, the parameters F_m and CR_m would stop update in original JADE. However, the empty of the sets S_F and S_{CR} indicates that the current parameters F_m and CR_m are unsuitable at the current stage and thus they actually need to be changed. Therefore, a modification is introduced for the case when S_F and S_{CR} are empty, and they are updated as follows.

$$F_m = (1 - c) \cdot F_m + c \cdot rand(0, 1) \tag{18}$$

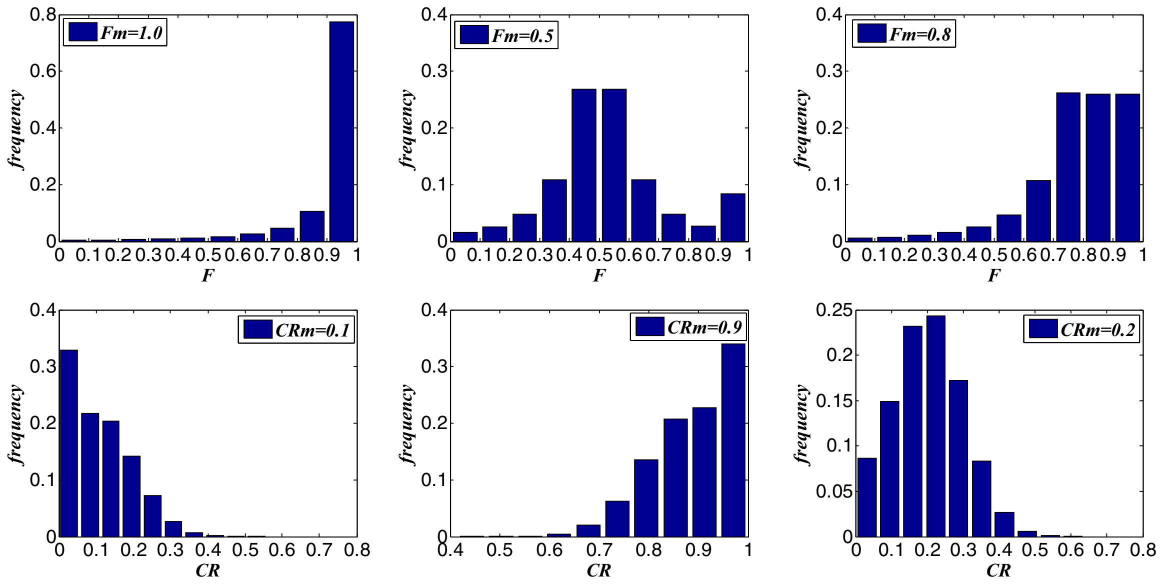

Fig. 4. Frequency distribution histogram of *F* and *CR* with different parameter settings.

$$CR_m = (1 - c) \cdot CR_m + c \cdot rand(0, 1) \tag{19}$$

By adding our proposed modification in original JADE, the pseudo-code of the complete MJADE algorithm is shown in Fig. 3.

4.2. Modified CoDE: MCoDE

Trial vector generation strategy (including mutation and crossover) and control parameters (*i.e.*, *F* and *CR*) have significant impacts on the performance of DE. CoDE integrates the advantages of different DE strategies by combining three well-studied trial vector generation strategies with three control parameter settings in a random way. These three selected trial vector generation strategies are “DE/rand/1/bin”, “DE/rand/2/bin”, and “DE/current-to-rand/1”. Note that the binominal crossover operator is not applied in “DE/current-to-rand/1”. Moreover, the three control parameter settings for *CR* and *F* are used, *i.e.*, [F = 1.0, CR = 0.1], [F = 1.0, CR = 0.9], and [F = 0.8, CR = 0.2]. Due to the use of “DE/rand/1/bin”, “DE/rand/2/bin” and “DE/current-to-rand/1” in CoDE, it emphasizes the exploration capability and thus is more suitable for solving complex multimodal functions, which is also confirmed by our experimental study in Section 5. In order to enhance the exploration ability of CoDE, some modifications are further introduced into CoDE to form MCoDE.

In MCoDE, “DE/rand/1/bin”, “DE/rand/2/bin” and “DE/current-to-rand/1” are also used to produce trial vectors. However, an external archive **A** is embedded into MCoDE for the enhancement of exploration capability, which records the target individuals defeated by their associated trial vectors in selection process. Regarding “DE/rand/1/bin” (defined in Eqs. (2) and (8)) and “DE/rand/2/bin” (defined in Eqs. (3) and (8)), X_{r1}^G is randomly selected from *P*, while X_{r2}^G, X_{r3}^G, X_{r4}^G and X_{r5}^G are randomly chosen from $P \cup A$. For “DE/current-to-rand/1” (defined in Eq. (6)), X_{r1}^G, X_{r2}^G and X_{r3}^G are all randomly picked out from $P \cup A$.

Moreover, similar to MJADE, the parameters *F* and *CR* in MCoDE are also sampled from Cauchy and Gaussian distributions, as defined in Eqs. (13) and (14). Three parameter settings for F_m and CR_m used in Eqs. (13) and (14) are given in MCoDE, as follows

$$[F_m = 1.0, CR_m = 0.1],\ [F_m = 0.5, CR_m = 0.9],\ [F_m = 0.8, CR_m = 0.2]$$

As our parameters *F* and *CR* are sampled from the Cauchy and Gaussian distributions, the final values of *F* and *CR* are near to the original parameters of CoDE with a high probability, so that our modified scheme obtains similar performance to original CoDE. Moreover, our *F* and *CR* values will diverge the original parameters of CoDE with a small probability and make our MCoDE obtain better exploration and adaptation. In order to validate the above viewpoints, 300,000 samples of *F* and *CR* are generated by the above three parameter settings respectively. The frequency distribution histograms are illustrated in Fig. 4. It is noted that the second parameter setting [F_m = 0.5, CR_m = 0.9] in MCoDE is used to replace the second one ([F = 1.0, CR = 0.9]) in CoDE as [F = 0.5, CR = 0.9] is one of the most commonly used parameter settings in different DE variants. In summary, our modifications are beneficial for improving the exploration and adaptation capabilities of CoDE. The pseudo-code of the complete MCoDE algorithm is illustrated in Fig. 5.

From Fig. 5, it is clear that the parameter settings of MCoDE are different from that of CoDE. The other difference is the selection of parents for the trial vector generation strategy. In MCoDE, some parents are randomly picked out from the union of current population and the archive $P \cup A$, while in CoDE, all parents are randomly selected from the population *P*.

4.3. The calculation of Q and archive update

4.3.1. The calculation of Q

As described in the above subsections, MCoDE is suitable for solving complicated multimodal functions due to its exploration capability, while MJADE is good at tackling unimodal and simply multimodal functions because of its exploitation ability. Therefore, in this paper, the advantages of two effective DE variants (*i.e.*, MCoDE and MJADE) with distinct characteristics are mixed in order to further enhance their performance. For this purpose, a hybrid framework for MCoDE and MJADE is designed, which runs MJADE and MCoDE alternatively according to the evolutionary situation. In order to clearly describe our hybrid framework, some definitions are given as follows.

Definition 1. **The Improvement Rate (*IR*)** indicates the fitness improvement at generation *G*, as defined by

$$IR = \frac{f_{best}^{G-1} - f_{best}^{G}}{f_{best}^{G-1}} \tag{20}$$

```
Algorithm 3: MCoDE
01: Generate a uniformly distributed random population
02: Set the parameter candidate pools [$F_m$=1.0, $CR_m$=0.1], [$F_m$=0.5, $CR_m$=0.9], [$F_m$=0.8, $CR_m$=0.2]
03: while termination condition is not satisfied
04:     for $i$=1 to $NP$
05:         Randomly select a set of parameter values for $F_m$ and $CR_m$
06:         $F_1 = randc(F_m, 0.1)$, $CR_1 = randn(CR_m, 0.1)$
07:         Generate trial vector $U_1^G$ by using "DE/rand/1/bin" (Eqs. (2) and (8))
08:         Randomly select a set of parameter values for $F_m$ and $CR_m$
09:         $F_2 = randc(F_m, 0.1)$, $CR_2 = randn(CR_m, 0.1)$
10:         Generate trial vector $U_2^G$ by using "DE/rand/2/bin" (Eqs. (3) and (8))
11:         Randomly select a set of parameter values for $F_m$ and $CR_m$
12:         $F_3 = randc(F_m, 0.1)$, $CR_3 = randn(CR_m, 0.1)$
13:         Generate trial vector $U_3^G$ by using "DE/current-to-rand/1" (Eq. (6))
14:         Choose the best trial vector (denoted as $U$) from the above three trial vectors
15:         if $f(U) > f(X_i^G)$   // unsuccessful update
16:             $X_i^{G+1} = X_i^G$
17:             $count(i)=count(i)+3$ // save the number of consecutive unsuccessful update
18:         else                  // successful update
19:             $X_i^{G+1} = U$, $X_i^G \rightarrow A$, $count(i)=0$
20:         end if
21:     end for
22: end while
```

Fig. 5. The pseudo-code of MCoDE.

```
Algorithm 4: Archive Update
01: if $|A| > NP$
02:     $L = |A| - NP$
03:     Remove $L$ individuals from $A$ randomly.
04: end
```

Fig. 6. The pseudo-code of archive update.

```
Algorithm 5: The procedure of the Initialization
01: Generate a uniformly distributed random population $P$
04: Set $FES$=$NP$
05: Set $F_m = 0.5$, $CR_m = 0.5$, $p = 0.05$, $m$=30, $c$=0.1
06: Set [$F_m = 1.0$, $CR_m = 0.1$], [$F_m = 0.5$, $CR_m = 0.9$], [$F_m = 0.8$, $CR_m = 0.2$]
07: Set $A = \varnothing$, $count(1{:}NP)$=0,$flag$=2, $Q$=0, $G$=1, max_$FES$=10000$D$
```

Fig. 7. The pseudo-code of initialization process.

where f_{best}^{G-1} is the best fitness value at generation G-1, and f_{best}^{G} is the best fitness value at generation G. Note that the fitness value must be a positive number here.

Definition 2. **The unsuccessful generation** indicates the fitness improvement of current generation is smaller than a pre-defined threshold (*i.e.*,ε in this paper). For example, at generation G, if $IR \leq \varepsilon$, this generation G is called an **unsuccessful generation**.

In our hybrid framework as illustrated in Fig. 2, Q records the number of unsuccessful generation and is initialized to be 0 at first use. When the value of Q is larger than the pre-defined threshold, the switch on MJADE and MCoDE is activated. For example, If MJADE is run at present, when Q is larger than $Q1$ at first time, MJADE will be stopped and MCoDE will be run instead with Q reset to 0. Similarly, if MCoDE is performed at current generation, when Q is larger than $Q2$, MJADE will be turned to run and then Q is also reset to 0. The parameters ε, $Q1$, $Q2$ are all the user pre-defined parameters. As they may have significant impact on the performance of HMJCDE, their sensitivity analysis will be conducted in Section 5.

4.3.2. Archive update

In HMJCDE, an external archive ***A*** is used for MCoDE and MJADE. When running MCoDE or MJADE, if the target vector (parent) is beaten by the associated trial vector as shown in line 16 of **Algorithm 2** or line 19 of **Algorithm 3**, it will be preserved in the external archive ***A***. Generally, the larger external archive can provide the better population diversity. However, too large external archive may reduce the rate of convergence speed. In this paper, the maximal size of ***A*** s also set to *NP* (the population size). With the execution of HMJCDE, if the size of ***A*** exceeds *NP*, some redundant individuals are randomly removed out from ***A*** in order to keep the archive size as *NP*. The pseudo-code of archive update is shown in Fig. 6.

4.4. The complete HMJCDE algorithm

The above subsections have introduced MJADE, MCoDE, archive update and the hybridized approach using Q in details. The initialization process of HMJCDE is shown in Fig. 7, and the pseudo-code of the complete HMJCDE algorithm is demonstrated in Fig. 8.

In line 7 of **Algorithm 5**, the parameter *flag* is initialized to 2, which means that MCoDE will be executed at first. The reason is that MCoDE pays more attention to exploration than MJADE. With the execution of MCoDE (as described in **Algorithm 3**), the number of the unsuccessful generation is recorded in Q according to the cases that the improvement rate is lower than a certain threshold ε at each generation (line 17–22 of **Algorithm 6**). When the number of the unsuccessful generation is larger than a certain threshold $Q2$, *flag* is set to 1 and Q is reset to 0 (line 19–21 of **Algorithm 6**). After that, MJADE (as described in **Algorithm 2**) is operated in HMJCDE. With the running of MJADE, the number of the unsuccessful generation is also recorded by Q (line 7–12 of **Algorithm 6**). When the number of the unsuccessful generation is larger than a certain threshold $Q1$, *flag* is set to 2 and Q is reset to 0 (line 9–11 of **Algorithm 6**). Instead, MCoDE will be executed in the next generation. At the end of each generation, the archive ***A*** will be updated (as described in **Algorithm** 4). This whole process will be repeated until the termination condition is satisfied. The individual with the smallest objective value in final population is reported as the final optimization result.

```
Algorithm 6:The procedure of HMJCDE
01: Initialization (Algorithm 5)
02: while  $FES \le \max\_FES$
03:     if  $flag==1$
04:         Execute MJADE (Algorithm 2)
05          $FES=FES+NP$
06:         $IR = (f_{best}^{G-1} - f_{best}^{G}) / f_{best}^{G-1}$
07:         if  $IR \le \varepsilon$
08:             $Q=Q+1$
09:             if  $Q>Q1$
10:                 $flag=2, Q=0$
11:             end if
12:         end if
13:     else
14:         Execute MCoDE (Algorithm 3)
15:         $FES=FES+3NP$
16:         $IR = (f_{best}^{G-1} - f_{best}^{G}) / f_{best}^{G-1}$
17:         if  $IR \le \varepsilon$
18:             $Q=Q+1$
19:             if  $Q > Q2$
20:                 $flag=1, Q=0$
21:             end if
22:         end if
23:     end if
24:     Archive update (Algorithm 4)
25:     $G=G+1$;
26: end while
Output: The individual with the smallest objective value in the population
```

Fig. 8. The pseudo-code of the complete HMJCDE algorithm.

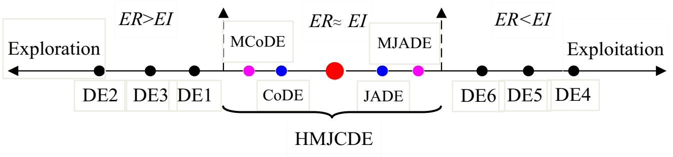

Fig. 9. ***TEE*** of DE mutation strategies and related algorithms.

Table 1
The parameter settings of all the DE variants.

Algorithms	Parameter settings
CoDE	$NP = 30$
MCoDE	$NP = 30$
JADE	$NP = 100$, $p = 0.05$, $c = 0.1$
MJADE	$NP = 100$, $p = 0.05$, $c = 0.1$, $m = 30$
ODE	$NP = 100$, $F = 0.5$, $CR = 0.9$, $J_r = 0.3$
SaDE	$NP = 100$, $k = 4$, $\varepsilon = 0.01$, $L = 50$
sinDE	$NP = 40$, $freq = 0.25$
rank-jDE	$NP = 100$ $\tau_1 = \tau_2 = 0.1$
AEPD-JADE	$NP = 20$, $c = 0.1$, $p = 0.2$, $T = 0.001$, $a = 0.0005$,
AuDE	$NP = 100$, $\tau_j = 0.1$ $(j = 1, 2, 3, 4, 5)$, $F_{\min} = CR_{\min} = 0, F_{\max} = CR_{\max} = 1$
DE-VNS	$NP = 100$, $F = [0.4, 0.6, 0.8, 1]$, $par_{\min} = 0$, $par_{\max} = 0.7$, $n_0 = 2, \delta = 0.05$
HMJCDE	$NP = 100$, $p = 0.05$, $m = 30$, $\varepsilon = 0.05$, $Q1 = 10$, $Q2 = 5$, $c = 0.1$

4.5. More discussions of exploration and exploitation

Exploration and exploitation are two basic search capabilities for problem solving. Exploration is the process of visiting entirely new regions of a search space, while exploitation is the process of visiting those regions of a search space within the neighborhood of previously visited points [59]. Different DE mutation strategies possess different characteristics, and each of them shows distinct tradeoff between exploration and exploitation (***TEE***). As claimed in [15,18,52], "DE/rand/1" (DE1) and "DE/rand/2" (DE2) are absolutely random in nature. Thus, DE1 and DE2 have a stronger exploration capability, but may lead to a slower convergence speed. "DE/current-to-rand/1" (DE3) is rotation-invariant strategy, the base vector is constructed in a random manner with the current vector, and the difference vector is also built in a random manner. DE3 also shows a stronger exploration capability than exploitation. On the other hand, other DE mutation strategies, *i.e.*, "DE/best/1" (DE4), "DE/current-to-best/1" (DE5) and "DE/rand-to-best/1" (DE6), exploit the information of the current best solution. Their base vectors are constructed relying on the best solution, and their difference vectors are obtained in a random manner. Hence, DE4, DE5 and DE6 demonstrate a stronger exploitation capability, but they are more likely to lower their population diversity and lead to premature convergence. Moreover, compared with DE4 and DE5, DE6 shows a slightly stronger exploration capability due to the introduction of more perturbations using random vectors. Therefore, among these six DE strategies, DE1, DE2, DE3 pay more attention to exploration, while DE4, DE5 and DE6 focus on exploitation.

The state-of-the-art DE variants, such as CoDE and JADE, are designed based on the above DE mutation strategies. CoDE can obtain a better tradeoff between exploitation and exploration by combing three DE mutation strategies (*i.e.*, DE1, DE2 and DE3). Compared to the single use of DE1, DE2 and DE3, CoDE is designed to have a stronger exploitation capability. On the other hand, JADE designs a new mutation strategy "DE/current-to-*p*best/1" (DE7). DE7 will degenerate to DE3 when p is set to 1, while DE7 will turn to DE5 when $p = 1/NP$. Therefore, DE7 obtains a better tradeoff than DE3 and DE5. Moreover, the parameter p can adaptively adjust the exploration capability and exploration capability of DE7. For example, when p is set to a small value (*e.g.*,$p \in (0, 0.2)$), DE7 pays more attention to exploitation. When considering the modifications on CoDE and JADE, it is noted that MCoDE additionally introduces an external archive ***A*** to enhance the population diversity. Thus, MCoDE has a slightly stronger exploration capability than CoDE. MJADE exploits the information of the current best solution in DE7, and resultantly it shows a slightly enhanced exploitation capability than JADE. Although there are a number of metrics about exploration and exploitation in literature [60], for the sake of simplicity, we follow the approach designed in [52] to show the tradeoff between exploration and exploitation, three cases are defined.

- Case 1 $ER \approx EI$: The mutation strategy or the algorithm has the exploration capability that is close to the exploitation capability.
- Case 2 $ER > EI$: The mutation strategy or the algorithm has the exploration capability that is stronger than the exploitation capability.
- Case 3 $ER < EI$: The mutation strategy or the algorithm has the exploitation capability that is stronger than the exploration capability.

Intuitively, the ***TEE*** of each DE mutation strategy and related algorithms are illustrated in Fig. 9. Our modified MCoDE and MJADE respectively enhance the exploration capability of CoDE and the exploitation ability of JADE slightly. The main purpose of this paper is to design an algorithm (*i.e.*, HMJCDE), which can adaptively exploit two competitive algorithms (*i.e.*, MCoDE and MJADE) with good ***TEE*** to look for a better balance.

5. Experimental results

5.1. Benchmark functions and experimental setup

In our experiments, HMJCDE will be tested by 30 benchmark functions proposed in the special session on real-parameter optimization of CEC2014 [16]. There are many properties of these

Table 2
Results of CoDE, MCoDE, JADE, MJADE and HMJCDE on 30 test functions with 30*D*.

Alg	CoDE	MCoDE	JADE	MJADE	HMJCDE
Fuc	Mean error (std dev)	Mean error (std dev)	Mean error (std dev)	Mean error (std dev)	Mean error (std dev)
F1	9.64e+03(1.06e+04)	1.83e+03(3.92e+03)	7.55e+02(1.76e+03)	**6.75e+02(1.32e+03)**	1.51e+03(2.53e+03)
F2	**0.00e+00(0.00e+00)**	1.43e−13(1.72e−13)	1.90e−14(1.32e−14)	4.07e−14(1.61e−14)	**0.00e+00(0.00e+00)**
F3	**0.00e+00(0.00e+00)**	**0.00e+00(0.00e+00)**	5.41e−04(2.90e−03)	2.93e+00(4.31e+00)	**0.00e+00(0.00e+00)**
F4	2.69e−02(5.03e−02)	8.92e−02(1.28e−01)	6.82e−14(2.75e−14)	1.65e−13(9.12e−14)	**3.03e−14(4.89e−14)**
F5	2.06e+01(4.92e−02)	2.05e+01(4.03e−02)	2.03e+01(3.59e−02)	**2.00e+01(1.17e−02)**	**2.00e+01(2.66e−02)**
F6	1.51e+01(8.03e+00)	2.46e+00(4.55e+00)	9.98e+00(2.45e+00)	6.57e+00(3.28e+00)	**2.38e+00(2.92e+00)**
F7	**0.00e+00(0.00e+00)**	**0.00e+00(0.00e+00)**	1.14e−14 (3.47e−14)	2.47e−04(1.35e−03)	**0.00e+00(0.00e+00)**
F8	1.85e+01(1.88e+00)	1.78e+00(1.32e+00)	**0.00e+00(0.00e+00)**	1.40e−13(4.89e−14)	**0.00e+00(0.00e+00)**
F9	1.27e+02(1.21e+01)	9.57e+01(9.31e+00)	4.59e+01(3.43e+00)	**3.13e+01(6.16e+00)**	3.92e+01(1.65e+00)
F10	8.07e+02(8.87e+01)	2.00e+02(2.54e+01)	**6.94e−03(9.98e−03)**	2.01e−02(2.59e−02)	1.51e−01(2.85e−02)
F11	4.84e+03(1.94e+02)	4.18e+03(1.83e+02)	2.64e+03(2.21e+02)	**2.02e+03(3.57e+02)**	2.28e+03(4.43e+02)
F12	1.00e+00(1.40e−01)	7.76e−01(1.13e−01)	3.64e−01(3.84e−02)	**2.71e−01(7.57e−02)**	3.23e−01(9.02e−02)
F13	3.88e−01(4.15e−02)	3.26e−01(3.88e−02)	3.11e−01(3.61e−02)	**2.28e−01(4.01e−02)**	2.46e−01(4.22e−02)
F14	2.76e−01(1.05e+02)	2.60e−01(2.60e−02)	2.41e−01(5.39e−02)	2.23e−01(3.30e−02)	**2.10e−01(3.71e−02)**
F15	1.23e+01 (8.76e−01)	9.37e+00(8.16e−01)	4.18e+00(4.17e−01)	**3.86e+00(8.34e−01)**	3.97e+00(1.02e+00)
F16	1.16e+01(2.47e−01)	1.11e+01(3.10e−01)	**9.32e+00(4.07e−01)**	9.71e+00(7.38e−01)	9.93e+00(5.27e−01)
F17	7.62e+02(2.00e+02)	3.77e+02(2.33e+02)	1.23e+03(3.90e+02)	3.52e+04(1.86e+05)	**3.63e+02(2.37e+02)**
F18	2.96e+01(6.11e+00)	1.09e+01(4.59e+00)	7.79e+01(3.75e+01)	5.07e+02(1.28e+03)	**1.40e+01(6.79e+00)**
F19	4.53e+00(2.88e−01)	4.51e+00(3.84e−01)	4.42e+00(7.72e−01)	4.56e+00(6.54e−01)	**3.84e+00(7.75e−01)**
F20	2.06e+01(3.07e+00)	1.13e+01(3.09e+00)	2.86e+03(2.34e+03)	3.23e+03(2.74e+03)	**1.01e+01(3.15e+00)**
F21	4.38e+02(1.05e+02)	2.26e+02(1.23e+02)	1.58e+04(6.61e+04)	1.93e+04(4.65e+04)	**1.88e+02(1.31e+02)**
F22	9.79e+01(4.89e+01)	5.00e+01(4.55e+01)	1.64e+02(8.03e+01)	1.84e+02(1.02e+02)	**8.25e+01(7.95e+01)**
F23	**3.15e+02(1.11e−13)**	**3.15e+02(1.11e−13)**	**3.15e+02(5.78e−14)**	**3.15e+02(2.42e−13)**	**3.15e+02(2.14e−13)**
F24	**2.14e+02(1.11e+01)**	2.22e+02(4.29e+00)	2.26e+02(3.26e+00)	2.25e+02(1.30e+00)	2.24e+02(2.34e+00)
F25	**2.00e+02(4.83e−02)**	2.02e+02(1.37e−01)	2.04e+02(1.30e+00)	2.04e+02(1.75e+00)	2.03e+02(4.95e−01)
F26	**1.00e+02(4.96e−02)**	**1.00e+02(3.88e−02)**	**1.00e+02(3.46e−02)**	**1.00e+02(4.31e−02)**	**1.00e+02(3.74e−02)**
F27	3.97e+02(1.83e+01)	4.00e+02(2.60e−01)	3.50e+02(5.05e+01)	**3.27e+02(4.69e+01)**	3.90e+02(3.06e+01)
F28	8.78e+02(4.62e+01)	7.91e+02(2.05e+01)	**7.85e+02(4.68e+01)**	7.99e+02(4.39e+01)	8.32e+02(3.66e+01)
F29	**4.85e+02(2.88e+02)**	7.16e+02(2.34e+00)	7.29e+02(1.25e+01)	7.88e+02(1.73e+02)	7.22e+02(2.59e+00)
F30	8.78e+02(1.08e+02)	**5.66e+02(1.19e+02)**	1.53e+03(5.09e+02)	1.72e+03(6.27e+02)	8.75e+02(3.46e+02)

test instances, such as unimodality, multimodality, non-separation, separation, rotation, symmetrisation, non-symmetrisation and quadratic ill-condition. All the test functions are treated as black-box problems, and their search ranges are $[-100, 100]^D$ (D is the number of decision variables). The detailed description of these benchmark functions can be found in [16]. In brief, these 30 benchmark functions can be divided into four categories: F1-F3 are unimodal functions, F4-F16 are simple multimodal functions, F17-F22 are hybrid functions, and F23-F30 are composition functions. To evaluate the performance of HMJCDE, it is compared to nine DE variants and five non-DE heuristic algorithms in this section. The parameter settings of all the compared algorithms are listed in Table 1, which are expected to solve all types of GOPs as suggested by their original authors. By this way, it is guaranteed to have a fair comparison among these compared algorithms [61].

In the following experimental results, the average and the standard deviation of the function error value $f(X_{best}) - f(X^*)$ are adopted to express the optimization performance, where X_{best} is the best solution found by the algorithm in each run and X^* is the true global optimal solution of the test function. The maximal function evaluation (max_*FES*) is used as the termination condition, which is set to 10000*D* according to the guidelines provided in the special session of CEC2014 [16] (*D* is the dimension of the feasible solution space). For all the experiments, 30 independent runs are conducted for each test function. The Wilcoxon's rank-sum test, which is a nonparametric statistic test for independent samples, is conducted on the experimental results with the 5% significant level to obtain a reliable statistic conclusion. It is noted that the best result for each test problem is marked in **boldface**. All the experiments are implemented in MATLAB2010a and run on a computer with 4-GB RAM Memory, 3.20-GHz CPU and Windows 7 system.

5.2. Performance enhancement by MCoDE, MJADE and HMJCDE

In order to show the performance enhancement brought by MCoDE, MJADE and HMJCDE, the comparison among CoDE, MCoDE, JADE, MJADE, and HMJCDE is performed on all the test functions with 30*D* (*D* denotes the number of decision variables). The parameter settings of these algorithms have been given in Table 1. Table 2 gives the experimental results and Table 3 presents the statistical comparison summary.

As shown in Table 2, considering ***unimodal functions F1-F3***, MCoDE is better than CoDE on F1, and they obtain similar results on F2 and F3. This observation validates that MCoDE is better than or similar to CoDE on these unimodal functions. For MJADE, it is better than, similar to and worse than JADE on F1, F2 and F3 respectively. These results show that MJADE performs comparably with JADE on these unimodal functions. In addition, HMJCDE is better than MCoDE (or CoDE) on F1, and similar to MCoDE (or CoDE) on F2 and F3; HMJCDE is better than, similar to and worse than MJADE (or JADE) on F3, F2, and F1 respectively. Obviously, HMJCDE surpasses MCoDE (or CoDE) on F1 by exploiting the merit of MJADE, and exceeds MJADE (or JADE) on F3 by taking advantage of MCoDE. Considering these results, we may draw a conclusion that HMJCDE can combine the merits of MJADE and MCoDE on these unimodal functions. Regarding ***simple multimodal functions F4-F16***, MCoDE outperforms CoDE on 9 test functions (*i.e.*, F6, F8-F15) out of 13 test functions and is similar to CoDE on 4 test functions (F4, F5, F7, F16). These experimental results demonstrate that MCoDE is better than CoDE on most cases, and similar to CoDE on a small number of cases when considering these 13 simple multimodal functions. Meanwhile, MJADE is better than JADE on 8 test functions (F5, F6, F9, F11-F15) out of 13 test functions, similar to JADE on 3 test functions (F4, F8, F16) and worse than JADE on 2 test functions (F7, F10). Similarly, MJADE can improve the performance of JADE on most cases, but may cause premature convergence on

Table 3
Comparison of CoDE and MCoDE, MCoDE and HMJCDE, JADE and MJADE, MJADE and HMJCDE.

Compare	Func	CoDE vs MCoDE	MCoDE vs HMJCDE	JADE vs MJADE	MJADE vs HMJCDE
+/=/–	unimoal	0/2/1	0/2/1	1/1/1	1/1/1
	multimodal	0/4/9	0/2/11	2/3/8	5/5/3
	hybrid	0/1/5	0/0/6	6/0/0	0/0/6
	composition	2/4/2	2/6/0	2/5/1	2/4/2
	total	2/11/17	2/10/18	11/9/10	8/10/12

"–", "+", and "=" respectively denote that the performance of the corresponding algorithm is worse than, better than, and similar to that of its competitor according to the Wilcoxon's rank test at a 0.05 significance level.

some cases (*e.g.*, F7 and F10) when considering these simple multimodal functions. When compared with CoDE and MCoDE, HMJCDE outperforms CoDE on all the test functions except F7, and surpasses MCoDE on all the test functions excluding F6 and F7. When compared with JADE and MJADE, HMJCDE is better than JADE on 8 test functions (F5, F6, F9, F11-F15), similar to JADE on 4 test functions (F4, F7, F8, F16) and worse than JADE on F10; HMJCDE is better than MJADE on 3 test functions (F6, F7, F13), similar to MJADE on 5 test functions (F4, F5, F8, F15, F16), and worse than MJADE on 5 test functions (F9-F13). Although HMJCDE is slightly beaten by MJADE on 5 test functions (F9-F13), HMJCDE can outperform CoDE, MCoDE and JADE on most cases. An interesting phenomenon should be pointed out that JADE (or MJADE) and CoDE (or MCoDE) are unable to obtain the global optimal solutions of F3 and F8 simultaneously, but HMJCDE can always find the global solution of F3 and F8 simultaneously. This kind of phenomenon also can validate that HMJCDE can combine the advantage of CoDE and JADE to some extent. Considering ***hybrid functions F17-F22***, MCoDE is better than CoDE on 5 test functions (F17, F18, F20-F21) and similar to CoDE on F19. These results indicate that MCoDE is better than CoDE on most of these hybrid functions. However, MJADE is worse than JADE on all these hybrid functions. This is because the mutation strategy of MJADE pays more attention to exploitation, which may easily cause premature convergence when solving this kind of hybrid functions characterized with many local optimal solutions. Thus, MJADE is worse than JADE on these hybrid functions. Interestingly, HMJCDE can outperform CoDE, MCoDE, JADE and MJADE on all these hybrid functions. Regarding ***composition functions F23-F30***, these functions are very complicated as each of them is composed by many sub-functions with a huge number of local optima. To the best of our knowledge, there is no DE variant that is able to find a near-global optimal solution for any of these test functions, and many DE variants show similar performance on most cases. MCoDE is better than CoDE on 2 test functions (F28, F30), similar to CoDE on 4 test functions (F23, F25-F27), and worse than CoDE on 2 test functions (F24, F29). Therefore, MCoDE is competitive to CoDE on these composition functions. Meanwhile, MJADE is better than JADE on F27, similar to JADE on 5 test functions (F23-26, F28), and worse than JADE on 2 test functions (F29, F30). In addition, HMJCDE is worse than MCoDE on 2 test functions (F28, F30), and similar to MCoDE on 6 test functions (F23-F27, F29); HMJCDE is better than MJADE on 2 test functions (F29, F30), similar to MJADE on 4 test functions (F23-26), and worse than MJADE on 2 test functions (F27, F28). Therefore, HMJCDE performs slightly worse than CoDE and MCoDE, and similarly with JADE and MJADE on these composition functions.

To summarize the above comparisons, all the statistical comparison results are collected in Table 3. MCoDE is better than CoDE on unimodal functions, simple multimodal functions and hybrid functions, and similar to CoDE on composition functions. Similarly, HMJCDE is better than MCoDE on unimodal functions, simple multimodal functions and hybrid functions, and slightly worse than MCoDE on composition functions. MJADE is better than JADE on simple multimodal functions, similar to JADE on unimodal functions and composition functions, and worse than JADE on hybrid functions. However, HMJCDE is only better than MJADE on hybrid functions, and similar to MJADE on unimodal, simple multimodal and composition functions. Therefore, when solving very complicated test functions (*e.g.*, composition functions F23-F30), HMJCDE may not be able to effectively combine the merits of CoDE and JADE. However, when considering other kinds of test functions, HMJCDE is better than CoDE (MCoDE) and JADE (MJADE).

Moreover, in order to clearly show the effect of MCoDE and MJADE in our hybrid framework, the following four combinations are used for comparison. The experimental results are given in Table 4.

(1) CoDE hybridizes with JADE, denoted by HJCDE.
(2) CoDE hybridizes with MJADE, denoted by HJMCDE.
(3) MCoDE hybridizes with JADE, denoted by HJCMDE.
(4) MCoDE hybridizes with MJADE, denoted by HMJCDE.

As shown in Table 4, HMJCDE is better than or at least similar to HJCDE, HJMCDE and HJCMDE on unimodal functions. Moreover, HMJCDE is significantly better than HJCDE, HJMCDE on simple multimodal functions and hybrid functions, and significantly better than and similar to HJCMDE on simple multimodal functions and hybrid functions respectively. On composition functions, all the four combinations obtain the similar results. In more detail, HMJCDE is better than HJCDE, HJMCDE and HJCMDE on 17, 16 and 12 out of 30 test functions. On the contrary, HMJCDE is only beaten by HJCDE, HJMCDE and HJCMDE on 2, 1 and 5 test functions, respectively. Therefore, it is reasonable to conclude that the improvement of HMJCDE comes from both of MJADE and MCoDE.

5.3. Comparison with state-of-the-art DE variants

In this subsection, HMJCDE is further compared to four classical state-of-the-art DE variants, *i.e.*, CoDE [15], JADE [14], ODE [17], SaDE [18], and five recently proposed DE variants, *i.e.*, sinDE [19], AuDE [20], DE-VNS [21], AEPD-JADE [22] and rank-jDE [23] on all the test functions with 30*D*, 50*D* and 100*D*. The parameters settings of theses DE-related algorithms are accordingly set as recommended by their authors, which are respectively given in Table 1.

5.3.1. Discussions on test functions with 30D

The experimental results for all the test problems with 30*D* are listed in Tables 5 and 6, where the symbols of "–", "+", "=" in last row of each table respectively denote that the performance of the corresponding algorithm is worse than, better than and similar to that of HMJCDE. From Table 5, it is found that HMJCDE has the best performance among the four classical state-of-the-art DE variants on all the 30 test functions with 30*D*. Regarding the unimodal functions F1-F3, all the algorithms cannot find the global optimal solution of F1; JADE obtains the best result while HMJCDE achieves the second best result for F1; For F2-F3, only HMJCDE, CoDE and ODE can find the global optimal solution or near global optimal solution in each run. Obviously, HMJCDE inherits the merits of CoDE and JADE on these unimodal functions with 30*D*. Concerning the simple multi-

Table 4
Results of HJCDE, HJMCDE, HJCMDE and HMJCDE on 30 test functions with 30*D*.

Alg Fuc	HJCDE Mean error (std dev)	HJMCDE Mean error (std dev)	HJCMDE Mean error (std dev)	HMJCDE Mean error (std dev)
F1	**1.39e+03(2.44e+03)+**	2.93e+03(3.17e+03)−	1.99e+03(3.76e+03)−	1.51e+03(2.53e+03)
F2	**0.00e+00(0.00e+00)=**	**0.00e+00(0.00e+00)=**	**0.00e+00(0.00e+00)=**	**0.00e+00(0.00e+00)**
F3	1.68e−08(2.46e−08)−	4.25e−08(3.39e−08)−	**0.00e+00(0.00e+00)=**	**0.00e+00(0.00e+00)**
F4	5.53e−09(3.03e−08)−	7.01e−14(1.03e−13)=	3.46e−14(5.52e−14)=	**3.03e−14(4.89e−14)**
F5	2.04e+01(2.90e−02)−	2.01e+01(2.42e−02)=	2.04e+01(5.63e−02)−	**2.00e+01(2.66e−02)**
F6	1.37e+01(1.66e+00)−	4.95e+00(3.14e+00)−	4.40e+00(5.03e+00)−	**2.38e+00(2.92e+00)**
F7	**0.00e+00(0.00e+00)=**	3.07e−13(1.68e−12)=	**0.00e+00(0.00e+00)=**	**0.00e+00(0.00e+00)**
F8	1.48e−11(9.92e−12)−	3.32e−02(1.82e−01)−	**0.00e+00(0.00e+00)=**	**0.00e+00(0.00e+00)**
F9	5.07e+01(6.51e+00)−	5.12e+01(1.32e+01)−	4.03e+01(5.73e+00)−	**3.92e+01(1.65e+00)**
F10	2.98e+00(1.05e+00)−	3.67e−01(7.81e−01)−	1.05e+00(2.06e−01)−	**1.51e−01(2.85e−02)**
F11	2.90e+03(2.41e+02)−	2.91e+03(5.70e+02)−	2.72e+03(2.03e+02)−	**2.28e+03(4.43e+02)**
F12	5.55e−01(5.28e−02)−	3.48e−01(1.05e−01)−	5.13e−01(6.45e−02)−	**3.23e−01(9.02e−02)**
F13	2.44e−01(3.92e−02)=	2.48e−01(3.77e−02)=	**2.13e−01(3.17e−02)+**	2.46e−01(4.22e−02)
F14	2.30e−01(2.35e−02)−	2.24e−01(3.33e−02)−	2.27e−01(2.07e−02)−	**2.10e−01(3.71e−02)**
F15	5.33e+00(4.81e−01)−	4.32e+00(1.59e+00)−	4.81e+00(5.29e−01)−	**3.97e+00(1.02e+00)**
F16	1.03e+01(3.42e−01)=	1.03e+01(4.58e−01)=	1.02e+01(3.25e−01)=	**9.93e+00(5.27e−01)**
F17	3.94e+02(2.22e+02)−	9.25e+02(4.54e+02)−	4.48e+02(2.16e+02)−	**3.63e+02(2.37e+02)**
F18	1.83e+01(6.51e+00)−	2.64e+01(1.20e+01)−	1.60e+01(7.87e+00)−	**1.40e+01(6.79e+00)**
F19	4.63e+00(9.26e−01)−	**3.48e+00(8.28e−01)+**	4.14e+00(6.17e−01)−	3.84e+00(7.75e−01)
F20	2.14e+01(8.08e+00)−	1.94e+01(7.18e+00)−	**9.22e+00(3.38e+00)+**	1.01e+01(3.15e+00)
F21	2.51e+02(2.44e+02)−	4.84e+02(2.13e+02)−	**1.62e+02(1.04e+02)+**	1.88e+02(1.31e+02)
F22	1.46e+02(6.61e+01)−	1.38e+02(8.21e+01)−	**6.25e+01(5.00e+01)+**	8.25e+01(7.95e+01)
F23	3.15e+02(5.78e−14)=	3.15e+02(5.78e−14)=	3.15e+02(2.27e−13)=	**3.15e+02(2.14e−13)**
F24	2.24e+02(1.16e+00)=	2.24e+02(7.76e−01)=	**2.24e+02(6.77e−01)=**	2.24e+02(2.34e+00)
F25	2.03e+02(3.73e−01)=	2.03e+02(4.03e−01)=	2.03e+02(4.70e−01)=	**2.03e+02(1.95e−01)**
F26	1.00e+02(2.72e−02)=	1.00e+02(4.66e−02)=	1.00e+02(3.14e−02)=	**1.00e+02(3.74e−02)**
F27	3.90e+02(3.06e+01)=	3.88e+02(3.21e+01)=	**3.84e+02(3.81e+01)=**	3.90e+02(3.06e+01)
F28	8.08e+02(2.56e+01)+	8.42e+02(3.67e+01)=	**7.87e+02(1.76e+01)+**	8.32e+02(3.66e+01)
F29	7.20e+02(4.14e+00)=	**7.19e+02(3.98e+00)=**	7.19e+02(5.86e+00)=	7.22e+02(2.59e+00)
F30	8.46e+02(3.49e+02)=	1.17e+03(5.18e+02)−	**8.37e+02(2.86e+02)=**	8.75e+02(3.46e+02)
+/=/−	**2/11/17**	**1/13/16**	**5/13/12**	

"−", "+", and "=" respectively denote that the performance of the corresponding algorithm is worse than, better than, and similar to that of its competitor according to the Wilcoxon's rank test at a 0.05 significance level.

Table 5
Comparisons of HMJCDE and four classical DE variants on 30 test functions with 30*D*.

Alg Fuc	CoDE Mean error(std dev)	JADE Mean error (std dev)	ODE Mean error (std dev)	SaDE Mean error (std dev)	HMJCDE Mean error (std dev)
F1	9.64e+03(1.06e+04)−	**7.55e+02(1.76e+03)+**	1.11e+05(7.64e+04)−	3.70e+05(2.31e+05)−	1.51e+03(2.53e+03)
F2	**0.00e+00(0.00e+00)=**	1.90e−14(1.32e−14)=	7.80e−13(8.01e−13)−	2.65e−14(8.31e−14)−	**0.00e+00(0.00e+00)**
F3	**0.00e+00(0.00e+00)=**	5.41e−04(2.90e−03)−	1.14e−14(2.31e−14)−	9.25e+00(2.52e+01)−	**0.00e+00(0.00e+00)**
F4	2.69e−02(5.03e−02)−	6.82e−14(2.75e−14)=	3.50e−01(2.07e−01)−	3.39e+01(3.47e+01)−	**3.03e−14(4.89e−14)**
F5	2.06e+01(4.92e−02)−	2.03e+01(3.59e−02)−	2.09e+01(4.75e−02)−	2.05e+01(6.15e−02)−	**2.00e+01(2.66e−02)**
F6	1.51e+01(8.03e+00)−	9.98e+00(2.45e+00)−	**6.00e−02(1.73e−01)+**	4.79e+00(1.45e+00)−	2.38e+00(2.92e+00)
F7	**0.00e+00(0.00e+00)=**	1.14e−14 (3.47e−14)=	3.79e−15(2.08e−14)=	9.84e−03(1.39e−02)−	**0.00e+00(0.00e+00)**
F8	1.85e+01(1.88e+00)−	**0.00e+00(0.00e+00)=**	1.53e+02(2.08e+01)−	3.32e−02(1.82e−01)=	**0.00e+00(0.00e+00)**
F9	1.27e+02 (1.21e+01)−	4.59e+01(3.43e+00)−	1.83e+02(1.25e+01)−	3.50e+01(7.79e+00)=	3.92e+01(1.65e+00)
F10	8.07e+02(8.87e+01)−	**6.94e−03(9.98e−03)+**	6.00e+03(5.01e+02)−	2.33e−01(4.76e−01)=	1.51e−01(2.85e−02)
F11	4.84e+03(1.94e+02)−	2.64e+03(2.21e+02)−	6.97e+03(2.12e+02)−	3.25e+03(5.19e+02)−	2.28e+03(4.43e+02)
F12	1.00e+00(1.40e−01)−	3.64e−01(3.84e−02)−	2.56e+00(2.16e−01)−	7.94e−01(1.04e−01)−	**3.23e−01(9.02e−02)**
F13	3.88e−01(4.15e−02)−	3.11e−01(3.61e−02)−	3.74e−01(4.99e−02)−	2.54e−01(4.18e−02)=	2.46e−01(4.22e−02)
F14	2.76e−01(1.05e−02)−	2.41e−01(5.39e−02)−	2.67e−01(3.51e−02)−	2.43e−01(2.63e−02)−	**2.10e−01(3.71e−02)**
F15	1.23e+01 (8.76e−01)−	4.18e+00(4.17e−01)−	1.55e+01(8.70e−01)−	4.27e+00(1.40e+00)=	**3.97e+00(1.02e+00)**
F16	1.16e+01(2.47e−01)−	**9.32e+00(4.07e−01)=**	1.26e+01(2.12e−01)−	1.10e+01(2.92e+01)−	9.93e+00(5.27e−01)
F17	7.62e+02(2.00e+02)−	1.23e+03(3.90e+02)−	1.51e+03(1.95e+02)−	1.22e+04(9.91e+03)−	**3.63e+02(2.37e+02)**
F18	2.96e+01(6.11e+00)−	7.79e+01(3.75e+01)−	5.72e+01(6.26e+00)−	3.60e+02(3.88e+02)−	**1.40e+01(6.79e+00)**
F19	4.53e+00(2.88e−01)−	4.42e+00(7.72e−01)−	4.74e+00(2.82e−01)−	4.02e+00(8.55e−01)=	**3.84e+00(7.75e−01)**
F20	2.06e+01(3.07e+00)−	2.86e+03(2.34e+03)−	3.83e+01(5.29e+00)−	1.21e+02(1.25e+02)−	**1.01e+01(3.15e+00)**
F21	4.38e+02(1.05e+02)−	1.58e+04(6.61e+04)−	8.19e+02(1.09e+02)−	3.52e+03(3.88e+03)−	**1.88e+02(1.31e+02)**
F22	9.79e+01(4.89e+01)−	1.64e+02(8.03e+01)−	8.42e+01(8.50e+01)=	1.42e+02(5.30e+01)−	**8.25e+01(7.95e+01)**
F23	**3.15e+02(1.11e−13)=**	**3.15e+02(5.78e−14)=**	**3.15e+02(5.78e−14)=**	**3.15e+02(2.15e−13)=**	**3.15e+02(2.14e−13)**
F24	**2.14e+02(1.11e+01)+**	2.26e+02(3.26e+00)−	2.16e+02(7.66e+00)+	2.28e+02(4.56e+00)−	2.24e+02(2.34e+00)
F25	**2.00e+02(4.83e−02)+**	2.04e+02(1.30e+00)=	2.03e+02(2.67e−01)=	2.05e+02(5.77e+00)=	2.03e+02(4.95e−01)
F26	**1.00e+02(4.96e−02)=**	**1.00e+02 (3.46e−02)=**	**1.00e+02(4.80e−02)=**	1.07e+02(2.53e+01)−	**1.00e+02(3.74e−02)**
F27	3.97e+02(1.83e+01)=	**3.50e+02(5.05e+01)+**	3.67e+02(4.83e+01)+	4.22e+02(3.22e+01)−	3.90e+02(3.06e+01)
F28	8.78e+02(4.62e+01)−	7.85e+02(4.68e+01)+	**7.63e+02(1.04e+02)+**	8.92e+02(3.05e+01)−	8.32e+02(3.66e+01)
F29	4.85e+02(2.88e+02)+	7.29e+02(1.25e+01)−	**3.43e+02(2.32e+02)+**	1.11e+03(2.00e+02)−	7.22e+02(2.59e+00)
F30	8.78e+02(1.08e+02)=	1.53e+03(5.09e+02)−	**7.66e+02(1.69e+02)=**	1.07e+03(2.90e+02)−	8.75e+02(3.46e+02)
+/=/−	**3/7/20**	**4/8/18**	**5/6/19**	**0/8/22**	

"−", "+", and "=" respectively denote that the performance of the corresponding algorithm is worse than, better than, and similar to that of HMJCDE according to the Wilcoxon's rank test at a 0.05 significance level.

Table 6
Comparisons of HMJCDE and five recent DE variants on 30 test functions with 30*D*.

Alg Fuc	AEPD-JADE Mean error (std dev)	DE-VNS Mean error (std dev)	rank-jDE Mean error (std dev)	sinDE Mean error (std dev)	AuDE Mean error (std dev)	HMJCDE Mean error (std dev)
F1	**2.14e + 02(3.16e + 02)+**	9.76e + 04(7.29e + 04)–	4.58e + 04(3.74e + 04)–	9.08e + 05(4.34e + 05)–	1.24e + 05(6.90e + 04)–	1.51e + 03(2.53e + 03)
F2	1.52e – 12(5.93e – 12)=	**0.00e + 00(0.00e + 00)=**	9.47e – 15(1.36e – 14)=	**0.00e + 00(0.00e + 00)=**	**0.00e + 00(0.00e + 00)=**	**0.00e + 00(0.00e + 00)**
F3	1.04e – 13(1.13e – 13)=	**0.00e + 00(0.00e + 00)=**	4.36e – 14(2.45e – 14)=	3.61e – 11(1.92e – 10)–	**0.00e + 00(0.00e + 00)=**	**0.00e + 00(0.00e + 00)**
F4	2.30e – 03(1.19e – 03)–	1.44e + 01(2.76e + 01)–	7.95e – 02(1.85e – 01)–	7.48e + 00(1.89e + 01)–	2.98e + 00(1.18e + 01)–	**3.03e – 14(4.89e – 14)**
F5	**2.00e + 01(1.63e – 02)=**	2.06e + 01(4.73e – 02)–	2.03e + 01(3.80e – 02)–	2.05e + 01(5.76e – 02)–	2.06e + 01(3.94e – 02)–	**2.00e + 01(2.66e – 02)**
F6	3.93e + 00(1.96e + 00)–	4.11e + 00(1.92e + 00)–	2.43e + 00(1.63e + 00)=	**1.90e – 02(1.02e – 01)+**	1.47e + 01(1.06e + 01)–	2.38e + 00(2.92e + 00)
F7	4.92e – 03(1.00e – 02)–	4.02e – 03(6.64e – 03)–	4.93e – 04(1.88e – 03)–	**0.00e + 00(0.00e + 00)=**	1.89e – 03(5.39e – 03)–	**0.00e + 00(0.00e + 00)**
F8	**0.00e + 00(0.00e + 00)=**	3.38e + 01(3.06e + 00)–	**0.00e + 00(0.00e + 00)=**	6.63e – 02(2.52e – 01)–	3.90e + 01(6.32e + 00)–	**0.00e + 00(0.00e + 00)**
F9	3.52e + 01(8.00e + 00)=	1.15e + 02(1.21e + 01)–	4.27e + 01(7.90e + 00)=	**3.32e + 01(7.92e + 00)=**	1.04e + 02(1.55e + 01)–	3.92e + 01(1.65e + 00)
F10	2.64e – 01(4.14e – 01)=	1.18e + 03(1.84e + 02)–	**6.94e – 03(1.26e – 01)+**	5.08e + 00(2.70e + 00)–	1.47e + 03(1.71e + 02)–	1.51e – 01(2.85e – 02)
F11	**1.50e + 03(2.60e + 02)+**	4.86e + 03(2.48e + 02)–	2.41e + 03(3.34e + 02)=	1.88e + 03(4.13e + 02)+	4.71e + 03(2.40e + 02)–	2.28e + 03(4.43e + 02)
F12	**1.11e – 01(2.94e – 02)+**	1.10e + 00(1.25e – 01)–	4.41e – 01(4.18e – 02)–	5.30e – 01(1.44e – 01)–	9.40e – 01(9.72e – 02)–	3.23e – 01(9.02e – 02)
F13	2.66e – 01(4.64e – 02)–	2.88e – 01(6.07e – 02)–	2.58e – 01(4.07e – 02)=	**1.40e – 01(3.19e – 02)+**	2.78e – 01(3.84e – 02)–	2.46e – 01(4.22e – 02)
F14	**1.60e – 01(2.57e – 02)+**	2.41e – 01(3.64e – 02)–	2.90e – 01(1.26e – 02)–	2.11e – 01(3.10e – 02)=	2.40e – 01(2.87e – 02)–	2.10e – 01(3.71e – 02)
F15	3.85e + 00(8.87e – 01)=	1.12e + 01(1.19e + 00)–	5.24e + 00(7.60e – 01)–	**3.50e + 00(8.43e – 01)+**	9.58e + 00(1.04e + 00)–	3.97e + 00(1.02e + 00)
F16	9.65e + 00(5.95e – 01)=	1.16e + 01(2.21e – 01)–	9.99e + 00(2.75e – 01)=	**9.62e + 00(5.35e – 01)=**	1.15e + 01(2.15e – 01)–	9.93e + 00(5.27e – 01)
F17	5.75e + 02(1.28e + 02)–	7.29e + 02(5.70e + 02)–	1.82e + 03(1.30e + 03)–	8.24e + 04(6.37e + 04)–	6.44e + 03(6.41e + 03)–	**3.63e + 02(2.37e + 02)**
F18	1.27e + 02(3.16e + 02)–	1.87e + 01(9.41e + 00)–	3.86e + 01(2.54e + 01)–	4.36e + 02(8.46e + 02)–	4.75e + 01(1.63e + 01)–	**1.40e + 01(6.79e + 00)**
F19	4.52e + 00(1.52e + 00)–	7.98e + 00(1.51e + 01)–	4.40e + 00(8.10e – 01)–	**3.38e + 00(7.19e – 01)+**	7.22e + 00(1.08e + 01)–	3.84e + 00(7.75e – 01)
F20	8.96e + 01(1.22e + 01)–	1.86e + 01(1.19e + 01)–	2.42e + 01(3.22e + 01)–	1.03e + 01(3.15e + 00)=	2.43e + 01(5.79e + 00)–	**1.01e + 01(3.15e + 00)**
F21	3.39e + 04(8.35e + 04)–	2.32e + 02(1.86e + 02)–	3.53e + 02(4.04e + 02)–	5.67e + 03(4.72e + 03)–	1.46e + 03(9.02e + 02)–	**1.88e + 02(1.31e + 02)**
F22	1.35e + 02(8.58e + 01)–	7.14e + 01(5.92e + 01)+	8.36e + 01(5.73e + 01)=	**5.43e + 01(5.26e + 01)+**	2.74e + 02(8.11e + 01)–	8.25e + 01(7.95e + 01)
F23	3.15e + 02(3.68e – 02)=	3.15e + 02(5.78e – 14)=	3.15e + 02(5.78e – 14)=	3.15e + 02(5.78e – 14)=	3.15e + 02(2.19e – 13)=	**3.15e + 02(2.14e – 13)**
F24	2.26e + 02(5.14e + 00)=	2.32e + 02(7.27e + 00)–	2.26e + 02(3.73e + 00)=	**2.22e + 02(4.24e + 00)=**	2.26e + 02(5.55e + 00)=	2.24e + 02(2.34e + 00)
F25	2.10e + 02(2.96e + 00)–	2.06e + 02(2.76e + 00)=	2.03e + 02(7.72e – 01)=	2.04e + 02(4.16e – 01)=	2.04e + 02(2.12e + 00)=	**2.03e + 02(4.95e – 01)**
F26	1.24e + 02(4.29e + 01)–	**1.04e + 02(1.82e + 01)=**	**1.00e + 02(4.35e – 02)=**	**1.00e + 02(2.74e – 02)=**	**1.00e + 02(3.85e – 02)=**	**1.00e + 02(3.74e – 02)**
F27	4.04e + 02(6.06e + 01)=	4.24e + 02(5.75e + 01)=	3.63e + 02(4.44e + 01)=	**3.00e + 02(0.00e + 00)+**	4.01e + 02(5.58e – 01)–	3.90e + 02(3.06e + 01)
F28	6.56e + 02(1.00e + 02)+	8.91e + 02(6.28e + 01)=	**8.35e + 02(3.32e + 01)=**	7.91e + 02(2.16e + 01)+	8.96e + 02(4.08e + 01)–	8.32e + 02(3.66e + 01)
F29	7.39e + 02(3.37e + 02)–	6.28e + 02(2.10e + 02)+	7.67e + 02(7.13e + 01)=	1.31e + 03(1.90e + 02)–	1.00e + 03(1.71e + 02)–	**7.22e + 02(2.59e + 00)**
F30	1.37e + 03(6.18e + 02)–	7.90e + 02(2.83e + 02)+	1.80e + 03(8.66e + 02)–	**7.08e + 02(1.53e + 02)+**	1.15e + 03(4.20e + 02)–	8.75e + 02(3.46e + 02)
+/=/–	**5/11/14**	**3/7/20**	**1/16/13**	**9/10/11**	**0/6/24**	

"–", "+", and "=" respectively denote that the performance of the corresponding algorithm is worse than, better than, and similar to that of HMJCDE according to the Wilcoxon's rank test at a 0.05 significance level.

modal functions F4-F16, HMJCDE obtains the best results on F4, F5, F7, F8, F12, F14 and F15. Moreover, HMJCDE is better than CoDE and JADE on most of simple multimodal functions. These results validate that HMJCDE can further improve the performances of CoDE and JADE on most cases. In addition, HMJCDE also outperforms ODE on all the simple multimodal functions except F6 and F7, and outperforms SaDE on most of test functions (*i.e.*, F4-F7, F11, F12, F14 and F16). Considering the hybrid functions F17-F22, HMJCDE obtains the best results on all the hybrid functions. These results justify that HMJCDE can enhance the performance of CoDE and JADE on hybrid functions, and is better than other state-of-the-art DE variants. About the composition functions F23-F30, CoDE, JADE, ODE, SaDE and HMJCDE cannot find any near-global optimal solutions as these test functions are very complicated. Overall, HMJCDE is better than the other algorithms as it outperforms CoDE, JADE, ODE and SaDE on 20, 18, 19 and 22 out of 30 test functions respectively. Moreover, it is observed that CoDE is suitable for solving hybrid functions and complicated composition functions, while JADE fits for tackling unimodal and simple multimodal functions. These experimental results confirm our above viewpoint that CoDE and JADE are respectively good at exploration and exploitation. As HMJCDE significantly outperforms CoDE and JADE on most of test functions, they also validate that HMJCDE can effectively combine the merits of CoDE and JADE so as to well solve various kinds of GOPs.

Table 6 also demonstrates that HMJCDE is better than five recent state-of-the-art DE variants on most of test functions with 30D. More detailed, regarding the unimodal functions F1-F3, all the algorithms can find the near-global optimal solutions for F2 and F3, in which HMJCDE, DE-VNS and AuDE obtain the better accuracy. However, all the algorithms cannot find any near-global optimal solutions for F1, in which AEPD-JADE and HMJCDE respectively achieve the best and second best results. These results demonstrate that HMJCDE is better than or at least comparable with the recent state-of-the-art DE variants in solving these unimodal functions with 30*D*. Considering the simple multimodal functions F4-F16, HMJCDE is better than AEPD-JADE, DE-VNS, rank-jDE, SinDE and AuDE on 4, 13, 6, 5, and 13 test functions respectively, but is worse than AEPD-JADE, DE-VNS, rank-jDE, SinDE and AuDE on 3, 0, 1, 4 and 0 test functions. Obviously, HMJCDE outperforms DE-VNS and AuDE significantly, and is better than or at least similar to AEPD-JADE, rank-jDE and SinDE on these simple multimodal functions with 30*D*. Concerning the hybrid functions F17-F22, HMJCDE is only beaten by DE-VNS on F22 and SinDE on F19 and F22, and outperforms or at least similar to the other algorithms on the rest test functions. Obviously, HMJCDE surpasses all the compared algorithms in tackling these hybrid test functions with 30*D*. On composition functions F23-F30, SinDE obtains the better results on a small part of test functions (*e.g.*, F24, F27, F30), while all the algorithms get the similar results nearly for the remaining test functions. In summary, HMJCDE outperforms AEPD-JADE, DE-VNS, rank-jDE, sinDE and AuDE on 14, 20, 13, 11 and 24 out of 30 test functions respectively, while HMJCDE is only beaten by AEPD-JADE, DE-VNS, rank-jDE and sinDE on 5, 3, 1, and 9 out of 30 test functions. Therefore, when considering all the test functions with 30D, HMJCDE performs better than AEPD-JADE, DE-VNS, rank-jDE, sinDE and AuDE.

5.3.2. Discussions on test functions with 50D

Tables 7 and 8 give the experimental results of all the compared algorithms on the 30 test functions with 50*D*. From Table 7, considering unimodal functions F1-F3, JADE and HMJCDE obtain the best and second best results respectively, and HMJCDE is significantly better than CoDE, JADE, ODE and SaDE on F2 and F3. Therefore,

Table 7
Comparisons of HMJCDE and four classical DE variants on 30 test functions with 50*D*.

Alg Fuc	CoDE Mean error (std dev)	JADE Mean error (std dev)	ODE Mean error (std dev)	SaDE Mean error (std)	HMJCDE Mean error (std dev)
F1	3.87e+05(1.66e+05)–	**1.47e+04(1.07e+04)+**	1.80e+06(5.23e+05)–	8.67e+05(2.22e+05)–	8.72e+04(4.11e+04)
F2	5.72e−03(2.09e−02)–	1.54e−13(1.96e−13)–	1.61e+03(1.61e+03)–	3.31e+03(2.11e+03)–	**5.68e−15(1.73e−14)**
F3	3.91e−04(1.46e−03)–	3.98e+03(1.96e+03)–	1.03e+00(1.83e+00)–	3.65e+03(2.39e+03)–	**7.93e−08(1.58e−07)**
F4	3.96e+01(3.64e+01)–	2.29e+01(4.22e+01)–	8.51e+01(1.77e+01)–	8.03e+01(3.40e+01)–	**1.71e+01(3.69e+01)**
F5	2.09e+01(5.51e−02)–	2.04e+01(3.50e−02)–	2.12e+01(2.54e−02)–	2.07e+01(3.86e−02)–	**2.01e+01(4.59e−02)**
F6	1.51e+01(3.23e+00)–	1.70e+01(6.57e+00)–	**7.72e−02 (2.02e−01)+**	1.89e+01(2.60e+00)–	5.63e+00(3.87e+00)
F7	4.93e−04(1.88e−03)=	1.31e−03(3.76e−03)–	**6.06e−14(5.77e−14)+**	1.08e−02(8.39e−03)–	5.75e−04(2.11e−03)
F8	7.55e+01(3.90e+00)–	**0.00e+00(0.00e+00)=**	2.80e+02(3.22e+01)–	1.23e+00(1.27e+00)–	**0.00e+00(0.00e+00)**
F9	2.82e+02(1.60e+01)–	8.26e+01(7.20e+00)–	3.60e+02(1.25e+01)–	8.84e+01(1.68e+01)–	**7.95e+01(1.98e+01)**
F10	3.52e+03(2.54e+02)–	**9.99e−03(9.51e−03)+**	1.13e+04(1.25e+03)–	1.41e+00(1.40e+00)–	2.78e−01(2.88e−01)
F11	1.04e+04(3.11e+02)–	4.81e+03(5.29e+02)–	1.33e+04(2.29e+02)–	6.64e+03(1.82e+03)–	**4.52e+03(8.82e+02)**
F12	1.55e+00(1.86e−01)–	**2.50e−01(2.70e−02)=**	3.42e+00(2.61e−01)–	1.09e+00(1.11e−01)–	3.89e−01(1.07e−01)
F13	5.03e−01(4.46e−02)–	**3.27e−01(4.08e−02)=**	4.93e−01(5.05e−02)–	4.30e−01(3.59e−02)–	3.37e−01(4.24e−02)
F14	3.11e−01(2.81e−02)–	2.79e−01(4.05e−02)=	3.21e−01(3.84e−02)–	3.14e−01(2.58e−02)–	**2.78e−01(2.40e−02)**
F15	2.61e+01(3.64e+01)–	**7.06e+00(1.00e+00)=**	3.13e+01(1.58e+00)–	1.53e+01(4.73e+00)–	7.74e+00(2.21e+00)
F16	2.11e+01(2.10e−01)–	**1.77e+01(4.80e−01)+**	2.24e+01(1.69e−01)–	2.02e+01(2.89e−01)–	1.85e+01(6.89e−01)
F17	2.92e+03(1.75e+03)=	**2.52e+03(7.44e+02)+**	1.23e+04(6.98e+03)–	6.16e+04(3.31e+04)–	3.43e+03(1.88e+00)
F18	**2.83e+01(1.37e+01)+**	1.82e+02(5.30e+01)–	1.41e+02(1.31e+01)–	5.66e+02(5.06e+02)–	8.97e+01(5.35e−01)
F19	1.21e+01(5.97e−01)+	1.40e+01(6.11e+00)–	**1.16e+01(1.00e+00)=**	1.69e+01(8.60e+00)–	1.27 e+01(8.67e+01)
F20	3.69e+01(1.56e+01)=	6.35e+03(5.95e+03)–	1.09e+02(2.17e+01)–	7.26e+02(5.14e+02)–	**3.32e+01(9.51e+00)**
F21	1.17e+03(5.07e+02)–	4.25e+04(2.26e+05)–	3.14e+03(8.23e+02)–	7.75e+04(4.67e+04)–	**8.97e+02(4.44e+02)**
F22	5.78e+02(1.70e+02)–	5.29e+02(1.27e+02)–	1.06e+03(4.25e+02)–	5.12e+02(1.69e+02)–	**3.81e+02(2.09e+02)**
F23	**3.44e+02(2.89e−13)=**	**3.44e+02(1.88e−13)=**	**3.44e+02(2.70e−13)=**	**3.44e+02(2.86e−13)=**	**3.44e+02(2.89e−13)**
F24	2.65e+02(1.90e+00)=	2.74e+02(1.84e+00)=	**2.44e+02(8.15e+00)+**	2.76e+02(3.06e+00)–	2.73e+02(2.02e+00)
F25	**2.00e+02(2.00e−01)+**	2.15e+02(6.68e+00)–	2.06e+02(9.66e−01)+	2.26e+02(1.08e+01)–	2.09e+02(5.13e+00)
F26	**1.00e+02(1.82e−01)=**	1.04e+02(1.82e+01)–	**1.00e+02(5.03e−02)=**	1.90e+02(3.04e+01)–	**1.00e+02(4.31e−02)**
F27	**3.00e+02(3.89e+01)+**	4.31e+02(4.86e+01)=	3.24e+02(3.73e+01)+	7.58e+02(7.37e+01)–	4.31e+02(4.89e+01)
F28	1.03e+03(1.27e+02)=	1.14e+03(5.22e+01)=	**9.82e+02(3.03e+02)+**	1.44e+03(1.04e+02)–	1.17e+03(7.20e+01)
F29	**4.12e+02(1.44e+02)+**	8.89e+02(7.19e+01)=	8.35e+02(2.78e+02)=	1.33e+03(2.58e+02)–	8.60e+02(4.93e+01)
F30	**8.01e+03(3.20e+02)+**	9.66e+03(9.97e+02)–	9.65e+03(3.99e+02)–	1.23e+04(1.85e+03)–	8.95e+03(5.08e+02)
+/=/–	**6/7/17**	**4/10/16**	**6/4/20**	**0/1/29**	

"–", "+", and "=" respectively denote that the performance of the corresponding algorithm is worse than, better than, and similar to that of HMJCDE according to the Wilcoxon's rank test at a 0.05 significance level.

Table 8
Comparisons of HMJCDE and five recent DE variants on 30 test functions with 50*D*.

Alg Fuc	AEPD-JADE Mean error (std dev)	DE-VNS Mean error (std dev)	rank-jDE Mean error (std dev)	sinDE Mean error (std dev)	AuDE Mean error (std dev)	HMJCDE Mean error (std dev)
F1	**7.23e+03(5.34e+03)+**	6.19e+05(2.33e+05)–	3.56e+05(1.92e+05)–	1.60e+06(1.94e+05)–	6.92e+05(2.65e+05)–	8.72e+04(4.11e+04)
F2	4.38e−04(2.40e−03)–	6.05e+03(7.44e+03)–	1.24e−13(7.59e−14)=	3.63e+03(3.26e+03)–	2.92e−04(7.90e−04)–	**5.68e−15(1.73e−14)**
F3	5.01e+03(4.32e+03)–	4.38e+00(5.43e+00)–	3.25e−06(1.60e−05)–	4.89e+02(4.79e+02)–	4.90e−06(1.00e−05)–	**7.93e−08(1.58e−07)**
F4	**1.49e+01(2.63e+01)+**	6.44e+01(4.56e+01)–	6.99e+01(3.68e+01)–	9.00e+01(4.44e+00)–	2.83e+01(3.29e+01)–	1.71e+01(3.69e+01)
F5	**2.00e+01(2.06e−02)=**	2.08e+01(4.36e−02)–	2.04e+01(3.11e−02)–	2.07e+01(3.96e−02)–	2.08e+01(3.56e−02)–	2.01e+01(4.59e−02)
F6	1.20e+01(5.50e+00)–	2.18e+01(4.84e+00)–	1.12e+01(4.50e+00)–	**3.00e−02(1.63e−01)+**	3.11e+01(1.95e+01)–	5.63e+00(3.87e+00)
F7	1.26e−02(2.69e−02)–	1.14e−02(1.95e−02)–	2.31e−13(6.99e−14)+	**7.20e−14(5.57e−14)+**	1.15e−03(3.97e−03)=	5.75e−04(2.11e−03)
F8	1.14e−13(7.27e−13)=	1.01e+02(7.48e+00)–	6.82e−14(5.66e−14)=	2.29e−01(2.48e+00)–	9.45e+01(2.60e+01)–	**0.00e+00(0.00e+00)**
F9	1.03e+02(2.19e+01)–	2.17e+02(7.04e+01)–	8.60e+01(1.21e+01)=	**6.15e+01(9.95e+00)+**	2.28e+02(2.10e+01)–	7.95e+01(1.98e+01)
F10	8.40e+00(3.00e+01)–	3.98e+03(2.24e+02)–	**5.47e−02(1.67e−01)+**	3.12e+01(3.58e+01)–	3.82e+03(3.36e+02)–	2.78e−01(2.88e−01)
F11	**3.56e+03(4.82e+02)+**	1.03e+04(3.56e+02)–	5.20e+03(3.35e+02)–	4.23e+03(5.53e+02)=	9.78e+03(4.17e+02)–	4.52e+03(8.82e+02)
F12	**1.45e−01(3.40e−02)+**	1.43e+00(1.50e−01)–	4.77e−01(5.42e−02)=	5.90e−01(1.42e−01)–	1.27e+00(1.46e−01)–	3.89e−01(1.07e−01)
F13	4.08e−01(6.92e−02)–	4.83e−01(6.64e−02)–	3.27e−01(4.60e−02)=	**2.27e−01(5.53e−02)+**	3.97e−01(5.03e−02)–	3.37e−01(4.24e−02)
F14	2.29e−01(1.87e−02)+	2.97e−01(3.56e−02)=	4.33e−01(1.73e−01)–	**2.29e−01(3.77e−02)+**	2.97e−01(4.36e−02)=	2.78e−01(2.40e−02)
F15	1.12e+01(1.68e+00)–	2.56e+01(2.52e+00)–	1.14e+01(1.07e+00)–	**6.67e+00(1.35e+00)=**	2.19e+01(1.76e+00)–	7.74e+00(2.21e+00)
F16	**1.81e+01(6.02e−01)=**	2.08e+01(2.38e−01)–	1.84e+01(3.12e−01)=	1.83e+01(7.26e−01)=	2.08e+01(2.28e−01)–	1.85e+01(6.89e−01)
F17	3.90e+04(7.16e+04)–	4.93e+04(2.51e+04)–	1.61e+04(9.06e+03)–	2.83e+05(1.49e+05)–	8.83e+04(4.71e+04)–	**3.43e+03(1.88e+00)**
F18	7.30e+02(8.98e+02)–	9.67e+02(1.00e+03)–	9.14e+02(8.16e+02)–	3.55e+02(3.05e+02)–	4.75e+02(5.53e+02)–	**8.97e+01(5.35e−01)**
F19	2.00e+01(1.27e+01)–	2.47e+01(2.50e+01)–	1.20e+01(1.37e+00)=	**9.28e+00(7.70e−01)=**	3.30e+01(2.44e+01)–	1.27e+01(8.67e+01)
F20	5.36e+03(6.56e+03)–	1.96e+02(9.39e+01)–	9.42e+01(4.85e+01)–	4.22e+02(3.98e+02)–	7.89e+01(1.75e+01)–	**3.32e+01(9.51e+00)**
F21	6.77e+04(9.47e+04)–	1.23e+04(1.40e+04)–	7.93e+03(8.15e+03)–	2.29e+05(1.42e+05)–	3.91e+04(3.18e+04)–	**8.97e+02(4.44e+02)**
F22	5.50e+02(1.95e+02)–	6.26e+02(2.52e+02)–	4.71e+02(1.42e+02)–	**3.13e+02(1.63e+02)+**	8.89e+02(2.22e+02)–	3.81e+02(2.09e+02)
F23	**3.38e+02(1.58e+00)+**	3.44e+02(3.04e−13)=	3.44e+02(2.02e−13)=	3.44e+02(2.60e−13)=	3.44e+02(2.89e−13)=	3.44e+02(2.89e−13)
F24	2.76e+02(7.12e+00)=	2.86e+02(5.29e+00)=	2.72e+02(2.45e+00)=	**2.65e+02(3.47e+00)+**	2.73e+02(5.20e+00)=	2.73e+02(2.02e+00)
F25	2.28e+02(4.52e+00)–	2.26e+02(6.26e+00)–	**2.08e+02(3.14e+00)=**	2.09e+02(1.14e+00)=	2.14e+02(7.82e+00)–	2.09e+02(5.13e+00)
F26	1.07e+02(2.53e+01)=	1.31e+02(4.63e+01)–	1.04e+02(1.82e+01)=	1.04e+02(1.82e+01)–	1.04e+02(1.82e+01)–	**1.00e+02(4.31e−02)**
F27	7.79e+02(1.14e+02)–	1.06e+03(8.73e+01)–	5.15e+02(8.73e+01)–	**3.26e+02(2.19e+01)+**	5.79e+02(1.56e+02)–	4.31e+02(4.89e+01)
F28	**6.79e+02(2.66e+02)+**	1.47e+03(2.59e+02)–	1.24e+03(2.15e+02)=	1.10e+03(3.27e+01)+	1.40e+03(1.44e+02)–	1.17e+03(7.20e+01)
F29	1.10e+03(2.90e+02)–	1.09e+03(2.39e+02)–	9.95e+02(1.55e+02)–	1.73e+03(3.52e+02)–	1.18e+03(2.39e+02)–	**8.60e+02(4.93e+01)**
F30	**4.83e+03(2.25e+03)+**	1.07e+04(1.32e+03)–	9.39e+03(6.23e+02)=	8.87e+03(3.82e+02)=	9.99e+03(7.37e+02)–	8.95e+03(5.08e+02)
+/=/–	**8/5/17**	**0/3/27**	**2/13/15**	**9/7/14**	**0/4/26**	

"–", "+", and "=" respectively denote that the performance of the corresponding algorithm is worse than, better than, and similar to that of HMJCDE according to the Wilcoxon's rank test at a 0.05 significance level.

HMJCDE is better than these four classical state-of-the-art DE variants on these unimodal functions with 50*D*. Regarding multimodal functions F4-F16, HMJCDE gets the best results on F4, F5, F8, F9, F11 and F14, JADE obtains the best results on F10, F12, F13, F15, F16, and ODE gains the best results on F6 and F7. HMJCDE outperforms CoDE and SaDE on all these test functions (excluding F7), and surpasses ODE on all the test functions except F6 and F7. HMJCDE is better than, similar to and worse than JADE on 6, 5 and 2 test functions. Therefore, HMJCDE is better than or at least similar to these four classical state-of-the-art DE variants on these simple multimodal functions with 50*D*. Considering hybrid functions F17-F22, JADE, CoDE and ODE get the best performance on F17, F18 and F19 respectively, HMJCDE performs best on F20-F22. HMJCE surpasses ODE and SaDE on all the test functions except F19, and also outperforms JADE on F18-F22. Therefore, HMJCDE is better than or at least similar to the four classical state-of-the-art DE variants on these hybrid functions with 50*D*. Regarding composition functions F23-F30, all the compared algorithms have similar performance on F23. CoDE obtains the best results on F25, F27, F29 and F30, and ODE gains the best results on F24 and F28. CoDE, ODE and HMJCDE obtains the best results on F26. Obviously, HMJCDE is worse than CoDE and ODE, and similar to JADE and SaDE on these composition functions with 50*D*. Overall, HMJCDE is better than CoDE, JADE, ODE and SaDE on 17, 16, 20, 29 test functions respectively. In the contrary, CoDE, JADE, ODE and SaDE surpass HMJCDE on 6, 4, 6 and 0 test functions respectively. Therefore, HMJCDE is better than these four classical state-of-the-art DE variants when considering all the 30 test functions with 50*D*.

From Table 8, considering unimodal functions F1-F3, AEPD-JADE and HMJCDE obtain the best and second best results on F1 respectively, HMJCDE gains the best results on F2 and F3. Therefore, HMJCDE is better than these five recent DE variants on these unimodal functions with 50*D*. Regarding simple multimodal functions F4-F16, AEPD-JADE gets the best results on F4, F5, F11, F12 and F16, rank-jDE obtains the best results on F10, sinDE performs best on F6, F7, F9, F12-F5, and HMJCDE shows the best performance on F8. Although HMJCDE only gets the best performance on one functions, HMJCDE is better than AEPD-JADE, DE-VNS, rank-jDE, SinDE and AuDE on 6, 12, 6, 5 and 11 functions. In the contrary, AEPD-JADE, DE-VNS, rank-jDE, SinDE and AuDE surpass on HMJCDE on 4, 0, 2, 5 and 0 functions. Therefore, HMJCDE is better than or at least similar to these recent DE variants on these simple multimodal functions with 50*D*. Considering hybrid functions F17-F22, SinDE performs best on F19, SinDE and HMJCDE obtain the best and second best results on F22 respectively, HMJCDE performs best on other test functions (F17, F18, F20 and F21). Obviously, HMJCDE is better than these five recent DE variants on these hybrid functions with 50*D*. Regarding composition functions F22-F30, AEPD-JADE, rank-jDE, sinDE and HMJCDE obtain the best results on 3 functions (F23, F28, F30), F25, 2 functions (F24, F27) and 2 functions (F26, F29) respectively. More detailed, HMJCDE is better than AEPD-JADE, DE-VENS, rank-jDE, sinDE and AuDE on 3, 6, 2, 2 and 6 test functions. In the contrary, AEPD-JADE, DE-VENS, rank-jDE, sinDE and AuDE surpass HMJCDE on 3, 0, 0, 3 and 0 functions. Obviously, HMJCDE performs similarly with AEPD-JADE and sinDE, and better than DE-VNS, rank-jDE and AuDE on these composition functions with 50D.

To conclude, the optimization ability of HMJCDE will deteriorate to a certain extent when the dimension of search space *D* is increased from 30 to 50. However, it still obtains the best comprehensive performance as HMJCDE is better than CoDE, JADE, ODE, SaDE, AEPD-JADE, DE-VNS, rank-jDE, SinDE and AuDE on 17, 16, 20, 29, 17, 27, 15, 14 and 26 out of the 30 test functions respectively, while HMJCDE is only beaten by CoDE, JADE, ODE, SaDE, AEPD-JADE, DE-VNS, rank-jDE, SinDE and AuDE on 6, 4, 6, 0, 8, 0, 2, 9 and 0 test functions respectively.

5.3.3. *Discussions on test functions with 100D*

When the dimension of search space *D* is further raised to 100, the experimental results are given in Tables 9 and 10. From Table 9, considering unimmodal functions F1-F3, JADE and HMJCDE obtain the best and seconded best results on F1, and HMJCDE performs best on F2 and F3. Therefore, HMJCDE is better than CoDE, JADE, ODE and SaDE on these unimodal functions with 100*D*. Regarding simple multimodal functions F14-F16, HMJCDE gains the best results on F4, F5, F11 and F13-F15, JADE gets the best results on F8-F10, F12 and F16, ODE obtains the best results on F6 and F7. In more detail, HMJCDE is better than CoDE, JADE, ODE and SaDE on 10, 4, 10, and 13 test functions. On the contrary, HMJCDE is only beaten by CoDE, JADE, ODE and SaDE on 1, 3, 1 and 0 test functions. Therefore, HMJCDE is better than these four classical state-of-the-art DE variants on these simple multimodal functions with 100*D*. Considering hybrid functions F17-F22, HMJCDE, CoDE, JADE and SaDE obtain the best performance on F18, F20, 2 functions (F17, F21) and 2 functions (F19, F22). In more detail, HMJCDE outperforms CoDE, JADE, ODE and SaDE on 3, 4, 6 and 3 out of 6 test functions, On the contrary, CoDE, JADE, ODE and SaDE is only better than HMJCDE on 1, 2, 0, and 1 test functions. Therefore, HMJCDE performs better than or at least similarly with these classical state-of-the-art DE variants on these hybrid functions with 100*D*. Regarding composition functions F23-F30, HMJCDE obtains the best results on F26 and F29, CoDE gets the best result on F25, ODE gains the best results on F23, F24, F27, F28 and F30. In more detail, HMJCDE outperforms CoDE, JADE, ODE, and SaDE on 2, 5, 3 and 6 test functions respectively, but HMJCDE is beaten by CoDE, JADE, ODE and SaDE on 3, 0, 4 and 0 test functions respectively. Therefore, HMJCDE is slightly worse than CoDE and ODE, better than JADE and SaDE on these composition functions with 100*D*.

From Table 10, considering unimodal functions F1-F3, AEPD-JADE, HMJCDE and AuDE perform best on F1, F2 and F3 respectively. HMJCDE is at least similar to AEPD-JADE, rank-jDE and AuDE, and better than DE-VNS and sinDE on these unimodal functions with 100*D*. Regarding simple multimodal functions F4-F16, HMJCDE is better than AEPD-JADE, DE-VNS, rank-jDE, SinDE and AuDE on 7, 10, 8, 8 and 12 out of 13 test functions respectively. On the contrary, HMJCDE is only beaten by AEPD-JADE, DE-VNS, rank-jDE, SinDE and AuDE on 2, 0, 1, 5, and 0 test functions respectively. Therefore, HMJCDE is at least similar to AEPD-JADE, rank-jDE and SinDE, and better than DE-VNS and AuDE on these simple functions with 100*D*. Considering hybrid functions F17-F22, HMJCDE is better than AEPD-JADE, DE-VNS, rank-jDE, SinDE and AuDE on 4, 6, 3, 3 and 4 test functions respectively. On the contrary, HMJCDE is only beaten by AEPD-JADE, DE-VNS, rank-jDE, SinDE and AuDE on 1, 0, 1, 2 and 0 functions. Therefore, HMJCDE performs similarly to AEPD-JADE, rank-jDE and SinDE, and better than DE-VNS and AuDE on these hybrid functions with 100D. Regarding composition functions F23-F30, HMJCDE outperforms AEPD-JADE, DE-VNS, rank-jDE, SinDE and AuDE on 4, 6, 6, 4 and 6 test functions respectively. On the contrary, AEPD-JADE, DE-VNS, rank-jDE, SinDE and AuDE surpass HMJCDE on 3, 0, 0, 1 and 0 test functions respectively. Therefore, HMJCDE is at least similar to AEPD-JADE and sinDE, better than DE-VNS, rank-jDE, and AuDE on these composition functions with 100*D*.

In summary, the performance of HMJCDE will also degrade to a certain extent when the dimension of search space *D* is increased from 50 to 100. However, it still obtains the best comprehensive performance as HMJCDE outperforms CoDE, JADE, ODE, SaDE, AEPD-JADE, DE-VNS, rank-jDE, SinDE and AuDE on 18, 15, 22, 25, 17, 25, 18, 18 and 24 out of all the 30 test functions respectively, while HMJCDE is only beaten by CoDE, JADE, ODE, SaDE, AEPD-JADE,

Table 9
Comparisons of HMJCDE and four classical DE variants on 30 test functions with 100*D*.

Alg Fuc	CoDE Mean error(std dev)	JADE Mean error (std dev)	ODE Mean error (std dev)	SaDE Mean error (std)	HMJCDE Mean error(std dev)
F1	1.38e+06(3.57e+05)−	**1.48e+05(1.84e+05)+**	6.70e+06(2.16e+06)−	4.59e+06(9.67e+05)−	2.70e+05(9.60e+04)
F2	8.60e+00(1.71e+01)−	1.68e−09(3.32e−09)−	1.18e+04(1.05e+04)−	1.26e+04(6.73e+03)−	**4.03e−10(1.44e−09)**
F3	3.53e+02(3.78e+02)−	5.90e+03(3.39e+03)−	2.18e+03(1.63e+03)−	4.20e+03(2.02e+03)−	**3.05e+01(4.52e+01)**
F4	1.65e+02(3.37e+01)−	8.89e+01(6.06e+01)=	1.92e+02(5.32e+01)−	2.40e+02(6.70e+01)−	**7.63e+01(5.17e+01)**
F5	2.12e+01(2.25e−02)−	2.05e+01(1.26e−01)−	2.13e+01(3.19e−02)−	2.10e+01(2.93e−02)−	**2.00e+01(7.59e−02)**
F6	1.53e+01(4.10e+00)+	4.48e+01(1.50e+01)−	**1.27e+01(2.82e+00)+**	5.88e+01(4.68e+00)−	3.35e+01(8.54e+00)
F7	7.39e−04(2.83e−03)=	1.97e−03(4.73e−03)=	**6.57e−04(2.58e−03)=**	4.10e−03(6.90e−03)−	2.54e−03(5.62e−03)
F8	3.48e+02(1.69e+01)−	**1.14e−13(0.00e+00)=**	2.84e+02(1.88e+02)−	9.94e−01(9.78e−01)−	1.98e−01(4.82e−01)
F9	3.52e+02(2.40e+02)=	**1.47e+02(2.16e+01)+**	8.40e+02(2.19e+01)−	2.53e+02(2.94e+01)−	2.21e+02(4.17e+01)
F10	1.29e+04(6.71e+02)−	**1.73e−02(1.12e−02)+**	2.02e+04(2.35e+03)−	1.66e+02(4.82e+01)−	1.46e+00(1.02e+00)
F11	2.68e+04(6.18e+02)−	1.05e+04(5.12e+02)=	3.02e+04(5.33e+02)−	2.20e+04(4.62e+02)−	**1.03e+04(1.08e+03)**
F12	2.69e+00(2.03e−01)−	**3.33e−01(3.19e−02)+**	4.11e+00(1.50e−01)−	1.86e+00(1.40e−01)−	4.95e−01(1.53e−01)
F13	5.59e−01(4.14e−02)−	4.09e−01(4.58e−02)=	5.86e−01(5.55e−02)−	4.65e−01(3.84e−02)−	**4.00e−01(4.44e−02)**
F14	3.37e−01(2.43e−02)−	3.30e−01(6.66e−02)−	3.43e−01(2.47e−02)=	3.19e−01(1.84e−02)−	**3.01e−01(4.82e−01)**
F15	6.48e+01(7.42e+00)−	2.91e+01(3.63e+00)−	7.43e+01(1.97e+00)−	4.48e+01(1.24e+01)−	**2.50e+01(3.35e+00)**
F16	4.56e+01(2.67e−01)−	**4.00e+01(6.09e−01)=**	4.66e+01(2.37e−01)−	4.41e+01(3.55e−01)−	4.07e+01(8.43e−01)
F17	2.47e+05(9.26e+04)−	**1.21e+04(4.09e+03)+**	6.85e+05(2.91e+05)−	2.58e+05(9.09e+04)−	4.16e+04(2.59e−13)
F18	4.50e+02(4.45e+02)−	1.19e+03(1.15e+03)−	7.56e+02(5.46e+02)−	7.07e+02(5.54e+02)=	**3.88e+02(4.56e+00)**
F19	9.36e+01(3.42e+00)−	9.88e+01(1.18e+01)−	9.44e+01(9.56e+00)−	**8.30e+01(2.98e+01)=**	9.00e+01(1.96e+01)
F20	**3.37e+02(1.24e+02)+**	6.94e+03(1.45e+04)−	1.54e+03(6.98e+02)−	2.74e+03(1.50e+03)−	4.14e+02(1.35e+02)
F21	1.04e+05(5.15e+04)=	**3.76e+03(1.03e+03)+**	1.73e+05(7.42e+04)−	1.67e+05(6.76e+04)−	1.03e+04(8.08e+03)
F22	1.73e+03(5.11e+02)=	1.61e+03(2.35e+02)−	4.17e+03(2.13e+02)−	**1.33e+03(2.63e+02)+**	1.57e+03(3.68e+02)
F23	3.48e+02(1.46e−13)=	3.48e+02(1.01e−12)=	**3.17e+02(3.31e+01)+**	3.48e+02(6.01e−08)=	3.48e+02(2.58e−13)
F24	3.76e+02(3.50e+00)=	4.00e+02(5.26e+00)−	**3.59e+02(2.71e+01)+**	3.87e+02(6.53e+00)=	3.88e+02(4.56e+00)
F25	**2.22e+02(2.26e+01)=**	2.75e+02(5.67e+00)−	2.56e+02(1.64e+01)−	2.41e+02(1.92e+01)−	2.25e+02(2.31e+01)
F26	1.97e+02(1.82e+01)−	2.00e+02(5.77e−03)−	2.01e+02(1.17e+00)−	2.00e+02(2.99e−02)−	**1.76e+02(1.82e+01)**
F27	5.64e+02(8.18e+01)+	1.07e+03(1.02e+02)−	**4.85e+02(6.38e+01)+**	1.41e+03(8.91e+01)−	9.30e+02(8.70e+01)
F28	2.16e+03(1.14e+02)+	2.38e+03(2.92e+02)=	**2.15e+03(6.08e+02)=**	2.80e+03(2.51e+02)−	2.34e+03(2.66e+02)
F29	1.70e+03(1.66e+02)−	1.34e+03(1.74e+02)=	2.02e+03(3.04e+02)−	1.96e+03(1.78e+02)−	**1.33e+03(1.79e+02)**
F30	4.98e+03(9.18e+02)+	8.54e+03(1.43e+03)−	**4.93e+03(1.13e+03)+**	1.05e+04(2.51e+03)−	7.78e+03(1.08e+03)
+/=/−	**5/7/18**	**6/9/15**	**5/3/22**	**1/4/25**	

"−", "+", and "=" respectively denote that the performance of the corresponding algorithm is worse than, better than, and similar to that of HMJCDE according to the Wilcoxon's rank test at a 0.05 significance level.

Table 10
Comparisons of HMJCDE and five recent DE variants on 30 test functions with 100*D*.

Alg Fuc	AEPD-JADE Mean error(std dev)	DE-VNS Mean error (std dev)	rank-jDE Mean error (std dev)	sinDE Mean error (std dev)	AuDE Mean error (std dev)	HMJCDE Mean error(std dev)
F1	**1.20e+05(5.53e+04)+**	2.27e+06(5.08e+05)−	1.53e+06(4.88e+05)−	1.89e+07(5.03e+06)−	2.22e+06(5.72e+05)−	2.70e+05(9.60e+04)
F2	1.57e+03(8.59e+03)−	1.41e+04(1.25e+04)−	9.51e−11(4.26e−10)=	8.84e+03(5.74e+03)−	2.72e−06(6.20e−06)−	**4.03e−10(1.44e−09)**
F3	8.57e+03(5.99e+03)−	1.42e+03(1.64e+03)−	2.80e−03(1.40e−02)+	3.89e+03(1.53e+03)−	**2.99e−03(5.95e−03)+**	3.05e+01(4.52e+01)
F4	1.09e+02(4.85e+01)−	1.82e+02(4.90e+01)−	1.56e+02(3.43e+01)−	1.76e+02(4.07e+01)−	1.21e+02(4.42e+01)−	**7.63e+01(5.17e+01)**
F5	2.00e+01(5.51e−02)=	2.11e+01(2.29e−02)−	2.07e+01(2.90e−02)−	2.10e+01(2.49e−02)−	2.10e+01(1.80e−02)−	**2.00e+01(7.59e−02)**
F6	7.05e+01(5.66e+00)−	9.18e+01(5.18e+00)−	6.13e+01(8.86e+00)−	**4.87e+00(2.89e+00)+**	9.30e+01(3.24e+01)−	3.35e+01(8.54e+00)
F7	2.06e−02(6.33e−02)=	6.31e−03(1.28e−02)=	9.04e−04(2.78e−03)=	**3.79e−12(5.12e−12)+**	3.03e−03(1.06e−02)−	2.54e−03(5.62e−03)
F8	6.00e+00(8.48e+00)−	2.87e+02(4.70e+01)−	**6.63e−02(2.52e−01)=**	5.26e+01(1.07e+01)−	2.72e+02(8.00e+01)−	1.98e−01(4.82e−01)
F9	3.60e+02(4.44e+01)−	3.90e+02(3.77e+01)−	**2.18e+02(2.21e+01)=**	1.25e+02(2.36e+01)+	6.32e+02(8.26e+01)−	2.21e+02(4.17e+01)
F10	7.32e+00(2.15e+01)−	1.30e+04(5.37e+02)−	**2.53e−02(6.16e−02)+**	1.53e+03(3.61e+02)−−−	1.19e+04(5.45e+02)−	1.46e+00(1.02e+00)
F11	**1.02e+04(9.67e+02)=**	2.47e+04(6.07e+02)−	1.36e+04(5.11e+02)−	1.17e+04(9.24e+02)−	2.41e+04(5.88e+02)−	1.03e+04(1.08e+03)
F12	**2.34e−01(4.21e−02)+**	2.15e+00(1.38e−01)−	6.29e−01(5.00e−02)−	1.54e+00(2.19e−01)−	1.80e+00(1.47e−01)−	4.95e−01(1.53e−01)
F13	4.60e−01(4.10e−02)−	5.48e−01(7.42e−02)=	4.60e−01(4.88e−02)−	4.64e−01(5.41e−02)−	4.88e−01(5.31e−02)−	**4.00e−01(4.44e−02)**
F14	**2.22e−01(1.34e−02)+**	3.12e−01(2.77e−02)=	3.52e−01(2.08e−02)−	2.72e−01(1.79e−02)+	3.21e−01(2.89e−02)=	3.01e−01(4.82e−01)
F15	3.35e+01(4.29e+00)−	7.45e+01(1.01e+01)−	3.39e+01(5.26e+00)−	**1.70e+01(3.07e+00)+**	6.28e+01(8.95e+00)−	2.50e+01(3.35e+00)
F16	**4.04e+01(1.16e+00)=**	4.47e+01(4.29e−01)−	4.05e+01(6.35e−01)=	4.27e+01(6.94e−01)−	4.45e+01(2.66e−01)−	4.07e+01(8.43e−01)
F17	6.99e+05(1.29e+06)−	1.75e+05(6.64e+04)−	1.09e+05(5.02e+04)−	2.33e+06(6.37e+05)−	9.18e+05(4.97e+05)−	**4.16e+04(2.59e−13)**
F18	1.59e+03(1.70e+03)−	2.08e+03(1.76e+03)−	1.68e+03(1.71e+03)−	**1.66e+02(1.60e+02)+**	6.03e+02(4.09e+02)−	3.88e+02(4.56e+00)
F19	**6.78e+01(1.56e+01)+**	1.09e+02(3.92e+01)−	9.56e+01(3.69e+00)=	8.92e+01(1.45e+00)=	9.28e+01(3.32e+01)=	9.00e+01(1.96e+01)
F20	1.15e+04(2.13e+04)−	4.43e+03(2.69e+03)−	**3.70e+02(9.80e+01)+**	6.66e+03(2.40e+03)−	4.31e+02(1.68e+01)=	4.14e+02(1.35e+02)
F21	2.66e+05(3.91e+05)−	9.73e+04(3.05e+04)−	3.84e+04(1.81e+04)−	1.70e+06(5.94e+05)−	5.42e+05(3.43e+05)−	**1.03e+04(8.08e+03)**
F22	1.41e+03(2.83e+02)+	2.81e+03(8.02e+02)−	1.66e+03(2.91e+02)=	**1.19e+03(3.29e+02)+**	2.81e+03(5.20e+02)−	1.57e+03(3.68e+02)
F23	**3.46e+02(2.19e−01)+**	3.48e+02(3.20e−13)=	3.48e+02(1.44e−12)=	3.48e+02(2.80e−05)=	3.48e+02(8.43e−13)=	3.48e+02(2.58e−13)
F24	3.89e+02(1.44e+01)=	4.30e+02(1.60e+01)=	3.89e+02(4.02e+00)=	**3.63e+02(1.57e+00)=**	3.91e+02(1.03e+01)=	3.88e+02(4.56e+00)
F25	2.80e+02(1.22e+01)−	2.66e+02(8.56e+00)−	2.66e+02(1.81e+01)−	2.49e+02(4.36e+00)−	2.38e+02(2.45e+01)−	**2.25e+02(2.31e+01)**
F26	2.00e+02(1.02e−02)−	2.00e+02(3.49e−02)−	1.94e+02(2.53e+01)−	2.01e+02(2.06e−01)−	1.94e+02(2.53e+01)−	**1.76e+02(1.82e+01)**
F27	1.79e+03(2.24e+02)−	2.81e+03(1.63e+02)−	1.13e+03(1.68e+02)−	3.03e+02(7.04e+00)+	1.27e+03(2.41e+02)−	**9.30e+02(8.70e+01)**
F28	**1.92e+03(4.97e+02)+**	3.78e+03(6.47e+02)−	3.02e+03(6.59e+02)−	2.22e+03(6.18e+01)=	7.19e+03(2.68e+03)−	2.34e+03(2.66e+02)
F29	1.42e+03(9.99e+02)−	1.59e+03(3.05e+02)−	1.51e+03(1.44e+02)−	2.80e+03(5.37e+02)−	1.53e+03(2.41e+02)−	**1.33e+03(1.79e+02)**
F30	**3.51e+03(1.96e+03)+**	1.04e+04(3.18e+03)−	8.95e+03(9.06e+02)−	9.51e+03(1.83e+03)−	8.28e+03(1.44e+02)−	7.78e+03(1.08e+03)
+/=/−	**8/5/17**	**0/5/25**	**3/9/18**	**8/4/18**	**1/5/24**	

"−", "+", and "=" respectively denote that the performance of the corresponding algorithm is worse than, better than, and similar to that of HMJCDE according to the Wilcoxon's rank test at a 0.05 significance level.

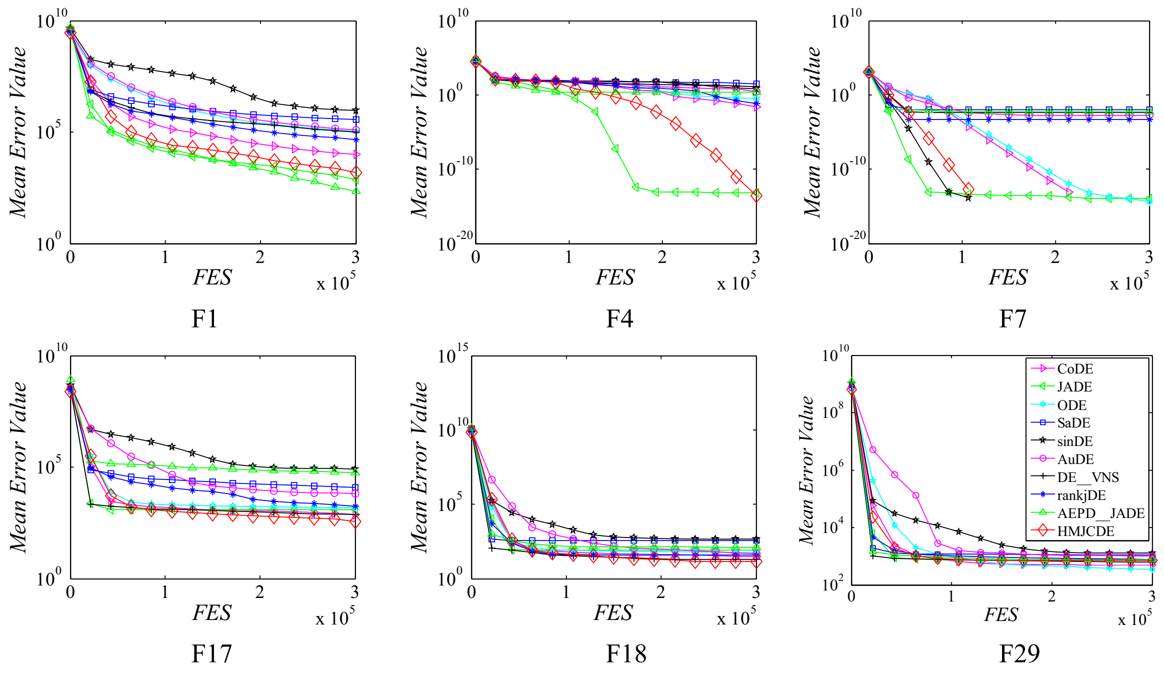

Fig. 10. Evolution of the mean function error values.

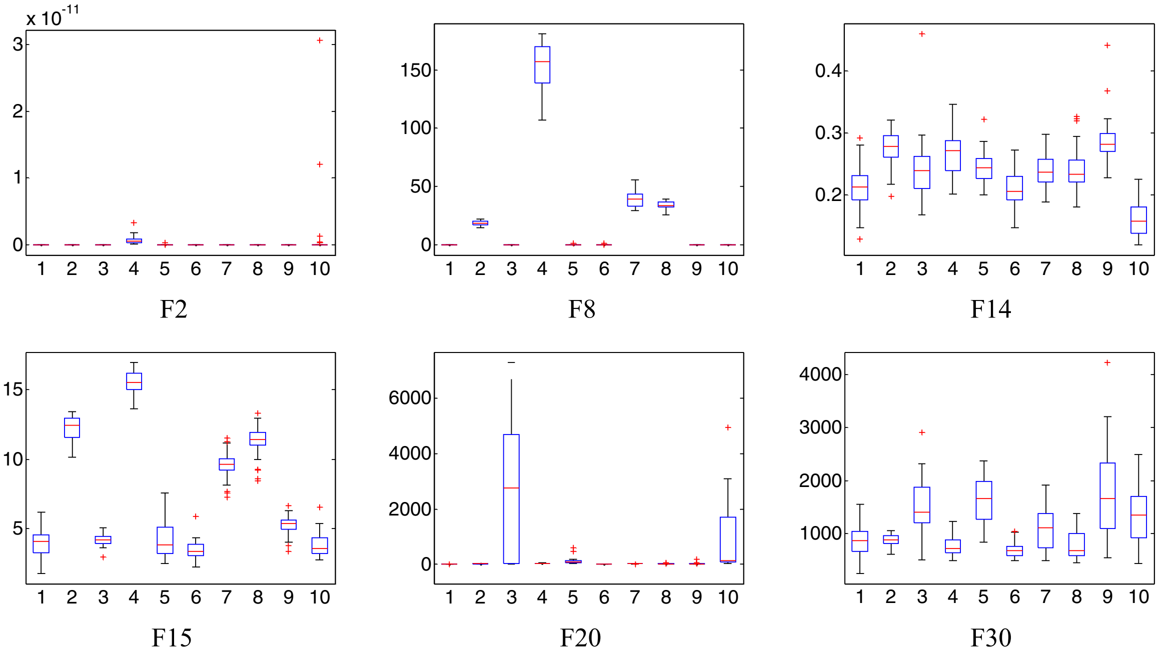

Fig. 11. Box plot of 30 function error values of each algorithm.

DE-VNS, rank-jDE, SinDE and AuDE on 5, 6, 5, 1, 8, 0, 3, 8 and 1 test functions respectively.

Overall, with the increase of the dimensions in search space, the optimization ability of HMJCDE will deteriorate gradually. Nevertheless, the superiority of HMJCDE is not affected by the growth of the dimensions in search space, as HMJCDE is better than or at least similar to all the competitors on unimodal, simple multimodal, and hybrid functions, and slightly worse than CoDE and ODE on composition functions regardless of the dimension is 30, 50, or 100.

5.3.4. *Evolutionary curves for all the algorithms*

The evolutionary curves of the *mean function error values* derived from CoDE, JADE, ODE, SaDE, AEPDE-JADE, DE-VNS, rank-jDE, sinDE, AuDE, and HMJCDE versus the number of *FES* are plotted in Fig. 10 and the box plots of 30 function error values of each algorithm are plotted in Fig. 11 for some typical test functions with 30*D*. In Fig. 11, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 denote HMJCDE, CoDE, JADE, ODE, SaDE, sinDE, AuDE, DE-VNS, rank-jDE and AEPD-JADE respectively. Obviously, as shown in Figs. 10 and 11, HMJCDE demonstrates the

Table 11
The parameter settings of all the non-DE methods.

Algorithms	Parameter settings
TLBO	$Pn = 100$
ABC	$SN = 50$, $limit = 200$
EHS	$HMS = 50$, $HMCR = 0.99$, $PAR = 0.33$, $k = 1.17$
CLPSO	$NP = 40$, $c = 1.49445$, $w_0 = 0.9$, $w_1 = 0.4$
CMA-ES	$NP = D$, $\sigma = 0.25$, $\sigma_{\min} = 1e-15$, $\sigma_{\max} = (X^{\max} - X^{\min})/\sqrt{D}$

better performance on accuracy, convergence rate and robustness than most of compared algorithms.

5.4. Comparison with non-DE algorithms

In this subsection, HMJCDE is further compared with five non-DE algorithms, namely, EHS [10], CLPSO [3], TLBO [24,62], ABC [6,7], and CMA-ES [25]. In EHS, a simple mathematical analysis on the explorative search behaviour of harmony search (HS) was performed, and thus a simple but very useful modification was added into the classical HS to form EHS. For CLPSO, a particle uses the personal historical best information of all the particles to update its velocity. Artificial bee colony (ABC) was proposed in [6], which simulates the foraging behaviour of honeybee swarms. The Teaching-Learning-based Optimization (TLBO) algorithm was presented in [24], which is based on the influence of a teacher on the output of learners. CMA-ES [25] is a very efficient and famous evolution strategy. In our experiments, the parameters of EHS [10], CLPSO [3] and CMA-ES [25] are set as suggested in their original papers. Note that we only use the original ABC and TLBO algorithms for comparison. The source code of ABC could be downloaded from the Homepage of Artificial Bee Colony (ABC) Algorithm (http://mf.erciyes.edu.tr/abc/index.htm) and the source code of TLBO can be obtained from literature [62]. Moreover, all the parameter settings of these non-DE algorithms are given in Table 11. The max_FES for all these algorithms is set to 300,000, and each algorithm is run 30 times for each test function.

Table 12 gives the experimental results obtained by HMJCDE, EHS, CLPSO, TLBO, ABC and CMA-ES. Overall, HMJCDE significantly outperforms EHS, CLPSO, TLBO, ABC and CMA-ES. More specially, HMJCDE performs better than EHS, CLPSO, TLBO, ABC and CMA-ES on 24, 26, 27, 24, and 23 out of 30 test functions respectively, while HMJCDE is only beaten by EHS, CLPSO, TLBO, ABC and CMA-ES on 3, 0, 1, 2 and 2 out of 30 test functions respectively.

At last, to summarize all the above comparisons, all the statistical comparison results are collected in Table 13. When compared to the other algorithms in Table 13, HMJCDE performs better on most of test functions.

5.5. Sensitivity analysis of parameters ε, Q1, Q2 and m

As introduced in Subsection 4.3, three important parameters, *i.e.*, ε, Q1 and Q2, and m, are used in HMJCDE. In order to study their effects on the performance of HMJCDE, some experimental studies are conducted as follows.

When analyzing the influence of ε, Q1, Q2 and m are fixed to 10, 5 and 30 respectively, and six different values are assigned to ε, namely, 0.001, 0.005, 0.01, 0.05, 0.1, and 0.5. 30 independently runs are conducted on each test function with 30D for each value of ε. The experimental results are not shown as different ε values do not bring great influence on the performance of HMJCDE. The final statistical comparison of the results according to Wilcoxon's rank-sum test at 5% significant level is listed in Table 14. It is noted that all the other values of ε are compared with $\varepsilon = 0.05$ in Table 14, and it is reasonable to conclude that the performance of HMJCDE is insensitive to the setting of ε on most of test functions.

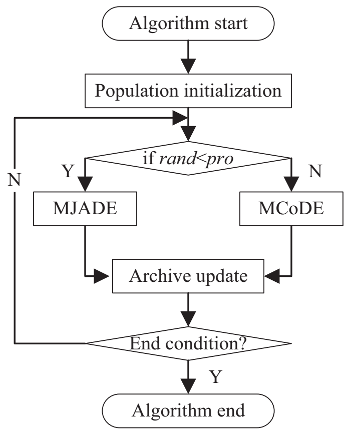

Fig. 12. The flowchart of HMJCDE-I.

On the other hand, ε and m is fixed to 0.05 and 30 respectively when analyzing the impacts of $Q1$ and $Q2$. 11 combinations of $Q1$ and $Q2$ are used in our experimental study, which are shown in Table 15. Note that $Q1 = 10$ and $Q2 = 5$ are employed in the above parameter setting of HMJCDE. The reason for this setting is that MJADE only employs one trial vector generation strategy for each target vector, while MCoDE employs three trial vector generation strategies for each target vector. Thus, the value of $Q1$ is set larger than $Q2$. Each combination is run independently 30 times for each test function with 30D, and the final comparison results are shown in Table 15, which are examined by the Wilcoxon's rank-sum test at 5% significant level to test their significant difference. Note that in Table 15, all other combinations of $Q1$ and $Q2$ are compared with $Q1 = 10$ and $Q2 = 5$. Obviously, when $Q1 = Q2$ (*e.g.*, a small value $Q1 = Q2 = 1$ or a large value $Q1 = Q2 = 20$), the performance will be affected on a small part of test functions. When $Q1$ and $Q2$ are assigned a large value with $Q1 = 2Q2$ (e.g., $Q1 = 40$ and $Q2 = 20$), the optimization performance will be decreased on 5 out of 30 test functions. Nevertheless, the performance of HMJCDE is also not very sensitive to the settings of $Q1$ and $Q2$ on most of test functions.

In order to analyze the effect of parameter m, ε, $Q1$ and $Q2$ are fixedly set to 0.05, 10 and 5 respectively according to the above analysis. Five different representative values are assigned to m, namely, 5, 15, 25, 30 and 45. 30 independently runs are conducted on each test function with 30D for each value of m. The experimental results are collected in Table 16.

From Table 16, considering unimodal functions F1-F3, m is insensitive to HMJCDE on F2 and F3, but a small value (*e.g.*, 5, 15, 25) of m may degrade the performance of F1. Regarding simple multimodal functions F4-F16, in most cases (*i.e.*, F4, F6, F7-F11), a large value of m (*e.g.*, 25, 30) obtains the better performance, while a small value of m (*e.g.*, 5, 15) will significantly degrade the performance. However, on F5 and F12, a smaller value of m can obtain the better performance. For the remainder test functions (F13-F16), relatively speaking, m will not significantly affect the performance of HMJCDE. Considering hybrid function F17-F22, a smaller value of m (*e.g.*, 5, 15) shows the better performance on F18, while a larger value of m (*e.g.*, 30, 45) will obtain the better performance on F21 and F22. In addition, HMJCDE gets the best results on F17, F19, F20 when $m = 25$. Regarding composition functions F23-F30, HMJCDE gets the best result on F29 when $m = 5$, while it obtains the better results on F30 and F28 when $m = 30$ and $m = 45$ respectively. For the other test functions, HMJCDE is not significantly influenced by the values of m. Overall, these experimental results show that the selection of m has a significant effect on the optimization performance. Generally speaking, a small value of m (*e.g.*, 5) may easily cause premature convergence on unimodal, simple multimodal, and hybrid

Table 12
Comparisons of HMJCDE and five non-DE algorithms on 30 test functions with 30*D*.

Alg Fuc	EHS Mean error (std dev)	CLPSO Mean error (std dev)	TLBO Mean error (std dev)	ABC Mean error (std dev)	CMA-ES Mean error (std dev)	HMJCDE Mean error (std dev)
F1	5.07e + 06(3.08e + 06)–	8.54e + 06(1.98e + 06)–	5.15e + 05(1.30e + 05)–	7.51e + 06(3.36e + 06)–	**2.75e – 14(1.05e – 14)+**	1.51e + 03(2.53e + 03)
F2	1.10e + 04(6.99e + 03)–	1.17e + 02(3.49e + 02)–	9.19e + 06(1.32e + 06)–	1.65e + 02(2.12e + 02)–	5.49e – 14(2.35e – 14)–	**0.00e + 00(0.00e + 00)**
F3	1.22e + 04(1.07e + 04)–	1.97e + 02(2.34e + 02)–	1.58e + 01(3.20e + 00)–	6.30e + 02(5.00e + 02)–	9.66e – 14(4.76e – 14)–	**0.00e + 00(0.00e + 00)**
F4	1.03e + 02(2.25e + 01)–	7.18e + 01(1.92e + 01)–	1.74e + 02(1.93e + 01)–	3.55e + 01(2.69e + 01)–	1.12e – 13(3.80e – 14)=	**3.03e – 14(4.89e – 14)**
F5	2.09e + 01(3.98e – 02)–	2.04e + 01(4.82e – 02)–	2.08e + 01(5.33e – 02)–	2.03e + 01(4.39e – 02)–	2.00e + 01(1.15e – 05)=	**2.00e + 01(2.66e – 02)**
F6	2.21e – 01(5.47e – 01)+	1.26e + 01(1.38e + 00)–	2.48e + 01(1.53e + 00)–	1.46e + 01(1.44e + 00)–	4.38e + 01(8.65e + 00)–	**2.38e + 00(2.92e + 00)**
F7	5.68e – 04(9.75e – 04)–	7.06e – 06(9.45e – 06)–	1.09e + 00(1.72e – 02)–	1.68e – 05(1.75e – 02)–	1.48e – 03(3.01e – 03)–	**0.00e + 00(0.00e + 00)**
F8	5.66e – 04(2.36e – 03)–	1.06e – 13(2.88e – 14)–	1.54e + 02(3.62e + 01)–	2.39e – 13(8.63e – 14)–	4.39e + 02(8.78e + 01)–	**0.00e + 00(0.00e + 00)**
F9	1.50e + 02(8.59e + 00)–	4.90e + 01(6.72e + 00)–	1.98e + 02(1.10e + 01)–	8.55e + 01(1.13e + 01)–	6.20e + 02(1.63e + 02)–	**3.92e + 01(1.65e + 00)**
F10	1.90e + 02(9.79e + 01)–	3.33e + 00(1.42e + 00)–	5.08e + 03(2.79e + 02)–	1.46e + 00(8.14e – 01)–	5.01e + 03(8.13e + 02)–	**1.51e – 01(2.85e – 02)**
F11	6.46e + 03(2.59e + 02)–	2.26e + 03(2.40e + 02)=	4.89e + 03(2.92e + 02)–	**1.98e + 03(2.76e + 02)+**	5.14e + 03(7.68e + 02)–	2.28e + 03(4.43e + 02)
F12	2.36e + 00(2.88e – 01)–	3.90e – 01(6.40e – 02)–	1.93e + 00(1.49e – 01)–	3.07e – 01(3.89e – 02)=	2.56e – 01(2.22e – 01)+	3.23e – 01(9.02e – 02)
F13	**2.19e – 01(3.29e – 02)+**	3.20e – 01(3.92e – 02)–	5.14e – 01(3.42e – 02)–	2.38e – 01(2.60e – 02)=	2.53e – 01(6.55e – 02)–	2.46e – 01(4.22e – 02)
F14	2.82e – 01(3.86e – 02)–	2.56e – 01(2.66e – 02)–	2.02e – 01(3.74e – 02)+	**1.92e – 01(1.51e – 02)+**	3.75e – 01(1.06e – 01)–	2.10e – 01(3.71e – 02)
F15	1.40e + 01(8.19e – 01)–	7.22e + 00(8.95e – 01)–	2.98e + 01(5.01e + 00)–	8.91e + 00(1.44e + 00)–	**3.56e + 00(7.18e – 01)=**	3.97e + 00(1.02e + 00)
F16	1.12e + 01(3.53e – 01)–	1.04e + 01(3.61e – 01)–	1.23e + 01(4.31e – 01)–	1.01e + 01(3.45e – 01)–	1.42e + 01(5.63e – 01)–	**9.93e + 00(5.27e – 01)**
F17	6.99e + 05(3.30e + 05)–	9.74e + 05(4.17e + 05)–	1.72e + 03(1.18e + 02)–	2.92e + 06(1.40e + 06)–	1.72e + 03(3.32e + 02)–	**3.63e + 02(2.37e + 02)**
F18	5.99e + 02(7.58e + 02)–	9.07e + 01(3.25e + 01)–	1.03e + 02(1.29e + 01)–	2.54e + 03(1.44e + 03)–	1.53e + 02(4.82e + 01)–	**1.40e + 01(6.79e + 00)**
F19	7.79e + 00(1.03e + 01)–	7.49e + 00(5.75e – 01)–	1.06e + 01(5.74e – 01)–	7.43e + 00(6.00e – 01)–	9.50e + 00(1.66e + 00)–	**3.84e + 00(7.75e – 01)**
F20	1.78e + 03(1.36e + 03)–	2.97e + 03(1.88e + 03)–	8.32e + 01(7.23e + 00)–	5.41e + 03(2.10e + 03)–	2.46e + 02(9.07e + 01)–	**1.01e + 01(3.15e + 00)**
F21	2.35e + 05(2.26e + 05)–	9.67e + 04(5.71e + 04)–	1.14e + 03(1.53e + 02)–	3.04e + 05(1.23e + 05)–	1.06e + 03(2.97e + 01)–	**1.88e + 02(1.31e + 02)**
F22	1.01e + 02(5.79e + 01)–	1.93e + 02(7.72e + 01)–	2.88e + 02(6.32e + 01)–	2.72e + 02(7.85e + 01)–	3.38e + 02(2.71e + 02)–	**8.25e + 01(7.95e + 01)**
F23	3.16e + 02(4.63e – 01)=	3.15e + 02(7.73e – 06)=	3.16e + 02(2.35e – 02)=	3.15e + 02(1.02e – 01)=	3.15e + 02(2.97e – 12)=	**3.15e + 02(2.14e – 13)**
F24	2.25e + 02(1.83e + 00)–	**2.24e + 02(3.07e + 00)=**	2.50e + 02(2.68e + 00)–	2.27e + 02(1.27e + 00)–	3.47e + 02(4.09e + 02)–	**2.24e + 02(2.34e + 00)**
F25	2.06e + 02(1.10e + 00)–	2.08e + 02(1.09e + 00)–	2.06e + 02(4.08e – 01)–	2.08e + 02(1.31e + 00)–	2.05e + 02(3.25e + 00)–	**2.03e + 02(4.95e – 01)**
F26	1.06e + 02(2.09e + 01)=	1.00e + 02(1.17e – 01)=	1.00e + 02(4.48e – 02)=	1.00e + 02(5.86e – 02)=	1.10e + 02(3.04e + 01)–	**1.00e + 02(3.74e – 02)**
F27	**3.14e + 02(2.47e + 01)+**	4.13e + 02(5.36e + 00)–	4.09e + 02(3.29e + 00)–	4.10e + 02(2.96e + 00)–	4.03e + 02(8.95e + 01)=	3.90e + 02(3.06e + 01)
F28	**8.29e + 02(2.96e + 01)=**	9.00e + 02(3.21e + 01)–	1.73e + 03(2.08e + 02)–	9.71e + 02(5.69e + 01)–	4.78e + 03(3.24e + 03)–	8.32e + 02(3.66e + 01)
F29	1.19e + 03 (1.04e + 02)–	1.00e + 03(1.27e + 02)–	4.69e + 03(2.31e + 03)–	1.01e + 03(6.55e + 01)–	7.97e + 02(7.31e + 01)–	**7.22e + 02(2.59e + 00)**
F30	3.25e + 03(6.89e + 02)–	3.79e + 03(1.01e + 03)–	2.54e + 03(4.49e + 02)–	3.48e + 03(7.46e + 02)–	2.27e + 03(7.00e + 02)–	**8.75e + 02(3.46e + 02)**
+/=/–	**3/3/24**	**0/4/26**	**1/2/27**	**2/4/24**	**2/5/23**	

"–", "+", and "=" respectively denote that the performance of the corresponding algorithm is worse than, better than, and similar to that of HMJCDE according to the Wilcoxon's rank test at a 0.05 significance level.

Table 13
Summary of all the comparison results.

D = 30										
HMJCDE	CoDE	JADE	ODE	SaDE	AEPD-JADE	DE-VNS	rank-jDE	sinDE	AuDE	
+/=/–	3/7/20	4/8/18	5/6/19	0/8/22	5/11/14	3/7/20	1/16/13	9/10/11	0/6/24	
HMJCDE	EHS	CLPSO	TLBO	ABC	CMA-ES					
+/=/–	3/3/24	0/4/26	1/2/27	2/4/24	2/5/23					
D = 50										
HMJCDE	CoDE	JADE	ODE	SaDE	AEPD-JADE	DE-VNS	rank-jDE	sinDE	AuDE	
+/=/–	6/7/17	4/10/16	6/4/20	0/1/29	8/5/17	0/3/27	2/13/15	9/7/14	0/4/26	
D = 100										
HMJCDE	CoDE	JADE	ODE	SaDE	AEPD-JADE	DE-VNS	rank-jDE	sinDE	AuDE	
+/=/–	5/7/18	6/9/15	5/3/22	1/4/25	8/5/17	0/5/25	3/9/18	8/4/18	1/5/24	

"–", "+", and "=" respectively denote that the performance of the corresponding algorithm is worse than, better than, and similar to that of HMJCDE according to the Wilcoxon's rank test at a 0.05 significance level.

Table 14
Comparisons of $\varepsilon = 0.05$ with other values.

ε	$\varepsilon = 0.001$	$\varepsilon = 0.005$	$\varepsilon = 0.01$	$\varepsilon = 0.1$	$\varepsilon = 0.1$	$\varepsilon = 0.05$
+	1	0	0	0	0	
=	28	29	26	28	28	
–	1	1	4	2	2	

"–", "+", and "=" respectively denote that the performance of the corresponding algorithm is worse than, better than, and similar to that of HMJCDE ($\varepsilon = 0.05$) according to the Wilcoxon's rank test at a 0.05 significance level.

Table 15
Comparisons of Q1 = 10, Q2 = 5 with other values.

Q1,Q2	1,1	3,3	5,5	10,10	20,20	10,5
+	2	2	1	2	0	
=	21	23	26	24	23	
–	7	5	3	4	7	
Q1,Q2	2,1	6,3	16,8	20,10	40,20	10,5
+	2	0	0	0	0	
=	26	29	29	29	25	
–	2	1	1	1	5	

"–", "+", and "=" respectively denote that the performance of the corresponding algorithm is worse than, better than, and similar to that of HMJCDE (*Q*1 = 10, *Q*2 = 5) according to the Wilcoxon's rank test at a 0.05 significance level.

Table 16
Results of HMJCDE with different values of m on 30 test functions with 30*D*.

Alg Fuc	HMJCDE (*m* = 5) Mean error (std dev)	HMJCDE (*m* = 15) Mean error (std dev)	HMJCDE (*m* = 25) Mean error (std dev)	HMJCDE (*m* = 30) Mean error (std dev)	HMJCDE (*m* = 45) Mean error (std dev)
F1	6.74e+03(6.69e+03)	4.17e+03(4.61e+03)	3.54e+03(4.14e+03)	**1.51e+03(2.53e+03)**	2.36e+03(3.99e+03)
F2	**0.00e+00(0.00e+00)**	**0.00e+00(0.00e+00)**	**0.00e+00(0.00e+00)**	**0.00e+00(0.00e+00)**	**0.00e+00(0.00e+00)**
F3	**0.00e+00(0.00e+00)**	**0.00e+00(0.00e+00)**	**0.00e+00(0.00e+00)**	**0.00e+00(0.00e+00)**	**0.00e+00(0.00e+00)**
F4	1.25e−09(6.51e−09)	2.84e−14(3.58e−14)	1.74e−14(3.39e−14)	3.03e−14(4.89e−14)	**1.33e−14(2.45e−14)**
F5	**2.00e+01(4.64e−03)**	**2.00e+01(1.11e−02)**	**2.00e+00(1.58e−02)**	**2.00e+01(2.66e−02)**	2.01e+01(2.68e−02)
F6	1.02e+01(2.83e+00)	4.84e+00(3.73e+00)	2.20e+00(2.34e+00)	2.38e+00(2.92e+00)	**1.31e+00(2.17e+00)**
F7	5.75e−04(3.61e−02)	**0.00e+00(0.00e+00)**	**0.00e+00(0.00e+00)**	**0.00e+00(0.00e+00)**	**0.00e+00(0.00e+00)**
F8	1.40e+01(9.62e−01)	1.63e+00(1.44e+00)	9.95e−02(3.04e−01)	**0.00e+00(0.00e+00)**	**0.00e+00(0.00e+00)**
F9	5.37e+01(7.60e+01)	5.32e+01(1.20e+01)	4.13e+01(1.05e+01)	3.92e+01(1.65e+00)	**3.59e+01(8.47e+00)**
F10	2.13e+02(1.67e+02)	1.98e+01(4.34e+01)	6.22e−01(1.09e+00)	1.51e−01(2.85e−02)	2.06e−01(5.48e−02)
F11	2.95e+03(5.48e+02)	2.42e+03(5.11e+02)	2.29e+03(6.43e+02)	**2.28e+03(4.43e+02)**	2.66e+03(3.75e+02)
F12	**4.33e−02(3.34e−02)**	1.82e−01(1.52e−01)	3.39e−01(8.54e−02)	3.23e−01(9.02e−02)	3.68e−01(8.72e−02)
F13	2.47e−01(3.90e−02)	2.42e−01(3.68e−02)	2.37e−01(4.38e−02)	2.46e−01(4.22e−02)	**2.35e−01(4.07e−02)**
F14	2.31e−01(3.61e−02)	2.18e−01(3.49e−02)	2.34e−01(2.76e−02)	**2.10e−01(3.71e−02)**	2.19e−01(2.46e−02)
F15	4.11e+00(9.62e−01)	**3.73e+00(9.95e−01)**	3.95e+00(1.40e+00)	3.97e+00(1.02e+00)	4.81e+00(1.06e+00)
F16	1.07e+01(7.60e−01)	1.03e+01(7.98e−01)	1.02e+01(7.92e−01)	**9.93e+00(5.27e−01)**	9.99e+00(4.11e−01)
F17	5.30e+02(3.61e+02)	4.17e+02(2.71e+02)	**3.47e+02(1.90e+02)**	3.63e+02(2.37e+02)	5.04e+02(3.02e+02)
F18	**1.20e+01(4.86e+00)**	**1.20e+01(4.98e+00)**	1.53e+01(6.16e+00)	1.40e+01(6.79e+00)	1.34e+01(7.24e+00)
F19	4.06e+00(9.79e−01)	3.58e+00(7.98e−01)	**3.63e+00(9.46e−01)**	3.84e+00(7.75e−01)	3.97e+00(5.82e−01)
F20	1.10e+01(4.54e+00)	9.68e+00(5.16e+00)	**9.30e+00(3.03e+00)**	1.01e+01(3.15e+00)	9.86e+00(3.52e+00)
F21	3.53e+02(2.33e+02)	2.19e+02(1.43e+02)	2.30e+02(1.22e+02)	**1.88e+02(1.31e+02)**	2.00e+02(1.25e+02)
F22	3.32e+02(1.67e+02)	2.53e+02(1.71e+02)	1.26e+02(9.29e+01)	8.25e+01(7.95e+01)	**6.42e+01(5.09e+01)**
F23	**3.15e+02(1.57e−13)**	**3.15e+02(1.57e−13)**	**3.15e+02(2.02e−13)**	**3.15e+02(2.14e−13)**	**3.15e+02(2.19e−13)**
F24	2.28e+02(4.02e+00)	2.25e+02(9.93e−01)	2.24e+02(8.39e−01)	2.24e+02(2.34e+00)	**2.23e+02(4.51e+00)**
F25	2.04e+02(2.32e+00)	2.03e+02(1.49e+00)	**2.03e+02(4.66e−01)**	**2.03e+02(4.95e−01)**	**2.03e+02(5.39e−01)**
F26	**1.00e+02(4.48e−02)**	**1.00e+02(4.95e−02)**	**1.00e+02(4.82e−02)**	**1.00e+02(3.74e−02)**	**1.00e+02(4.02e−02)**
F27	4.00e+02(3.38e+00)	3.90e+02(3.11e+01)	**3.72e+02(4.54e+01)**	3.90e+02(3.06e+01)	3.92e+02(2.74e+01)
F28	8.93e+02(3.69e+01)	8.64e+02(3.90e+01)	8.53e+02(2.88e−01)	8.32e+02(3.66e+01)	**7.91e+02(2.51e+01)**
F29	**6.69e+02(1.58e+02)**	7.44e+02(5.28e+01)	7.19e+02(5.89e+00)	7.22e+02(2.59e+00)	7.18e+02(3.99e+00)
F30	1.86e+03(6.53e+02)	1.53e+03(5.92e+02)	1.24e+03(4.04e+02)	**8.75e+02(3.46e+02)**	8.90e+02(2.81e+02)

Table 17
Comparisons of HMJCDE with HMJCDE-I with different pro values.

Alg Fuc	*pro* = 0.1 Mean error (std dev)	*pro* = 0.3 Mean error (std dev)	*pro* = 0.5 Mean error (std dev)	*pro* = 0.7 Mean error (std dev)	*pro* = 0.9 Mean error (std dev)	HMJCDE Mean error (std dev)
F1	**8.60e+02(1.39e+03)+**	1.60e+03(2.50e+03)=	2.87e+03(3.45e+03)−	1.20e+03(2.27e+03)=	1.30e+03(1.66e+03)=	1.51e+03(2.53e+03)
F2	8.53e−15(1.32e−14)=	**0.00e+00(0.00e+00)=**	**0.00e+00(0.00e+00)=**	**0.00e+00(0.00e+00)=**	**0.00e+00(0.00e+00)=**	**0.00e+00(0.00e+00)**
F3	**0.00e+00(0.00e+00)=**	**0.00e+00(0.00e+00)=**	**0.00e+00(0.00e+00)=**	**0.00e+00(0.00e+00)=**	1.14e−14(2.31e−14)=	**0.00e+00(0.00e+00)**
F4	2.01e−02(3.98e−02)−	4.69e−04(1.67e−03)−	2.19e−10(1.08e−09)−	**1.89e−14(1.04e−14)=**	**1.71e−14(3.04e−14)=**	**3.03e−14(4.89e−14)**
F5	2.04e+01(6.38e−02)−	2.02e+01(6.91e−02)−	2.01e+01(3.63e−02)=	2.01e+01(2.38e−02)=	**2.00e+01(1.63e−02)=**	**2.00e+01(2.66e−02)**
F6	2.00e+00(2.27e+00)+	2.34e+00(2.08e+00)=	1.68e+00(1.99e+00)+	**1.98e+00(1.95e+00)+**	2.16e+00(2.72e+00)+	2.38e+00(2.92e+00)
F7	**0.00e+00(0.00+00)=**	**0.00e+00(0.00e+00)=**	**0.00e+00(0.00e+00)=**	**0.00e+00(0.00e+00)=**	**0.00e+00(0.00e+00)=**	**0.00e+00(0.00e+00)**
F8	1.93e−01(1.08e+01)−	5.64e−01(7.70e−01)−	2.32e−01(4.28e−01)−	3.32e−02(1.82e−01)−	**0.00e+00(0.00e+00)=**	**0.00e+00(0.00e+00)**
F9	4.80e+01(1.73e+01)−	4.40e+01(1.41e+01)−	4.28e+01(1.40e+01)−	4.07e+01(1.61e+01)−	**3.62e+01(1.25e+01)=**	3.92e+01(1.65e+00)
F10	3.41e+00(2.70e+00)−	3.81e+00(4.40e+00)−	2.21e+00(3.86e+00)−	2.62e−01(4.40e−01)−	**6.57e−02(2.20e−01)+**	1.51e−01(2.85e−02)
F11	3.71e+03(3.53e+02)−	3.02e+03(6.78e+02)−	2.50e+03(8.64e−01)−	2.34e+03(4.11e+02)=	**2.19e+03(4.17e+02)=**	2.28e+03(4.43e+02)
F12	6.10e−01(3.98e−02)−	4.40e−01(1.59e−01)−	3.63e−01(8.60e−02)−	2.98e−01(7.73e−02)+	**2.90e−01(8.52e−02)=**	3.23e−01(9.02e−02)
F13	2.78e−01(4.49e−02)−	2.60e−01(2.68e−02)−	2.36e−01(4.11e−02)−	2.32e−01(3.59e−02)=	**2.32e−01(3.06e−02)=**	2.46e−01(4.22e−02)
F14	2.44e−01(3.70e−02)−	2.38e−01(2.58e−02)−	2.32e−01(3.23e−02)−	2.23e−01(3.44e−02)−	2.26e−01(3.71e−02)−	**2.10e−01(3.71e−02)**
F15	7.52e+00(1.36e+00)−	5.19e+00(1.64e+00)−	4.00e+00(2.56e−13)=	4.41e+00(1.13e+00)+	**3.89e+00(8.77e−01)=**	3.97e+00(1.02e+00)
F16	1.08e+01(3.18e−01)−	1.04e+01(5.37e−01)−	1.01e+01(5.21e−01)=	9.99e+00(4.13e−01)=	**9.65e+00(6.48e−01)=**	9.93e+00(5.27e−01)
F17	3.51e+02(1.79e+02)+	2.74e+02(1.77e+02)+	**2.41e+02(1.65e+02)+**	5.16e+02(2.27e+02)−	8.88e+02(3.41e+02)−	3.63e+02(2.37e+02)
F18	9.29e+00(5.06e+00)+	**8.67e+00(3.35e+00)+**	9.15e+00(3.89e+00)+	1.68e+01(9.17e+00)−	4.74e+01(2.87e+01)−	1.40e+01(6.79e+00)
F19	3.93e+00(7.15e−01)−	**3.44e+00(6.75e−01)=**	3.68e+00(8.64e−01)=	3.67e+00(8.28e−01)=	4.24e+00(7.00e−01)−	3.84e+00(7.75e−01)
F20	9.09e+00(3.03e+00)=	**8.68e+00(2.47e−00)+**	9.04e+00(2.36e+00)=	1.01e+01(2.71e+00)=	1.44e+01(3.63e+00)−	1.01e+01(3.15e+00)
F21	2.19e+02(1.34e+02)−	1.54e+02(1.17e+02)+	**1.69e+02(1.38e+02)+**	2.35e+02(4.66e+01)−	3.30e+02(1.38e+02)−	1.88e+02(1.31e+02)
F22	9.15e+01(6.11e+01)−	1.11e+02(9.76e+01)−	9.12e+01(8.11e+01)−	1.30e+02(7.92e+01)−	1.60e+02(9.74e+01)−	**8.25e+01(7.95e+01)**
F23	**3.15e+02(1.11e−13)=**	**3.15e+02(1.73e−13)=**	**3.15e+02(2.56e−13)=**	**3.15e+02(1.96e−13)=**	**3.15e+02(2.15e−13)=**	**3.15e+02(2.14e−13)**
F24	**2.23e+02(8.42e−01)=**	2.24e+02(8.86e−01)=	2.24e+02(7.66e−01)=	2.24e+02(7.11e−01)=	2.24e+02(2.01e+00)=	2.24e+02(2.34e+00)
F25	**2.03e+02(1.70e−01)=**	**2.03e+02(3.04e−01)=**	**2.03e+02(4.46e−01)=**	**2.03e+02(4.76e−01)=**	**2.03e+02(6.73e−01)=**	**2.03e+02(4.95e−01)**
F26	**1.00e+02(4.32e−02)=**	**1.00e+02(4.36e−02)=**	**1.00e+02(3.86e−02)=**	**1.00e+02(3.60e−02)=**	**1.00e+02(2.84e−02)=**	**1.00e+02(3.74e−02)**
F27	3.97e+02(1.83e+01)−	3.98e+02(1.15e+01)−	3.80e+02(4.09e+01)=	3.80e+02(4.09e+01)=	**3.55e+02(5.00e+01)+**	3.90e+02(3.06e+01)
F28	8.27e+02(3.64e+01)=	8.31e+02(4.08e+01)=	8.36e+02(3.57e+01)=	8.25e+02(4.66e+01)=	**8.05e+02(6.76e+01)=**	8.32e+02(3.66e+01)
F29	**6.95e+02(7.29e+00)+**	7.16e+02(1.31e+00)=	7.16e+02(1.55e+00)=	7.02e+02(8.85e+01)=	7.47e+02(4.45e+01)=	7.22e+02(2.59e+00)
F30	**6.08e+02(3.52e+02)+**	7.38e+02(2.32e+02)+	8.45e+02(2.82e+02)=	1.14e+03(4.96e+02)−	1.41e+03(4.73e+02)−	8.75e+02(3.46e+02)
+/=/−	**5/10/15**	**5/12/13**	**4/16/10**	**3/18/9**	**3/19/8**	

"−", "+", and "=" respectively denote that the performance of the corresponding algorithm is worse than, better than, and similar to that of HMJCDE according to the Wilcoxon's rank test at a 0.05 significance level.

Table 18
Comparisons of HMJCDE with HMJCDE-II with different number values.

Alg Fuc	*number* = 1 Mean error (std dev)	*number* = 5 Mean error (std dev)	*number* = 10 Mean error (std dev)	*number* = 15 Mean error (std dev)	*number* = 20 Mean error (std dev)	HMJCDE Mean error (std dev)
F1	3.66e + 03(4.14e + 03)–	5.16e + 03(4.67e + 03)–	6.32e + 03(5.40e + 03)–	4.44e + 03(6.17e + 03)–	**1.31e + 03(2.76e + 03)=**	1.51e + 03(2.53e + 03)
F2	**0.00e + 00(0.00e + 00)=**	9.10e – 11(3.05e – 10)–	1.21e – 13(6.64e – 13)=	1.81e – 13(2.64e – 13)–	1.66e – 13(2.27e – 13)–	**0.00e + 00(0.00e + 00)**
F3	**0.00e + 00(0.00e + 00)=**	2.87e – 10(9.47e – 10)–	4.15e – 09(1.37e – 08)–	**0.00e + 00(0.00e + 00)=**	**0.00e + 00(0.00e + 00)=**	**0.00e + 00(0.00e + 00)**
F4	2.04e – 12(3.42e – 12)–	4.16e – 10(1.20e – 09)–	2.33e – 05(8.60e – 05)–	6.65e – 02(1.22e – 01)–	2.17e + 00(1.16e – 01)–	**3.03e – 14(4.89e – 14)**
F5	2.01e + 00(3.52e + 00)=	2.01e + 01(2.64e – 02)=	2.01e + 01(2.26e – 02)=	**2.00e + 01(2.33e – 02)=**	**2.00e + 01(2.67e – 02)=**	**2.00e + 01(2.66e – 02)**
F6	3.58e + 00(3.52e + 00)=	4.75e + 00(3.33e + 00)–	5.05e + 00(3.77e + 00)–	6.20e + 00(4.06e + 00)–	9.31e + 00(3.49e + 00)–	**2.38e + 00(2.92e + 00)**
F7	**0.00e + 00(0.00e + 00)=**	**0.00e + 00(0.00e + 00)=**	**0.00e + 00(0.00e + 00)=**	**0.00e + 00(0.00e + 00)=**	**0.00e + 00(0.00e + 00)=**	**0.00e + 00(0.00e + 00)**
F8	1.32e – 01(5.68e – 01)=	1.89e – 14(4.31e – 14)–	3.90e – 13(5.86e – 13)–	2.64e – 11(6.01e – 11)–	2.17e – 10(3.32e – 10)–	**0.00e + 00(0.00e + 00)**
F9	4.35e + 01(1.05e + 01)=	4.10e + 01(1.47e + 01)=	3.96e + 01(1.39e + 01)=	4.24e + 01(9.73e + 00)=	4.09e + 01(9.08e + 00)=	**3.92e + 01(1.65e + 00)**
F10	9.61e – 01(1.84e + 00)–	9.02e – 02(2.13e – 01)+	2.43e – 02(2.57e – 02)+	1.59e – 01(5.61e – 02)–	2.10e – 01(5.13e – 02)–	**1.51e – 01(2.85e – 02)**
F11	2.70e + 03(4.60e + 02)–	2.63e + 03(4.75e + 02)–	2.55e + 03(4.98e + 02)–	2.42e + 03(5.20e + 02)=	2.43e + 03(5.57e + 02)=	**2.28e + 03(4.43e + 02)**
F12	3.58e – 01(1.32e – 01)=	**3.10e – 01(9.86e – 02)=**	3.53e – 01(1.06e – 01)=	3.24e – 01(8.42e – 02)=	3.19e – 01(9.78e – 02)=	3.23e – 01(9.02e – 02)
F13	2.45e – 01(3.98e – 02)=	2.45e – 01(3.27e – 02)=	**2.22e – 01(3.57e – 02)+**	2.29e – 01(3.50e – 02)=	2.34e – 01(4.68e – 02)=	2.46e – 01(4.22e – 02)
F14	2.34e – 01(3.30e – 02)–	2.26e – 01(3.24e – 02)=	2.24e – 01(3.39e – 02)=	2.24e – 01(2.74e – 02)=	2.26e – 01(2.57e – 02)=	**2.10e – 01(3.71e – 02)**
F15	4.71e + 00(1.16e + 00)–	4.37e + 00(1.52e + 00)=	4.68e + 00(1.11e + 00)–	4.40e + 00(7.88e – 01)=	4.38e + 00(7.70e – 01)=	**3.97e + 00(1.02e + 00)**
F16	1.00e + 01(4.95e – 01)=	1.01e + 01(4.21e – 01)=	**9.82e + 00(6.00e – 01)=**	1.01e + 01(4.18e – 01)=	1.00e + 01(5.44e – 01)=	9.93e + 00(5.27e – 01)
F17	6.15e + 02(2.63e + 02)–	7.02e + 02(2.81e + 02)–	7.68e + 02(2.07e + 02)–	7.15e + 02(2.25e + 02)–	7.47e + 02(2.01e + 02)–	**3.63e + 02(2.37e + 02)**
F18	1.69e + 01(5.70e + 00)–	2.39e + 01(6.11e + 00)–	2.34e + 01(5.79e + 00)–	2.63e + 01(5.65e + 00)–	2.34e + 01(5.08e + 00)–	**1.40e + 01(6.79e + 00)**
F19	3.85e + 00(8.17e – 01)=	4.13e + 00(6.55e – 01)=	4.15e + 00(5.04e – 01)=	4.37e + 00(7.08e – 01)–	4.51e + 00(7.65e – 01)–	**3.84e + 00(7.75e – 01)**
F20	1.23e + 01(3.09e + 00)–	1.41e + 01(3.97e + 00)–	1.50e + 01(3.34e + 00)–	1.71e + 01(3.22e + 00)–	1.71e + 01(3.65e + 00)–	**1.01e + 01(3.15e + 00)**
F21	2.94e + 02(1.52e + 02)–	3.08e + 02(1.24e + 02)–	3.94e + 02(1.24e + 02)–	4.43e + 02(1.27e + 02)–	4.17e + 02(9.65e + 01)–	**1.88e + 02(1.31e + 02)**
F22	9.89e + 01(9.16e + 01)=	1.25e + 02(1.11e + 02)–	9.67e + 01(7.36e + 01)–	1.26e + 02(8.00e + 01)–	1.33e + 02(9.17e + 01)–	**8.25e + 01(7.95e + 01)**
F23	**3.15e + 02(1.96e – 13)=**	**3.15e + 02(1.57e – 13)=**	**3.15e + 02(3.67e – 12)=**	**3.15e + 02(1.46e – 13)=**	**3.15e + 02(1.11e – 13)=**	**3.15e + 02(2.14e – 13)**
F24	2.24e + 02(2.23e + 00)=	2.24e + 02(1.10e + 00)=	2.24e + 02(9.91e – 01)=	2.24e + 02(8.49e – 01)=	**2.23e + 02(7.83e – 01)=**	2.24e + 02(2.34e + 00)
F25	**2.03e + 02(7.26e – 01)=**	**2.03e + 02(5.00e – 01)=**	**2.03e + 02(5.60e – 01)=**	2.04e + 02(1.32e + 00)=	**2.03e + 02(3.39e – 01)=**	**2.03e + 02(4.95e – 01)**
F26	**1.00e + 02(3.88e – 02)=**	**1.00e + 02(4.56e – 02)=**	**1.00e + 02(2.89e – 02)=**	**1.00e + 02(3.60e – 02)=**	**1.00e + 02(3.52e – 02)=**	**1.00e + 02(3.74e – 02)**
F27	**3.78e + 02(4.14e + 01)=**	3.89e + 02(3.08e + 01)=	3.87e + 02(3.37e + 01)=	3.94e + 02(2.39e + 01)=	4.00e + 02(2.00e – 01)=	3.90e + 02(3.06e + 01)
F28	8.41e + 02(3.62e + 01)=	8.52e + 02(4.08e + 01)=	8.16e + 02(3.99e + 01)=	**8.04e + 02(2.75e + 01)+**	8.06e + 02(2.32e + 01)=	8.32e + 02(3.66e + 01)
F29	7.24e + 02(2.58e + 01)=	7.19e + 02(8.24e + 00)=	1.04e + 03(2.08e + 02)–	7.24e + 02(2.06e + 01)=	**6.86e + 02(1.24e + 02)=**	7.22e + 02(2.59e + 00)
F30	1.13e + 03(4.14e + 02)–	1.34e + 03(5.82e + 02)–	1.27e + 03(4.55e + 02)–	1.00e + 03(2.32e + 02)=	8.77e + 02(1.86e + 02)=	**8.75e + 02(3.46e + 02)**
+/=/–	**0/19/11**	**1/16/13**	**1/14/15**	**1/17/12**	**0/19/11**	

"–", "+", and "=" respectively denote that the performance of the corresponding algorithm is worse than, better than, and similar to that of HMJCDE according to the Wilcoxon's rank test at a 0.05 significance level.

functions. However, in some special cases, such as F5, F12 and F18, a small value of *m* can give the better results. When considering all the test problems, *m* is suggested to set in [25,45].

5.6. The performance analysis of our hybrid framework

In this subsection, in order to verify the effectiveness of our proposed hybrid framework, two other hybrid frameworks are used for comparison with HMJCDE, *i.e.*, HMJCDE-I and HMJCDE-II. HMJCDE-I is a random hybrid framework, which uses a given probability *pro* to randomly control the execution of MJADE or MCoDE. When *rand* < *pro*, MJADE will be operated. Otherwise, MCoDE will be run. The flowchart of HMJCDE-I is demonstrated in Fig. 12.

HMJCDE-II uses the average number of ***unsuccessful update*** to control the execution of MCoDE or MJADE. The flowchart of HMJCDE-II is demonstrated in Fig. 13, which is similar to HMJCDE. The *flag* parameter is also initialized to 2 for running MCoDE at first use. At the final steps of MCoDE or MJADE, the average number of unsuccessful update (*ANU*) is calculated. If *ANU* is larger than a predefined threshold *number*, the *flag* parameter will be changed to execute the other algorithm, and the *count* parameter that denotes the number of recently consecutive *unsuccessful updates* for each individual will be reset to 0. Note that the *count* parameter is a vector with *NP* elements.

In HMJCDE-I and HMJCDE-II, the *pro* and *number* parameters may have the significant effects on the optimization performance. In order to conduct a comprehensive comparison, five different values are assigned to *pro* (*i.e.*, 0.1, 0.3, 0.5, 0.7, and 0.9) and *number* (*i.e.*, 1, 5, 10, 15, and 20) respectively. The other parameter settings of HMJCDE-I and HMJCDE-II are set the same as that of HMJCDE in Table 1. Each value of *pro* and *number* is run independently by 30 times for each test function with 30*D*, and the experimental

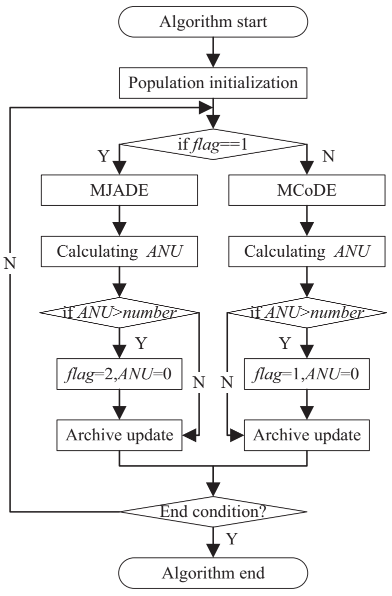

Fig. 13. The flowchart of HMJCDE-II.

results are respectively given in Tables 17 and 18. From Table 17, it is firstly found that HMJCDE-I with *pro* = 0.1 (operation probability of MJADE is 0.1 and operation probability of MCoDE is 0.9) is suitable for solving the complicated composition functions (F23-F30), while HMJCDE-I with *pro* = 0.9 (operation probability of MJADE is 0.9 and operation probability of MCoDE is 0.1) is good at handling the unimodal and simple multimodal functions (F1-F16). This is due to that MCoDE pays more attention to explo-

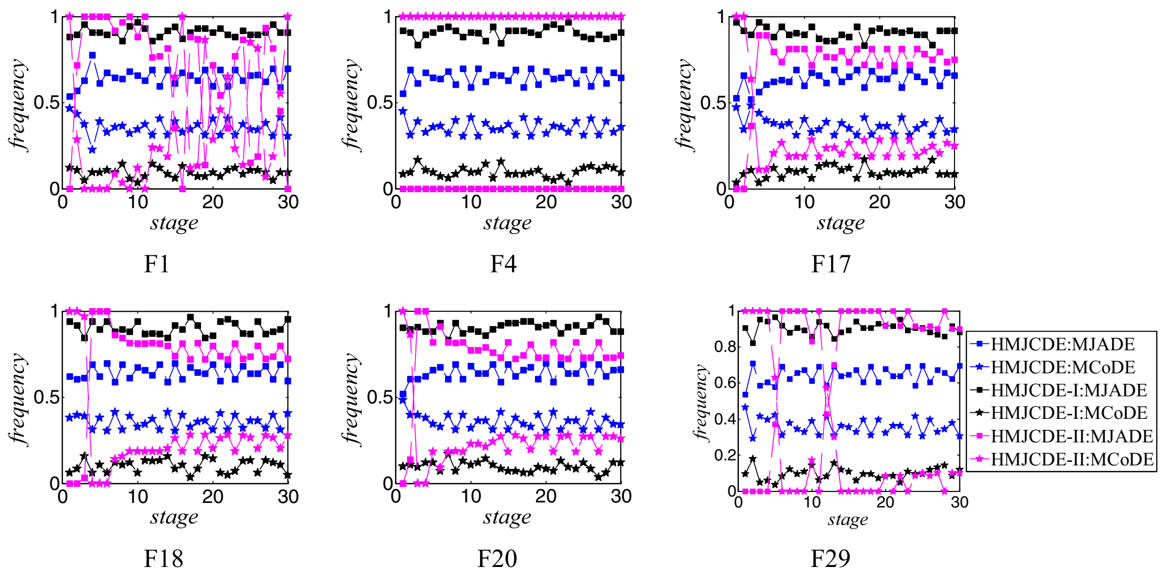

Fig. 14. The frequency of executing MCoDE and MJADE.

ration, while MJADE puts emphasis up on exploitation relatively. Nevertheless, HMJCDE is better than HMJCDE-I as HMJCDE outperforms HMJCDE-I with *pro* = 0.1, *pro* = 0.3, *pro* = 0.5, *pro* = 0.7 and *pro* = 0.9 on 15, 13, 10, 9 and 8 out of 30 test functions respectively. However, HMJCDE-I with *pro* = 0.1, *pro* = 0.3, *pro* = 0.5, *pro* = 0.7 and *pro* = 0.9 only surpass HMJCDE on 5, 5, 4, 3 and 3 test functions respectively. From Table 18, it is also observed that HMJCDE outperforms HMJCDE-II on almost half of test functions, *e.g.*, HMJCDE is better than HMJCDE-II with *number* = 1, *number* = 5, *number* = 10, *number* = 15 and *number* = 20 on 11, 13, 15, 12, and 11 out of 30 test functions respectively, but HMJCDE-II with *number* = 1, *number* = 5, *number* = 10, *number* = 15 and *number* = 20 only outperforms HMJCDE on 0, 1, 1, 1 and 0 test functions respectively. In addition, HMJCDE and HMJCDE-II obtain the similar performance on almost half of test functions. Therefore, these comparison results validate the effectiveness of our hybrid framework. This in turn confirms the superiority of our control approach based on the improvement rate to adjust the executions of MJADE and MCoDE.

In order to study the difference among HMJCDE-I, HMJCDE-II and HMJCDE, the execution frequency of MCoDE and MJADE at each stage of the evolution process is demonstrated for some typically test functions with 30*D* in Fig. 14. For the evolutionary process, 30 stages are divided equally according to the number of function evaluations (*i.e.*, 10000 function evaluations are treated as one stage in this study). At each stage, the execution numbers of MCoDE and MJADE are recorded respectively, and then the operating *frequency* of MCoDE and MJADE are calculated respectively. It is noted that the *pro* parameter is set to 0.9 in HMJCDE-I, while the *number* parameter is set to 15 in HMJCDE-II. An interesting phenomenon is observed that in HMJCDE, the operating frequency of MCoDE is between 0.3 and 0.5 and that of MJADE is between 0.5 and 0.7 at each stage for almost all the test functions. Obviously, the operating frequency of MCoDE and MJADE in HMJCDE are different from those of HMJCDE-I and HMJCDE-II. This indicates that our hybrid framework performs better than the other two hybrid frameworks in balancing the trade-off between exploration and exploitation.

6. Conclusions and future works

In this paper, we propose a novel hybrid framework based on JADE and CoDE. Due to the fact that JADE pays more attention to exploitation and is suitable for solving unimodal and simple multimodal problems, while CoDE focuses on exploration to make it good at tackling complicated multimodal problems, the aim of our proposed hybrid framework is to combine their merits and further enhance the robustness and optimization performance. To achieve this purpose, Gaussian mutation based on the best individual is introduced into JADE and the methods of updating parameter F_m and CR_m are modified for JADE. These modifications form a new algorithm MJADE with enhanced exploitation ability and better adaptation. For CoDE, an extra external archive is embedded, and the parameters *F* and *CR* are modified to be sampled following the Cauchy and Gaussian distributions. These modifications compose a new algorithm MCoDE with stronger exploration ability and better adaptation. At last, to adaptively control the executions of MCoDE and MJADE, a novel hybrid framework (HMJCDE) is accordingly proposed. Depending on the improvement rate obtained by MCoDE and MJADE, they are executed alternatively. That is to say, when the exploration algorithm (MCoDE) shows a negative performance according to the improvement rate, the exploitation algorithm (MJADE) will be executed in the next generation, and vice versa.

The effectiveness of MCoDE and MJADE is validated by the comparisons with CoDE and JADE. Moreover, HMJCDE also demonstrates the superior performance when compared with nine DE variants and five non-DE algorithms. The parameter sensitivity of HMJCDE is also analysed by some experimental studies. To further confirm the effectiveness of our proposed hybrid framework in HMJCDE, two other hybrid frameworks are introduced and compared to HMJCDE. The experimental results justify that our proposed hybrid framework has the obvious advantages over the other two simple frameworks.

In our future work, other competitive evolutionary algorithms will be studied in our hybrid framework, and HMJCDE can also be modified to handle some practical optimization problems.

Acknowledgments

This work is supported by the National Natural Science Foundation of China under Grants 61402291 and 61402294, Guangdong Natural Science Foundation under Grant S2013040012895, Foundation for Distinguished Young Talents in Higher Education of Guangdong, China under Grant 2013LYM_0076 and 2014KQNCX129, Major Fundamental Research Project in the Science and Technology Plan of Shenzhen under

Grants JCYJ20130329102017840, JCYJ20130329102032059 and JCYJ20140828163633977, and Natural Science Foundation of SZU under Grant 201531.

References

[1] A. Georgive, I. Jordanov, Global optimization based on novel heuristics, low-discrepancy sequences and genetic algorithm, Eur. J. Oper. Res. 196 (2) (2009) 413–422.
[2] M. Thakur, A new genetic algorithm for global optimization of multimodal continuous functions, J. Comput. Sci. 5 (2) (2014) 298–311.
[3] J.J. Liang, A.K. Qin, Ponnuthurai Nagaratnam Suganthan, S. Baskar, Comprehensive learning particle swarm optimizer for global optimization of multimodal functions, IEEE Trans. Evol. Comput. 10 (3) (2006) 281–295.
[4] Z. Beheshti, S.M. Shamsuddin, Non-parametric particle swarm optimization for global optimization, Appl. Soft Comput. 28 (2015) 345–359.
[5] Q.D. Qin, S. Cheng, Q.Y. Zhang, L. Li, Y.H. Shi, Biomomicry of parasitic behavior in a coevolutionary particle swarm optimization algorithm for global optimization, Appl. Soft Comput. 32 (2015) 224–240.
[6] D. Karaboga, A powerful and efficient algorithm for numerical function optimization: artificial bee colony (ABC) algorithm, J. Glob. Optim. 29 (2007) 459–471.
[7] W.F. Gao, L.L. Huang, S.Y. Liu, C. Dai, Artificial bee colony algorithm based on information Learning, IEEE Trans. Cybernet (2016), http://dx.doi.org/10.1109/TCYB.2014.2387067.
[8] Q.Z. Lin, Q.L. Zhu, P.Z. Huang, J.Y. Chen, Z. Ming, J.P. Yu, A novel hybrid multi-objective immune algorithm with adaptive differential evolution, Comput. Oper. Res. 65 (2015) 95–111.
[9] Z.P. Liang, R.Z. Song, Q.Z. Lin, Z.H. Du, J.Y. Chen, Z. Ming, J.P. Yu, A double-module immune algorithm for multi-objective optimization problems, Appl. Soft Comput. 35 (2015) 161–174.
[10] S. Das, A. Mukhopadhyay, A. Roy, A. Abraham, B.K. Panigrahi, Exploratory power of the harmony search algorithm: analysis and improvements for global numerical optimization, IEEE Trans. Syst. Man Cybernet. Part B: Cybernet. 40 (1) (2011) 89–106.
[11] R. Storn, K. Price, Differential evolution: a simple and efficient heuristic for global optimization over continuous space, J. Glob. Optim. 11 (1997) 341–359.
[12] R. Mallipeddi, P.N. Suganthan, Q.K. Pan, M.F. Tasgetiren, Differential evolution algorithm with ensemble of parameters and mutation strategies, Appl. Soft Comput. 11 (2) (2011) 1679–1696.
[13] Y. Chen, W.C. Xie, X.F. Zou, A binary differential evolution algorithm learning from explored solutions, Neurocomputing 149 (2015) 1038–1047.
[14] J.Q. Zhang, A.C. Sanderson, JADE: adaptive differential evolution with optional external archive, IEEE Trans. Evol. Comput. 13 (5) (2009) 945–958.
[15] Y. Wang, Z.X. Cai, Q.F. Zhang, Differential evolution with composite trial vector generation strategies and control parameters, IEEE Trans. Evol. Comput. 15 (1) (2011) 55–66.
[16] J.J. Liang, B.Y. Qu, P.N. Suganthan, Problem definitions and evaluation criteria for CEC 2014 special session and competition on single objective real-parameter numerical optimization. Technical Report Nanyang Technological University (Singapore) and Zhenzhou University (China), Dec. 2013. (Online available) http://www.ntu.edu.sg/home/epnsugan/.
[17] S. Rahnamayan, H.R. Tizhoosh, M.M.A. Salama, Opposition-based differential evolution, IEEE Trans. Evol. 12 (1) (2008) 64–79.
[18] A.K. Qin, V.L. Huang, P.N. Suganthan, Differential evolution algorithm with strategy adaptation for global numerical optimization, IEEE Trans. Evol.Comput. 12 (1) (2009) 398–417.
[19] A. Draa, S. Bouzoubia, I. Boukhalfa, A sinusoidal differential evolution algorithm for numerical optimisation, Appl. Soft Comput. 27 (2015) 99–126.
[20] Q. Ji, M. Chad, An Adaptive Unified Differential Evolution Algorithm, for Global Optimization, Online available http://www.osti.gov/scitech/biblio/1163660.
[21] D. Kovacevic, N. Mladenovic, B. Petrovic, P. Milosevic, DE-VNS: self-adaptive differential evolution with crossover neighborhood search for continuous global optimization, Comput. Oper. Res. 52 (2014) 157–169.
[22] M. Yang, C. Li, Z.H. Cai, J. Guan, Differential evolution with auto-enhanced population diversity, IEEE Trans. Cybernet. 45 (2) (2015) 302–315.
[23] W. Gong, Z. Cai, Differential evolution with ranking-based mutation operators, IEEE Trans. Cybernet. 43 (6) (2013) 2066–2081.
[24] R.V. Rao, V.J. Savsani, D.P. Vakharia, Teaching-learning-baded optimization: an optimization method for continuous non-liner large scale problems, Inf. Sci. 183 (2012) 1–15.
[25] N. Hansen, A. Ostermeier, Completely derandomized self-adaptation in evolution strategies, Evolut. Comput. 9 (2) (2001) 159–195.
[26] H.Y. Fan, J. Lampinen, A trigonometric mutation operation to differential evolution, J. Glob. Optim. 27 (1) (2007) 105–129.
[27] P. Kaelo, M.M. Ali, Differential evolution algorithm using hybrid mutation, Comput. Optim. Appl. 37 (2) (2007) 231–246.
[28] S. Das, A. Abraham, U.K. Chakraborty, A. Konar, Differential evolution using a neighborhood-based mutation operator, IEEE Trans. Evol. Comput. 13 (3) (2009) 526–553.
[29] S.M. Islam, S. Das, S. Ghosh, S. Roy, P.N. Suganthan, An adaptive differential evolution algorithm with novel mutation and crossover strategies for global numerical optimization, IEEE Trans. Syst. Man Cybernet. Part B: Cybernet. 42 (2) (2012) 482–500.
[30] W.J. Yu, M. Shen, W.N. Chen, Z.H. Zhan, Y.J. Gong, Y. Lin, Differential evolution with two-level parameter adaption, IEEE Trans. Cybernet. 44 (7) (2014) 1080–1099.
[31] L.Z. Cui, G.H. Li, Q.Z. Lin, J.Y. Chen, N. Lu, Adaptive differential evolution algorithm with novel mutation strategies in multiple sub-populations, Comput. Oper. Res. 67 (2016) 155–173.
[32] Y. Wang, Z.X. Cai, Q.F. Zhang, Enhancing the search ability of differential evolution through orthogonal crossover, Inf. Sci. 185 (2012) 153–177.
[33] S.M. Guo, C.C. Yang, Enhancing differential revolution utilizing eigenvector-based crossover operator, IEEE Trans. Evol. Comput. 19 (1) (2015) 31–49.
[34] Y. Wang, H.X. Li, T.W. Huang, L. Li, Differential evolution based on covariance matrix learning and bimodal distribution parameter setting, Appl. Soft Comput. 18 (2014) 232–247.
[35] J. Liu, J. Lampinen, A fuzzy adaptive differential evolution algorithm, Soft Comput. 9 (6) (2005) 448–462.
[36] J. Brest, S. Greiner, B. Boskovic, M. mernik, V. Zumer, Self-adapting control parameters in differential evolution: a comparative study on numerical benchmark problems, IEEE Trans. Evol. Comput. 10 (6) (2006) 646–657.
[37] R. Tanabe, A. Fukunaga, Success-history based parameter adaptation for differential evolution, 2013 IEEE Congress on Evolutionary Computation (CEC) (2013) 71–78.
[38] R. Tanabe, A. Fukunaga, Improving the search performance of SHADE using linear population size reduction, 2014 IEEE Congress on Evolutionary Computation (CEC) (2014) 1658–1665.
[39] R.A. Sarker, S.M. Elsayed, T. Ray, Differential evolution with dynamic parameters selection for optimization problems, IEEE Trans. Evol. Comput. 18 (5) (2014) 689–707.
[40] R. Mallipeddi, P.N. Suganthan, Differential evolution algorithm with ensemble of parameters and mutation and crossover strategies, in: Swarm, Evolutionary, and Memetic Computing, 2010, pp. 71–78.
[41] R. Mallipeddi, P.N. Suganthan, Ensemble differential evolution algorithm for CEC2011 problems, 2011 IEEE Congress on Evolutionary Computation (CEC) (2011) 1557–1564.
[42] Y. Wang, B.C. Wang, H.X. Li, G.G. Yen, Incorporating objective function information into the feasibility rule for constrained evolutionary optimization, IEEE Trans. Cybernet. 99 (2015) 1–15.
[43] Y. Wang, H.X. Li, G.G. Yen, W. Song, MOMMOP: multiobjective optimization for locating multiple optimal solutions of multimodal optimization problems, IEEE Trans. Cybernet. 45 (4) (2015) 830–843.
[44] X. Yang, G. Liu, Self-adaptive clustering-based differential evolution with new composite trial vector generation strategies, in: F.L. Gaol, Q.V. Nguyen (Eds.), Proceedings of the 2011 2nd International Congress on Computer Applications and Computational Science, vol. 144, Springer, Berlin/Heidelberg, 2012, pp. 261–267.
[45] S.M. Elsayed, R.A. Sarker, D.L. Essam, Multi-operator based evolutionary algorithms for solving constrained optimization problems, Comput. Oper. Res. 38 (2011) 1877–1896.
[46] S.M. Elsayed, R.A. Sarker, D.L. Essam, On an evolutionary approach for constrained optimization problem solving, Appl. Soft Comput. 12 (2012) 3208–3227.
[47] S.M. Elsayed, R.A. Saker, D.L. Essam, An improved self-adaptive differential evolution algorithm for optimization problems, IEEE Trans. Ind. Inf. 9 (1) (2013) 89–99.
[48] S.M. Elsayed, R.A. Sarker, D.L. Essam, Training and Testing a self-adaptive multi-operator evolutionary algorithm for constrained optimization, Appl. Soft Comput. 26 (2015) 515–522.
[49] S.M. Elsayed, R.A. Sarker, D.L. Essam, A self-adaptive combined strategies algorithm for constrained optimization using differential evolution, Appl. Math. Comput. 241 (2014) 267–282.
[50] M.G. Epitropakis, D.K. Tasoulis, N.G. Pavlidis, V.P. Plagianakos, M.N. Vrahatis, Enhancing differential evolution utilizing proximity based mutation operators, IEEE Trans. Evol. Comput. 15 (1) (2011) 99–119.
[51] W.Y. Gong, Z.H. Cai, D.W. Liang, Adaptive ranking mutation operator based differential evolution for constrained optimization, IEEE Trans. Cybernet. 45 (4) (2015) 716–727.
[52] Y. Cai, J. Wang, Differential evolution with neighborhood and direction information for numerical optimization, IEEE Trans. Cybernet. 43 (6) (2013) 2202–2215.
[53] S.M. Guo, C.C. Yang, P.H. Hsu, J.S.-H. Tsai, Improving differential evolution with successful-parent-selecting framework, IEEE Trans. Evol. Comput. 19 (5) (2015) 717–730.
[54] R. Thangaraj, M. Pant, A. Abraham, P. Bouvry, Particle swarm optimization: hybridization perspectives and experimental illustrations, Appl. Math. Comput. 217 (12) (2011) 5208–5226.
[55] A. Biswas, S. Dasgupta, S. Das, A. Abraham, A synergy of differential evolution and bacterial foraging optimization for global optimization, Neural Netw. World 17 (6) (2007) 607–626.
[56] W. Gong, Z. Cai, C.X. Ling, DE/BBO: a hybrid differential evolution with biogeography-based optimization for global numerical optimization, Soft Comput. 15 (4) (2011) 645–665.
[57] A. Abraham, R.K. Jatoth, A. Rajasekhar, Hybrid differential artificial bee colony algorithm, J. Comput. Theor. Nanosci. 9 (2012) 1–9.
[58] J.L. Zhang, Y.Q. Zhou, A hybrid optimization algorithm based on artificial glowworm swarm and differential evolution, Inf. Control 40 (5) (2011) 608–613.

[59] M. Crepinsek, S.-H. Liu, M. Mernik, Exploration and exploitation in evolutionary algorithms: a survey, ACM Comput. Surv. 45 (3) (2013), http://dx.doi.org/10.1145/2480741.2480752.
[60] M. Crepinsek, M. Mernik, S.H. Liu, Analysis of exploration and exploitation in evolutionary algorithm by ancestry trees, Int. J. Innov. Comp. Appl. 3 (1) (2011) 11–19.
[61] M. Crepinsek, S.-H. Liu, M. Mernik, Replication and comparison of computational experiments in applied evolutionary computing: common pitfalls and guideline to avoid them, Appl. Soft Comput. 19 (2014) 161–170.
[62] R.V. Rao, V. Patel, An elitist teaching-learning-based optimization algorithm for solving complex constrained optimization problems, Int. J. Ind. Eng. Comp. 3 (2012) 535–560.