Computers & Operations Research 75 (2016) 132–149

Contents lists available at ScienceDirect

Computers & Operations Research

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

A novel differential evolution algorithm using local abstract convex underestimate strategy for global optimization

Xiao-gen Zhou, Gui-jun Zhang*, Xiao-hu Hao, Li Yu

College of Information Engineering, Zhejiang University of Technology, Hangzhou, Zhejiang 310023, PR China

ARTICLE INFO

Article history:
Received 22 April 2015
Received in revised form
11 February 2016
Accepted 24 May 2016
Available online 27 May 2016

Keywords:
Differential evolution
Global optimization
Supporting hyperplane
Underestimate
Abstract convexity

ABSTRACT

Two main challenges in differential evolution (DE) are reducing the number of function evaluations required to obtain optimal solutions and balancing the exploration and exploitation. In this paper, a local abstract convex underestimate strategy based on abstract convexity theory is proposed to address these two problems. First, the supporting hyperplanes are constructed for the neighboring individuals of the trial individual. Consequently, the underestimate value of the trial individual can be obtained by the supporting hyperplanes of its neighboring individuals. Through the guidance of the underestimate value in the select operation, the number of function evaluations can be reduced obviously. Second, some invalid regions of the domain where the global optimum cannot be found are safely excluded according to the underestimate information to improve reliability and exploration efficiency. Finally, the descent directions of supporting hyperplanes are employed for local enhancement to enhance exploitation capability. Accordingly, a novel DE algorithm using local abstract convex underestimate strategy (DELU) is proposed. Numerical experiments on 23 bound-constrained benchmark functions show that the proposed DELU is significantly better than, or at least comparable to several state-of-the art DE variants, non-DE algorithms, and surrogate-assisted evolutionary algorithms.

1. Introduction

Global optimization has become one of the most widely used techniques for modeling and analyzing practical problems [1]. Much progress in the fields of science, economy, and engineering relies heavily on numerical techniques to obtain global optimal solutions for optimization problems. However, an algorithm may get trapped into the local optimum for large-scale real-world optimization problems. In view of the complexity of the problem, traditional algorithms, such as the gradient-based algorithms, cannot be used to find the global optimum.

Evolutionary algorithms (EAs) are a broad class of stochastic optimization algorithms inspired by the natural evolution of species. EAs have been successfully applied to solve numerous optimization problems in various fields. The EAs family includes differential evolution (DE) [2], genetic algorithm (GA) [3], particle swarm optimization (PSO) [4], evolution strategies (ES) [5], and evolutionary programming (EP) [6]. The properties such as derivative-free and strong robustness make these algorithms attractive to applications of various real-world optimization problems.

DE, which was proposed by Storn and Price [2], has been proven to be a simple yet effective stochastic global optimization algorithm in EAs. It is well-known for its simple structure, ease of use, robustness and speed. Owing to these advantages, DE has been successfully applied in diverse fields [7,8], such as power systems [9], communication [10], chemical engineering [11], optics [12], and bioinformatics [13]. However, despite having several striking advantages and successful applications in diverse fields, DE has been shown to have certain weaknesses. A large number of function evaluations are needed to find optimal solutions, which leads to an increase in computational time, particularly for expensive-to-evaluate optimization problems in real applications. For example, for the gasifier problem in [14], function evaluations account for more than 99% of the entire searching time. Greedy selection strategy accelerates the convergence of the algorithm but makes the algorithm easy to get trapped into local optimum. In addition, DE is fast at exploring but slow at exploiting the solution [15].

Surrogate model algorithms [16–18] have been recently developed to reduce the necessary number of function evaluations while searching the global optimum of the problem. In surrogate model algorithms, a surrogate model is used to replace computationally expensive real function evaluations. A surrogate model usually be represented as $f(x) = s(x) + \epsilon$, where $f(x)$ is the real function value of the point x, $s(x)$ is the value obtained by the surrogate model, and ϵ is the error between them. The main

* Corresponding author.
E-mail address: zgj@zjut.edu.cn (G.-j. Zhang).

http://dx.doi.org/10.1016/j.cor.2016.05.015
0305-0548/© 2016 Elsevier Ltd. All rights reserved.

advantage of this approach is that the surrogate model has a lower computational complexity than the original model of the problem, thereby significantly reducing the computational cost. Because of this advantage, surrogate models are widely utilized in many fields, such as the optimization of helicopter rotor blades [19], the optimization design of corrugated beam guardrails [20], and the inverse calculation of in situ stress in rock mass [21]. However, selecting the appropriate surrogate model is a challenge because a certain surrogate model cannot be suitable for all kinds of problems [22].

Abstract convexity theory [23,24] generalizes the property of convex analysis that every convex function is the upper envelop of its affine minorants [25]. Therefore, many nonconvex functions (so-called abstract convex functions) can be represented as lower envelopes of some basic simple functions. Based on this property, Beliakov [26] proposed a cutting angle method (CAM). In CAM, the simple functions are replaced by support functions. By using a sequence of support functions, a lower approximation of the original problem can be obtained from below. Since the minimization of the lower approximation is much simpler than the original problem, it is called a relaxed model. Consequently, the global minimum of the objective problem can be obtained by enumerating all local minima of the relaxed model efficiently. Such underestimate technique is very useful in various applications [27–29]. However, to obtain a promising solution through a more accurate lower approximation, the method needs to use numerous support functions, which lead to the extremely high complexity. Therefore, CAM is efficient only for the problem with less than 10 variables [30–32].

In this paper, an effective DE algorithm based on the underestimate technique presented in CAM is proposed. The proposed algorithm, called DELU, uses local abstract convex underestimate strategy to reduce the number of function evaluations and to balance the exploration and exploitation of DE. Unlike the underestimate technique used in CAM, the local underestimate strategy only constructs the supporting hyperplanes for the neighboring individuals of the trial individual, and the neighboring individuals are selected by the Euclidean distance from the trial individual to the other individuals. Then, in the selection operation, the underestimate value of the trial individual is calculated by the supporting hyperplanes of the individuals near the trial. According to the underestimate value, we can judge whether the trial individual is worth evaluating, thus reducing the number of function evaluations. Specifically, the underestimate information also can be used to safely exclude some invalid regions of the domain where the global optimum cannot be found. This procedure prevents the algorithm from getting trapped in the local minimum in some case and improves exploration efficiency. In addition, exploitation capability is enhanced by employing the descent directions of supporting hyperplanes for local enhancement. Competitive experimental results are observed with respect to commonly used benchmark functions. Compared with four state-of-the-art DE variants, three non-DE algorithms and three surrogate-assisted evolutionary algorithms, our approach performs better, or at least comparably, in terms of the quality of the final solutions and convergence speed. In addition, the proposed local abstract convex underestimate strategy is also integrated into some advanced DE variants to verify the effect on them. Experimental results show that our proposed local abstract convex underestimate strategy is able to enhance the advanced DE algorithms.

The reminder of this paper is organized as follows. In Section 2, some related works of DE are presented. Section 3 briefly describes the DE algorithm and the cutting angle method. Section 4 describes the proposed algorithm at a finer level of detail. Experimental results demonstrating the performance of the proposed algorithm in comparison with five state-of-the art DE variants, four non-DE algorithms, and three surrogate-assisted EAs over a suite of bound-constrained numerical optimization functions are presented in Section 5. Section 6 concludes this paper.

2. Related works

DE has drawn the attention of many researchers all over the world. They have proposed many variants to reduce the computational cost, balance the exploration and exploitation, and improve the optimization capability of DE. In this section, a brief overview of these enhanced approaches is presented.

Some work mainly focuses on the function approximation technique to reduce the computational cost. Queipo [33] proposed a meta-model (or surrogate-model) as the approximation of the original function to replace calls to the expensive function evaluations. Liu [16] proposed a Gaussian process surrogate model assisted evolutionary algorithm, which map the training data to a lower dimensional space by employing a dimension reduction technique, and a new surrogate model-aware search mechanism is used to make the search focus on the promising subregion. Zhou [17] proposed a memetic algorithm using multi-surrogates. It combined the regression and exact interpolating surrogate model for local search to solve the expensive-to-evaluate problems. Liu [34] proposed a fast differential evolution using *k*-nearest neighbor predictor as function approximation. Jin [35] combined a multi-layer perceptron (MLP), a kind of neural network (NN), with covariance matrix adaptation evolution strategy (CMA-ES) to build an efficient evolutionary optimization with function approximation. Zhang [36] developed a predictive distribution model combining Gaussian stochastic model with fuzzy clustering based model to measure the expected improvement of each individual. Park [37] also proposed an efficient differential evolution using *k*-nearest neighbor as function estimator to alleviate a burden of a large number of function evaluations.

Many attempts have also been made to balance the exploration and exploitation. Wang [38] proposed a novel hybrid discrete differential evolution algorithm (HDDE), in which a local search algorithm based on insert neighborhood structure is embedded to balance the exploration and exploitation by enhancing the local searching ability. Bhattacharya [9] proposed a hybrid differential evolution with biogeography-based optimization (DE/BBO), which combines the exploration of DE with the exploitation of BBO effectively. Cai [39] integrated the one-step *k*-means clustering into DE (CDE), which makes the original DE more effective and efficient. Piotrowski [40] proposed a differential evolution with separated groups (DE-SG), which distributes population into small groups, defines rules of exchange of information and individuals between the groups and uses two different strategies to keep balance between exploration and exploitation capabilities. Li [41] proposed a modified differential evolution with self-adaptive parameters method (MDE), in which a probability rule is used to combine two different mutation rules to enhance the diversity of the population and the convergence rate of the algorithm.

Apart from the above methods, some researches consider to improve the optimization capability by the adaptive control parameters and strategies. Qin [42] proposed a self-adaptive differential evolution algorithm (SaDE), in which both trial vector generation strategies and their associated control parameter values are gradually self-adapted by learning from their previous experiences in generating promising solutions. Zhang [43] proposed an adaptive differential evolution with optional external archive (JADE) which improved optimization performance by implementing a new mutation strategy with optional external archive and automatically updated the parameters by evolving the mutation factors and crossover probabilities based on their historical record of success. Wang [44] proposed a differential evolution algorithm with composite trial vector generation strategies and control parameters (CoDE), the primary idea of which is to randomly combine several trial vector generation strategies with a number of control parameter settings at each generation to create new trial vectors. Mallipeddi [45] proposed an ensemble of mutation strategies and control parameters with DE (EPSDE), in which a pool of distinct mutation strategies, along with a pool of values for each control parameter, coexists throughout the evolution process and competes to produce offspring. Pan [46] proposed a differential evolution algorithm with self-adapting strategy and control parameters (SspDE). In SspDE, each target individual has its own strategy, scaling factor (F) and crossover rate (CR), and they can be self-adapted in different search stages according to their previous successful experience in generations.

3. Preliminary

3.1. Differential evolution

Differential evolution [2] is a population-based stochastic search algorithm for global optimization. Similar to other EAs, DE also includes initialisation, mutation, crossover and selection operation. Overall, DE is a simple evolutionary algorithm and often performs better than the traditional EAs because of the greedy selection strategy. The details of DE are described as follows.

Suppose that the objective function to be minimized is $f(x)$, $x=(x_1, x_2, \ldots, x_D)\in R^D$. Firstly, at generation $g=0$, an initial population $P=\{x_i^0=(x_{i,1}^0, x_{i,2}^0, \ldots, x_{i,D}^0),\ i=1,2,\ldots,NP\}$ is randomly sampled from the feasible solution space, where NP is the population size. Then the mutation and crossover operation are used to generate the trial individuals. Finally, the selection operation is applied to determine whether the trial vector will survive to the next generation or not.

3.1.1. Mutation operation

To drive the population to explore the search space, the mutation operation is adopted. At each generation g, DE creates a mutation vector $v_i^g=(v_{i,1}^g, v_{i,2}^g, \ldots, v_{i,D}^g)$ for each individual x_i^g (called a target vector) in the current population. The mutation vector can be generated in the following way:

$$v_i^g = x_{r_1}^g + F\cdot\left(x_{r_2}^g - x_{r_3}^g\right) \tag{1}$$

where r_1, r_2, and r_3 are distinct integers randomly selected from the range $[1, NP]$ and are also different from i. The parameter F is called the scaling factor, which amplifies the difference variation.

3.1.2. Crossover operation

To enhance the potential diversity of the population, a crossover operation is performed after mutation. DE performs a binomial crossover operation on x_i^g and v_i^g to generate a trial vector $u_i^g=(u_{i,1}^g, u_{i,2}^g, \ldots, u_{i,D}^g)$. The trial vector is generated as

$$u_{i,j}^g = \begin{cases} v_{i,j}^g, & \text{if } \mathrm{rand}_j(0,1)\le CR \text{ or } j=j_{\mathrm{rand}} \\ x_{i,j}^g, & \text{otherwise} \end{cases} \tag{2}$$

where $j=1,2,\ldots,D$, j_{rand} is a randomly chosen index from $[1, D]$, $\mathrm{rand}_j(0,1)$ is a uniformly distributed random number generated from $[0, 1]$ for each j, and $CR\in(0,1)$ is called the crossover control parameter. Due to the use of j_{rand}, the trial vector u_i^g differs from its target vector x_i^g.

3.1.3. Selection operation

The purpose of selection operation is to select the better one from the target vector x_i^g and the trial vector u_i^g to enter the next generation. The selection operation is described as

$$x_i^{g+1} = \begin{cases} u_i^g, & \text{if } f(u_i^g)\le f(x_i^g) \\ x_i^g, & \text{otherwise} \end{cases} \tag{3}$$

Namely, if the trial vector yields an equal or lower value of the objective function, it replaces the corresponding target vector in the next generation; otherwise the target vector is retained in the population. Hence the population either gets better or remains the same in function status, but never deteriorates.

3.2. Cutting angle method

The cutting angle method [26,47] is a deterministic method for global Lipschitz optimization. It is based on the results of abstract convexity theory. In CAM, an approximation of the objective function can be obtained from below by using a subset of relaxed functions (or support functions). As the number of support functions increases, the lower approximation becomes more and more accurate, and the sequence of global minima of support functions converge to that of the objective function. In addition, it also allows one to safely exclude parts of the domain where the global optimum cannot be found by using global properties of the objective function. One such global property is a Lipschitz constant of a function [29], which puts a bound on the rate of change of the function.

CAM is applicable to IPH (Increasing Positively Homogeneous functions) with the unit simplex S ($S\equiv\{x\in R^D, x_j\ge 0, \sum_{j=1}^{D}x_j=1\}$) as their domain. Consider an objective problem $f(x)$ to be minimized satisfies Lipschitz condition [32] with a known Lipschitz constant M, and its domain is D-dimensional unit simplex. Assuming that the objective function f is extended to IPH with the help of an additive constant [23] $C > -\min f(x) + 2M$, where M is the Lipschitz constant of f in L_1-norm.

Given a set of K support vectors of the points x^k, and let J denote $\{1, 2, \ldots, D\}$. Then, a lower approximation of f can be built from below, using pointwise maxima of subsets of the support functions h^k, $k=1,2,\ldots,K$, namely

$$H^K(x) = \max_{k\le K} h^k(x) = \max_{k\le K}\min_{j\in J} l_j^k x_j \tag{4}$$

where the kth support function is given by

$$h^k(x) = \min_{j\in J} l_j x_j = f(x^k)\min\left\{\frac{x_1}{x_1^k}, \ldots, \frac{x_D}{x_D^k}\right\} \tag{5}$$

The vectors

$$l^k = \left(\frac{f(x^k)}{x_1^k}, \frac{f(x^k)}{x_2^k}, \ldots, \frac{f(x^k)}{x_D^k}\right) \tag{6}$$

is called support vectors, while the first D support vectors always based at the vertices of the unit simplex.

Such a lower approximation is usually called a relaxed model (or saw-tooth underestimate) of f. CAM obtains a more and more accurate underestimate of f by using more and more support functions, which are taken at the global minima of the lower approximation. At each iteration, the relaxed problem (4) is used to replace the original problem. Since the relaxed model always underestimates the objective function, its minima are always below f. Therefore, we can get the global minimum of the objective

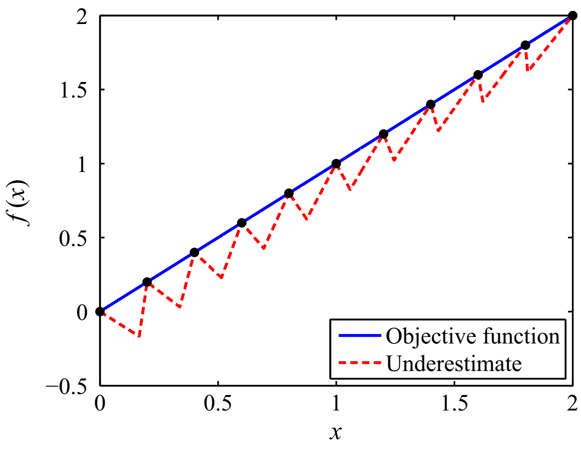


Fig. 1. Illustration of the underestimate.

function by enumerating all local minima of the relaxed model.

Beliakov has presented a combinatorial approach to solve the relaxed model [26]. Defined an ordered combinations $L = \{l^{k_1}, l^{k_2}, \ldots, l^{k_D}\}^T$ of D support vectors as an effective support matrix if it satisfies the following two conditions:

$$\text{(I)} \quad \forall\, m, n \in J,\ m \neq n:\ l_m^{k_m} < l_m^{k_n}$$

$$\text{(II)} \quad \forall\, v \in \Lambda^K \backslash L,\ \exists\, m \in J:\ l_m^{k_m} \geq v_m \tag{9}$$

Condition (I) indicates that the diagonal elements strictly dominate their columns, and condition (II) indicates that the diagonal of L does not dominate any other support vector v, not already in L.

Fig. 1 illustrates the underestimate of the objective function $f(x) = x$, where the solid line indicates the objective function, and the dotted line indicates its underestimate (relaxed model). As can be seen, the objective function is divided into many different regions by the supporting hyperplanes and each of these regions has its corresponding underestimate region (each underestimate region corresponds to a support matrix).

Think of an effective support matrix L, whose rows are support vectors $l^{k_1}, l^{k_2}, \ldots, l^{k_D}$:

$$L = \begin{pmatrix} l_1^{k_1} & l_2^{k_1} & \cdots & l_D^{k_1} \\ l_1^{k_2} & l_2^{k_2} & \cdots & l_D^{k_2} \\ \vdots & \vdots & \ddots & \vdots \\ l_1^{k_D} & l_2^{k_D} & \cdots & l_D^{k_D} \end{pmatrix} \tag{7}$$

The location of the local minimum $x_{\min}$ and its value $d = H^K(x_{\min})$ of the underestimate region corresponded to L can be found from the diagonal of L:

$$x_{\min}(L) = \operatorname{diag}(L)/\operatorname{tr}(L) \tag{8}$$

$$d(L) = H^K(x_{\min}) = 1/\operatorname{tr}(L) \tag{9}$$

4. DE algorithm using local abstract convex underestimate strategy

As mentioned in Section 3, the underestimate of the objective function can be calculated by the supporting hyperplanes, and some invalid regions of the domain where the global optimum cannot reside also can be excluded according to the underestimate information. Therefore, abstract convex supporting hyperplanes can be constructed to replace computationally expensive real function evaluations and to improve the reliability and exploration efficiency. Based on these advantages, in order to reduce the number of function evaluations and balance the exploration and exploitation of DE, in this section, a differential evolution algorithm using local abstract convex underestimate strategy is proposed, called DELU.

In CAM [26], an increasingly accurate lower approximation of the objective function is built by using more and more supporting hyperplanes, which are constructed for the global minima of the current lower approximation (the minimum of local minima of all underestimate regions). By contrast, in the proposed local abstract convex underestimate strategy, the supporting hyperplanes are only constructed for the nearest N neighboring individuals (measured in terms of Euclidean distances of parameter space) of the trial individual in DELU. Then the underestimate information is obtained from the supporting hyperplanes of the neighboring individuals of the trial to guide the selection operation, exclude some invalid regions of the search domain, and perform local enhancement. In addition, to reduce the complexity of the algorithm, all supporting hyperplanes are deleted at the end of each iteration. The local abstract convex underestimate strategy includes the local underestimate selection operator, the invalid region exclusion operator, and the local enhancement operator.

4.1. *Local underestimate selection operator*

After the generation of the trial individual x_{trial}, its N nearest neighboring individuals which are measured by Euclidean distances are selected to construct the supporting hyperplanes. Assuming that x_{near}^t, $t = 1, 2, \ldots, N$ is the N nearest individuals to x_{trial}. As the underestimate technique is applicable to IPH function restricted in the unit simplex S, each initial vector $x_{\text{near}}^t = (x_{\text{near},1}^t, \ldots, x_{\text{near},D}^t)$ needs to be transformed as

$$\begin{cases} x'_{\text{near},j} = (x_{\text{near},j} - a_j) \Big/ \sum_{j=1}^{D} (b_j - a_j) \\ x'_{\text{near},D+1} = 1 - \sum_{j=1}^{D} x'_{\text{near},j} \end{cases} \tag{10}$$

where $j = 1, 2, \ldots, D$, a_j and b_j are the lower bound and upper bound of $x_{\text{near},j}$, respectively. The vector $x'_{\text{near}} = (x'_{\text{near},1}, \ldots, x'_{\text{near},D+1})$ is the transform vector of x_{near} that is restricted in the unit simplex, where $x'_{\text{near},D+1}$ is the extra-addition variable. By adding a constant C, the objective function f is extended to IPH, then the support vectors of x_{near}^t can be calculated by

$$l^t = \left(\frac{f(x_{\text{near}}^t)}{x'^{\,t}_{\text{near},1}}, \frac{f(x_{\text{near}}^t)}{x'^{\,t}_{\text{near},2}}, \ldots, \frac{f(x_{\text{near}}^t)}{x'^{\,t}_{\text{near},D+1}} \right) \tag{11}$$

According to the abstract convexity theory, the underestimate value y_{trial} of the trial individual can be calculated through the support vectors of its N neighboring individuals, namely

$$y_{\text{trial}} = \max_{t \leq N} \min_{j=1,\ldots,D+1} l_j^t x'_{\text{trial},j} \tag{12}$$

Then y_{trial} can be used to replace the real function value $f(x_{\text{trial}})$ of x_{trial}. Hence, the selection operation can be guided by comparing y_{trial} with the function value $f(x_i^g)$ of the target individual x_i^g according to

$$x_i^{g+1} = \begin{cases} x_i^g, & \text{if } y_{\text{trial}} \ge f(x_i^g) \\ \begin{cases} x_{\text{trial}}, & \text{if } f(x_{\text{trial}}) \le f(x_i^g) \\ x_i^g, & \text{otherwise} \end{cases}, & \text{otherwise} \end{cases} \tag{13}$$

Namely, if the underestimate value y_{trial} of the trial individual x_{trial} is larger than the function value $f(x_i^g)$ of the target individual x_i^g, the target individual is retain unchanged, without need to evaluate the trial individual. Otherwise, x_{trial} replace x_i^g if x_{trial} obtains a lower value than x_i^g in aspects of function values, otherwise x_i^g is retain unchanged. Due to the guidance of the trial individual's underestimate y_{trial} in the selection operation, we do not need to evaluate each trial individual generated in every iteration. Thus the number of function evaluations can be reduced significantly.

4.2. Invalid region exclusion operator

Some invalid regions of the search domain that could not include the global optimum can be excluded according to the global property, that is, the Lipschitz constant of the objective function. To exclude some invalid regions, as well as prevent the trial individual from falling into them, the memory named *IR* is established to store support matrixes of invalid regions.

Define L^u as the support matrix of the underestimate region formed by the supporting hyperplanes of the N neighboring individuals of the trial individual. Since the underestimate is always below the objective function, therefore, if the local minimum value $d(L^u)$ of the underestimate region is larger than the function value of the best individual in the current generation, then we can conclude that the global optimum cannot be found in the search region corresponding to the underestimate region. Thus, this search region is considered an invalid region, and its support matrix is added to *IR*. This operation is executed only when the trial individual has a larger value than the function value of the target individual, and *IR* is initialized as empty at the beginning of the algorithm. The local minimum value $d(L^u)$ of the underestimate region can be calculated according to formula (9). Furthermore, when the trial individual is generated, each iteration should be assessed whether it is included in invalid regions, if *IR* is not empty. Suppose that there exists a search region, and its support matrix is L^R. Then, the trial vector x_{trial} is included in L^R (or the search region) if its all components satisfy

$$x_n'^{k_n} x'_{\text{trial},m} < x_m'^{\prime k_n} x'_{\text{trial},n}, \quad m, n \in \{1, \ldots, D+1\}, \quad m \ne n \tag{14}$$

where $x'_{\text{trial},m}$ and $x'_{\text{trial},n}$ are the mth and nth component of x'_{trial}, respectively, while x'_{trial} is the transform vector of x_{trial} that is obtained by formula (10). $x_n'^{k_n}$ is the nth diagonal element on the k_nth row of L^R, while $x_m'^{k_n}$ is the mth element on the k_nth row of L^R, but not a diagonal element.

This operation prevents the algorithm from searching in some invalid regions, but focus on the promising search region, thereby reducing the computational cost and speeding up convergence. In particular, it also prevents the premature convergence to a certain extent. Consequently, the reliability and exploration efficiency of the algorithm are improved.

4.3. Local enhancement operator

The descent directions of supporting hyperplanes can be employed for local enhancement to further accelerate the convergence. Define the directions of supporting hyperplanes as the generalized descent directions for underestimate. Since the point in the objective function corresponding to the local minimum of the underestimate region obtains the lowest underestimate value, then this point may correspond to the solutions with lower function values. When the trial individual x_{trial} is better than the target individual x_i^g in terms of function values, we continue to calculate the local minimum $x_{\min}(L^u)$ of the underestimate region wherein the trial individual is located. Then, a more better solution may be obtained through

$$x_i^{g+1} = \begin{cases} x_{\min}(L^u), & \text{if } f(x_{\min}(L^u)) \le f(x_i^g) \\ x_{\text{trial}}, & \text{otherwise} \end{cases} \tag{15}$$

where $x_{\min}(L^u)$ can be calculated according to formula (8). To be specific, if the local minimum $x_{\min}(L^u)$ yields a lower value than the target individual of the function, the target individual is replaced by $x_{\min}(L^u)$, otherwise x_{trial} replaces x_i^g. Consequently, the exploitation capability is enhanced according to the generalized descent directions of the support hyperplanes. Note that this operation is only executed in the case of the trial individual which is better than the target individual.

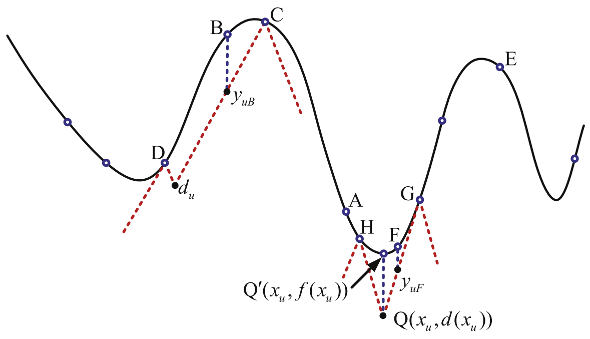


Fig. 2. Illustration of the selection operation in DELU.

4.4. Example of local abstract convex under estimate strategy

The local abstract convex underestimate strategy guides the selection operation of DE by constructing support hyperplanes for the N nearest neighboring individuals of the trial individual. Here, an example of two conditions that may occur in the selection operation is shown in Fig. 2. First, assume that B is the trial individual and A is the target individual. Then, construct supporting hyperplanes for the two individuals C and D that are nearest to B, and calculate the underestimate y_{uB} of B. Since y_{uB} is larger than the objective function value of A, A retains unchanged without need to evaluate the objective function value of B. Moreover, if the local minimum value d_u of the underestimate region which includes B is larger than the objective function value of the best solution found so far, the region is regarded as an invalid region where the global optimum cannot reside. Then, the supporting hyperplanes of C and D are deleted. Second, suppose that F is the trial individual that is not included in invalid regions, and E is the target individual, construct supporting hyperplanes for the two individuals G and H that are nearest to F and calculate the underestimate y_{uF} of F. Since y_{uF} is smaller than the objective function value of E, and the objective function value of F is smaller than that of E, F replaces E. To speed up the convergence, we calculate the local optimum point $Q(x_u, d(x_u))$ of the region in which F is located and its corresponding point $Q'(x_u, f(x_u))$ in the objective function. Since the function value of Q' is smaller than that of F, Q' replaces F, and the supporting hyperplanes of G and H are deleted. In Algorithm 1 the pseudocode of the local abstract convex underestimate strategy is presented.

Algorithm 1. The local abstract convex underestimate strategy.

```
Input: trial individual $x_{\text{trial}}$, target individual $x_i^g$, invalid region $IR$
Output: selection result
Begin
    for $t \leftarrow 1$ to $N$
        Select neighboring individuals $x_{\text{near}}^t$ of the trial;
        Calculate support vectors $l^t$ of $x_{\text{near}}^t$ according to (11);
    end
    Find out support matrix $L^u$ included $x_{\text{trial}}$ according to (14);
    if (the search region corresponded to $L^u \notin IR$)
        Calculate underestimate value $y_{\text{trial}}$ according to (12);
        if ($y_{\text{trial}} >= f(x_i)$)
            $x_i^{g+1} = x_i^g$;
            Calculate $d(L^u)$ according to (9);
            if ($d(L^u) >= f(x_{\text{best}}^g)$)
                Add $L^u$ of the search region to $IR$;
            end
        else
            if ($f(x_{\text{trial}}) < f(x_i)$)
                $x_i^{g+1} = x_{\text{trial}}$;
                Calculate $x_{\min}(L^u)$ according to (8);
                if ($f(x_{\min}(L^u)) < f(x_{\text{trial}})$)
                    $x_i^{g+1} = x_{\min}(L^u)$;
                end
            end
        end
    end
    delete all support vectors $l^t$;
End
```

4.5. Runtime complexity of local abstract convex underestimate strategy

For the local underestimate selection operator, the supporting hyperplanes for the N neighboring individuals of the trial individual must first be constructed before the underestimate value of the trial individual can be calculated. Since we use Euclidean distance to measure the distances between all individuals and the trial individual, the runtime complexity of computing the distances for all individuals is $O(NP\cdot D)$. The process of finding the N neighboring individuals of the trial individual can be completed in $O(NP\cdot N)$ time. The runtime complexity of constructing the supporting hyperplanes is $O(N\cdot(D+1))$. The runtime complexity of calculating the underestimate value of the trial individual also is $O(N\cdot(D+1))$. Hence, the overall runtime complexity of the local underestimate selection operator is $O(\max\{NP\cdot D, NP\cdot(D+1)\})$.

For the invalid region exclusion operator, the local minimum value of the underestimate region is first calculated, followed by obtaining the best solution according to the objective function value of each individual, and then judging whether the trial individual is included in invalid regions. The runtime complexity of calculating the local minimum of the underestimate region is $O(1)$. The procedure of finding the best solution can be completed in $O(NP-1)$ time. Since we adopt the n-ary tree data structure to preserve the underestimate value, the runtime complexity of judging whether the trial individual is included in invalid regions is $O(\log_D T\cdot(D+1))$, where T is the number of leaf nodes in the data structure. Thus, the overall runtime complexity of the invalid region exclusion operator is $O(\max\{NP-1, \log_D T\cdot(D+1)\})$.

For the local enhancement operator, only the local minimum of the underestimate region needs to be calculated. Therefore, the runtime complexity of the local enhancement operator is $O(D+1)$.

The invalid region exclusion operator and the local enhancement operator are executed only when specific conditions are satisfied. If all of the above three operations are executed at every iteration, the overall runtime complexity of the local abstract convex underestimate strategy would be $O(\max\{NP\cdot D, NP\cdot N, N\cdot(D+1), \log_D T\cdot(D+1)\})$. We only construct the supporting hyperplanes for the two neighboring individuals of the trial individual at every iteration; that is, $N=2$. Thus, the number T of leaf nodes in the data structure is also less than D. If $D > N$, the runtime complexity of the local abstract convex underestimate strategy is $O(NP\cdot D)$. Therefore, we can conclude that the local abstract convex underestimate strategy does not impose a serious burden on the runtime complexity. The runtime taken by the local abstract convex underestimate strategy can be ignored compared with the runtime needed to evaluate the functions of expensive-to-evaluate problems. This is because the runtime needed in evaluating the functions of expensive-to-evaluate problems is often very large that the runtime taken for the local abstract convex underestimate strategy is comparatively small.

5. Numerical experiments and results

In this section, various experiments on optimization benchmark problems are implemented to verify the performance of the proposed DELU algorithm.

5.1. Benchmark functions and parameter settings

Twenty-three different scalable benchmark functions are used to evaluate the proposed DELU algorithm. Specifically, functions f_1-f_{18} are high-dimensional functions. Functions f_1-f_8 are unimodal functions. Functions f_9-f_{18} are multimodal functions, and the number of local minima increases exponentially along with the dimension of the problem increasing. Functions $f_{19}-f_{23}$ are low-dimensional multimodal functions with few local minima. A detailed description of these benchmarks is presented in Table 1.

In the following experiments, the parameter self-adaptive technique proposed in SaDE [42] is adopted to adaptively select the values of the scaling factor (F) and the crossover rate (CR) in our proposed DELU. Briefly, the scaling factor (F) is generated by a normal distribution $N(0.5, 0.3)$, where 0.5 is the mean value and 0.3 is the standard deviation. To adapt the crossover rate (CR), its value is generated according to $N(CRm, 0.1)$. In the first LP generation, CRm is initialized as 0.5. At each generation after LP generations, the median value of the CR that has generated trial vectors successfully entering the next generation within the previous LP generations is calculated to overwrite CRm. More detailed description of the parameter self-adaptive technique can be found in [42]. Besides, the number N of neighboring individuals of the trial individual is used to construct supporting hyperplanes in DELU is set to 2, and the population size in DELU is set to 50. Each function for each algorithm was executed 30 times, respectively. For clarity, the best results are marked in boldface.

5.2. Comparison of DELU with five state-of-the-art DE variants

In this section, the proposed DELU is compared with five state-of-the-art DE variants, which are listed as follows:

(1) *SaDE*: DE with strategy adaptation [42].
(2) *EPSDE*: DE with ensemble of parameters and mutation strategies [45].
(3) *JADE*: adaptive DE with optional external archive [43].
(4) *CoDE*: DE with composite trial vector generation strategies and control parameters [44].
(5) *SHADE*: Success-history based parameter adaptation for DE [48].

Table 1
Twenty-three benchmark functions used in the experimental studies.

Name	Function	D	Search range	Optimum
Sphere	$f_1(x)=\sum_{i=1}^{D}x_i^2$	30	$[-100, 100]$	0
SumSquares	$f_2(x)=\sum_{i=1}^{D}ix_i^2$	30	$[-10, 10]$	0
Schwefel 2.22	$f_3(x)=\sum_{i=1}^{D}\lvert x_i\rvert+\prod_{i=1}^{D}\lvert x_i\rvert$	30	$[-10, 10]$	0
Exponential	$f_4(x)=-\exp\left(-0.5\sum_{i=1}^{D}x_i^2\right)$	30	$[-1, 1]$	−1
Tablet	$f_5(x)=10^6x_1^2+\sum_{i=2}^{D}x_i^2$	30	$[-100, 100]$	0
Step	$f_6(x)=\sum_{i=1}^{D}\lfloor x_i+0.5\rfloor^2$	30	$[-100, 100]$	0
Zakharov	$f_7(x)=\sum_{i=1}^{D}x_i^2+\left(\sum_{i=1}^{D}0.5ix_i\right)^2+\left(\sum_{i=1}^{D}0.5ix_i\right)^4$	30	$[-5, 10]$	0
Rosenbrock	$f_8(x)=\sum_{i=1}^{D-1}\left(100(x_{i+1}-x_i^2)^2+(x_i-1)^2\right)$	30	$[-2, 2]$	0
Griewank	$f_9(x)=1+\frac{1}{4000}\sum_{i=1}^{D}x_i^2-\prod_{i=1}^{D}\cos\left(\frac{x_i}{\sqrt{i}}\right)$	30	$[-600, 600]$	0
Schaffer 2	$f_{10}(x)=\sum_{i=1}^{D-1}\left(x_i^2+x_{i+1}^2\right)^{0.25}\left(\sin^2(50(x_i^2+x_{i+1}^2)^{0.1})+1\right)$	30	$[-100, 100]$	0
Schwefel 2.26	$f_{11}(x)=-\sum_{i=1}^{D}\left(x_i\sin(\sqrt{\lvert x_i\rvert})\right)$	30	$[-500, 500]$	$-418.983D$
Himmelblau	$f_{12}(x)=\frac{1}{D}\sum_{i=1}^{D}\left(x_i^4-16x_i^2+5x_i\right)$	30	$[-100, 100]$	−78.3323
Levy and Montalvo 1	$f_{13}(x)=\frac{\pi}{D}\left(10\sin^2(\pi y_1)+\sum_{i=1}^{D-1}(y_i-1)^2\left(1+10\sin^2(\pi y_i+1)\right)+(y_D-1)^2\right),$ $y_i=1+\frac{1}{4}(x_i+1)$	30	$[-10, 10]$	0
Levy and Montalvo 2	$f_{14}(x)=0.1\left(\sin^2(3\pi x_1)+\sum_{i=1}^{D-1}(x_i-1)^2\left(1+\sin^2(3\pi x_{i+1})+(x_D-1)^2\left(1+\sin^2(2\pi x_D)\right)\right)\right)$	30	$[-5, 5]$	0
Ackley	$f_{15}(x)=-20\exp\left(-0.2\sqrt{D^{-1}\sum_{i=1}^{D}x_i^2}\right)-\exp\left(D^{-1}\sum_{i=1}^{D}\cos(2\pi x_i)\right)+20+e$	30	$[-30, 30]$	0
Rastrigin	$f_{16}(x)=10D+\sum_{i=1}^{D}(x_i^2-10\cos(2\pi x_i))$	30	$[-5, 5]^D$	0
Penalized 1	$f_{17}(x)=\frac{\pi}{D}\left\{\sum_{i=1}^{D-1}(y_i-1)^2[1+\sin(\pi y_{i+1})]+(y_D-1)^2+(10\sin^2(\pi y_1))\right\},$ $+\sum_{i=1}^{D}u(x_i,10,100,4)$ $y_i=1+\frac{x_i+1}{4}$ $u(x_i,a,k,m)=\begin{cases}k(x_i-a)^m, & x_i>a\\ 0, & -a\le x_i\le a\\ k(-x_i-a)^m, & x_i<-a\end{cases}$	30	$[-50, 50]$	0
Penalized 2	$f_{18}(x)=0.1\left\{\sin^2(3\pi x_1)+\sum_{i=1}^{D-1}(x_i-1)^2[1+\sin^2(3\pi x_{i+1})]+(x_D-1)^2[1+\sin^2(2\pi x_D)]\right\}$ $+\sum_{i=1}^{D}u(x_i,5,100,4)$	30	$[-50, 50]$	0
Cosine Mixture	$f_{19}(x)=0.1\sum_{i=1}^{D}\cos(5\pi x_i)-\sum_{i=1}^{D}x_i^2$	4	$[-1, 1]$	−0.4
Kowalik	$f_{20}(x)=\sum_{i=1}^{11}\left[a_i-\left(x_1(b_i^2+b_ix_2)\right)/(b_i^2+b_ix_3+x_4)\right]^2$	4	$[-5, 5]$	0.0003075
Six-hump Camel-back	$f_{21}(x)=4x_1^2-2.1x_1^4+\frac{1}{3}x_1^6+x_1x_2-4x_2^2+4x_2^4$	2	$[-5, 5]$	−1.0316285
Branin	$f_{22}(x)=\left(x_2-\frac{5.1}{4\pi^2}x_1^2+\frac{5}{\pi}x_1-6\right)^2+10\left(1-\frac{1}{8\pi}\right)\cos x_1+10$	2	$[-5, 10]$	0.39789
Goldstein–Price	$f_{23}(x)=[1+(x_1+x_2+1)^2(19-14x_1+3x_1^2-14x_2+6x_1x_2+3x_2^2)]$ $\cdot[30+(2x_1-3x_2)^2(18-32x_1+12x_1^2+48x_2-36x_1x_2+27x_2^2)]$	2	$[-2, 2]$	3

In the following experimental studies, two criteria are used to measure the performance of each algorithm: (1) mean values of the number of function evaluations (FES) and success rate (SR); (2) quality of the final solutions. For the first criterion, when the function error value $(f(x)-f(x^*))$ is below the predefined accuracy level (δ) within the MaxFES, it is deemed a success run, and the FES is recorded. In this experiment, MaxFES is set to 300 000, and δ is set to 1E − 05. For the second criterion, the mean and standard deviation of the function error value $(f(x)-f(x^*))$ are recorded when the number of function evaluations (FES) reaches the maximum FES (MaxFES), where x is the best solution found so far and x^* is the global optimum of the benchmark function that is presented in Table 1. Additionally, the Wilcoxon Signed Rank Test [49] is also adopted to study the difference between any two algorithms in a meaningful way. The results are based on the solutions obtained in the 30 independent runs. Here, three signs (+ , − , ≈) are used to

Table 2
Results of mean values of FES and SR on SaDE, EPSDE, JADE, CoDE, SHADE, and DELU under predefined accuracy level for all benchmark functions.

Fun	D	SaDE		EPSDE		JADE		CoDE		SHADE		DELU	
		FES	SR (%)	FES	SR (%)	FES	SR (%)	FES	SR (%)	FES	SR (%)	FES	SR (%)
f_1	30	2.03E+04	100	1.67E+04	100	2.36E+04	100	4.02E+04	100	2.35E+04	100	**1.22E+04**	100
f_2	30	1.84E+04	100	1.51E+04	100	2.16E+04	100	3.63E+04	100	2.16E+04	100	**1.07E+04**	100
f_3	30	2.44E+04	100	**2.32E+04**	100	3.58E+04	100	5.64E+04	100	3.42E+04	100	3.15E+04	100
f_4	30	1.10E+04	100	9.36E+03	100	1.34E+04	100	2.24E+04	100	1.67E+04	100	**7.98E+03**	100
f_5	30	2.11E+04	100	**1.83E+04**	100	2.70E+04	100	4.13E+04	100	2.67E+04	100	2.55E+04	100
f_6	30	1.07E+04	100	**8.33E+03**	100	1.18E+04	100	2.01E+04	100	1.19E+04	100	1.16E+04	100
f_7	30	1.37E+05	100	1.33E+05	100	**5.61E+04**	97	7.80E+04	100	5.98E+04	100	7.71E+04	100
f_8	30	NA	0	1.16E+05	87	9.78E+04	100	2.37E+05	100	1.09E+05	100	**7.09E+04**	100
f_9	30	2.17E+04	73	1.76E+04	83	2.69E+04	97	4.39E+04	100	2.56E+04	100	**1.54E+04**	100
f_{10}	30	**7.18E+04**	100	1.13E+05	100	2.80E+05	100	1.73E+05	100	1.08E+05	97	9.78E+04	100
f_{11}	30	3.61E+04	100	3.71E+04	100	8.27E+04	97	6.00E+04	100	9.67E+04	100	**1.34E+04**	100
f_{12}	30	2.29E+04	100	2.33E+04	100	5.65E+04	100	3.56E+04	100	3.35E+04	97	**1.98E+04**	100
f_{13}	30	1.23E+04	100	1.30E+04	100	1.73E+04	100	2.63E+04	100	1.67E+04	100	**1.05E+04**	100
f_{14}	30	1.29E+04	100	1.15E+04	100	1.74E+04	100	2.70E+04	100	1.71E+04	100	**8.71E+03**	100
f_{15}	30	3.05E+04	93	**2.32E+04**	100	3.31E+04	100	5.68E+04	100	3.21E+04	100	2.65E+04	100
f_{16}	30	6.25E+04	93	6.40E+04	100	1.09E+05	100	1.30E+05	97	1.24E+05	100	**5.94E+04**	100
f_{17}	30	1.49E+04	100	1.29E+04	100	1.97E+04	100	3.03E+04	100	1.92E+04	100	**9.17E+03**	100
f_{18}	30	1.74E+04	87	1.59E+04	97	2.24E+04	100	3.54E+04	100	**1.50E+04**	100	1.52E+04	100
f_{19}	4	3.04E+03	100	2.71E+03	100	2.60E+03	100	3.42E+03	100	4.93E+03	100	**2.05E+03**	100
f_{20}	4	7.58E+03	100	7.32E+03	100	5.99E+03	100	6.97E+03	100	1.47E+04	100	**5.51E+03**	100
f_{21}	2	2.29E+03	100	2.14E+03	100	2.00E+03	100	2.30E+03	100	3.96E+03	100	**1.41E+03**	100
f_{22}	2	2.57E+03	100	2.02E+03	100	2.15E+03	100	2.45E+03	100	4.21E+03	100	**1.76E+03**	100
f_{23}	2	2.24E+03	100	2.11E+03	100	1.94E+03	100	2.15E+03	100	3.87E+03	100	**1.87E+03**	100
Total Average		4.70E+04	93.3	3.73E+04	98.6	5.29E+04	99.6	6.39E+04	99.9	4.39E+04	99.7	**2.91E+04**	**100**

Table 3
Mean function error values and standard deviation of SaDE, EPSDE, JADE, CoDE, SHADE, and DELU within the MaxFEs for all benchmark functions.

Fun	D	SaDE Mean Error(Std Dev)	EPSDE Mean Error(Std Dev)	JADE Mean Error(Std Dev)	CoDE Mean Error(Std Dev)	SHADE Mean Error(Std Dev)	DELU Mean Error(Std Dev)
f_1	30	1.29E−23 (3.07E−23)$^{+}$	3.87E−31 (9.46E−31)$^{+}$	3.01E−20 (8.74E−20)$^{+}$	2.16E−10 (1.73E−10)$^{+}$	2.00E−22 (2.54E−22)$^{+}$	**9.25E−37** (**2.99E−36**)
f_2	30	6.59E−25 (8.39E−25)$^{+}$	5.52E−31 (1.94E−30)$^{\approx}$	1.16E−21 (1.59E−21)$^{+}$	2.83E−11 (2.61E−11)$^{+}$	2.40E−23 (2.42E−23)$^{+}$	**2.56E−32** (**4.00E−32**)
f_3	30	8.70E−16 (5.17E−16)$^{-}$	**1.29E−16** (**2.15E−16**)$^{-}$	3.11E−10 (3.36E−10)$^{+}$	3.86E−06 (1.32E−06)$^{+}$	1.78E−11 (5.89E−12)$^{\approx}$	1.46E−11 (1.23E−11)
f_4	30	**0.00E+00** (**0.00E+00**)$^{\approx}$	**0.00E+00** (**0.00E+00**)$^{\approx}$	**0.00E+00** (**0.00E+00**)$^{\approx}$	1.30E−14 (1.13E−14)$^{\approx}$	7.40E−18 (2.82E−17)$^{+}$	**0.00E+00** (**0.00E+00**)
f_5	30	8.85E−23 (4.35E−22)$^{-}$	**8.85E−30** (**1.75E−29**)$^{-}$	1.83E−18 (6.07E−18)$^{+}$	4.04E−10 (3.04E−10)$^{+}$	5.60E−21 (4.23E−21)$^{-}$	1.65E−20 (4.70E−20)
f_6	30	**0.00E+00** (**0.00E+00**)$^{\approx}$	**0.00E+00** (**0.00E+00**)$^{\approx}$	**0.00E+00** (**0.00E+00**)$^{\approx}$	**0.00E+00** (**0.00E+00**)$^{\approx}$	**0.00E+00** (**0.00E+00**)$^{\approx}$	**0.00E+00** (**0.00E+00**)
f_7	30	9.27E−02 (1.30E−01)$^{+}$	1.42E+01 (2.25E+01)$^{+}$	6.16E−03 (3.13E−02)$^{+}$	5.69E−04 (7.53E−04)$^{+}$	2.01E−03 (1.07E−02)$^{+}$	**1.82E−04** (**3.39E−04**)
f_8	30	5.07E+01 (3.36E+01)$^{+}$	8.61E+00 (2.09E+00)$^{+}$	9.29E+00 (1.06E+00)$^{+}$	1.97E+01 (5.80E−01)$^{+}$	1.31E+01 (7.07E−01)$^{+}$	**1.28E−04** (**1.36E−04**)
f_9	30	2.22E−03 (4.73E−03)$^{+}$	6.57E−04 (2.50E−03)$^{+}$	9.70E−15 (5.25E−14)$^{+}$	2.47E−07 (7.34E−07)$^{+}$	**0.00E+00** (**0.00E+00**)$^{\approx}$	**0.00E+00** (**0.00E+00**)
f_{10}	30	**7.51E−04** (**7.06E−04**)$^{-}$	2.35E−01 (1.31E−01)$^{+}$	3.24E+00 (1.23E+00)$^{+}$	1.97E+00 (4.24E−01)$^{+}$	1.41E−01 (1.82E−02)$^{+}$	2.03E−02 (6.76E−03)
f_{11}	30	**0.00E+00** (**0.00E+00**)$^{\approx}$	**0.00E+00** (**0.00E+00**)$^{\approx}$	1.07E+01 (2.20E+01)$^{+}$	1.66E−02 (3.13E−02)$^{+}$	1.31E+02 (5.56E+01)$^{+}$	**0.00E+00** (**0.00E+00**)
f_{12}	30	**0.00E+00** (**0.00E+00**)$^{\approx}$	**0.00E+00** (**0.00E+00**)$^{\approx}$	**0.00E+00** (**0.00E+00**)$^{\approx}$	**0.00E+00** (**0.00E+00**)$^{\approx}$	3.14E−02 (1.72E−01)$^{+}$	**0.00E+00** (**0.00E+00**)
f_{13}	30	2.47E−27 (4.90E−27)$^{+}$	2.91E−29 (1.14E−28)$^{+}$	3.69E−21 (1.55E−20)$^{+}$	1.80E−13 (3.36E−13)$^{+}$	8.43E−25 (8.43E−25)$^{+}$	**2.19E−30** (**2.11E−30**)
f_{14}	30	1.46E−03 (3.80E−03)$^{+}$	1.75E−32 (9.04E−33)$^{+}$	5.07E−22 (2.22E−21)$^{+}$	1.31E−13 (1.51E−13)$^{+}$	1.46E−24 (1.71E−24)$^{+}$	**1.36E−32** (**3.13E−34**)
f_{15}	30	2.40E−01 (4.45E−01)$^{+}$	**6.04E−15** (**1.66E−15**)$^{-}$	4.19E−11 (4.49E−11)$^{-}$	3.75E−06 (1.88E−06)$^{+}$	2.30E−12 (1.02E−12)$^{-}$	2.42E−10 (1.35E−10)
f_{16}	30	4.11E−02 (1.82E−01)$^{+}$	5.48E−02 (1.39E−01)$^{+}$	4.76E+00 (8.78E−01)$^{+}$	3.30E+01 (5.81E+00)$^{+}$	1.63E+01 (2.16E+00)$^{+}$	**1.99E−06** (**2.50E−06**)
f_{17}	30	3.46E−03 (1.89E−02)$^{+}$	**6.02E−32** (**1.48E−31**)$^{-}$	5.65E−21 (1.54E−20)$^{+}$	1.44E−12 (1.26E−12)$^{+}$	8.71E−24 (1.19E−23)$^{+}$	4.57E−26 (7.30E−26)
f_{18}	30	1.83E−03 (4.16E−03)$^{+}$	3.66E−04 (2.01E−03)$^{+}$	3.60E−20 (7.81E−20)$^{+}$	2.45E−11 (2.36E−11)$^{+}$	**1.55E−22** (**1.44E−22**)$^{-}$	3.80E−21 (4.03E−21)
f_{19}	4	**0.00E+00**(**0.00E+00**)$^{\approx}$	**0.00E+00** (**0.00E+00**)$^{\approx}$	**0.00E+00** (**0.00E+00**)$^{\approx}$	**0.00E+00** (**0.00E+00**)$^{\approx}$	**0.00E+00** (**0.00E+00**)$^{\approx}$	**0.00E+00** (**0.00E+00**)
f_{20}	4	9.38E−06 (1.95E−05)$^{\approx}$	2.26E−05 (5.70E−05)$^{\approx}$	1.37E−03 (5.08E−03)$^{+}$	9.60E−06 (4.06E−05)$^{\approx}$	3.10E−04 (7.47E−05)$^{+}$	**5.52E−06** (**1.12E−05**)
f_{21}	2	**0.00E+00** (**0.00E+00**)$^{\approx}$	**0.00E+00** (**0.00E+00**)$^{\approx}$	**0.00E+00** (**0.00E+00**)$^{\approx}$	**0.00E+00** (**0.00E+00**)$^{\approx}$	**0.00E+00** (**0.00E+00**)$^{\approx}$	**0.00E+00 0.00E+00**
f_{22}	2	4.67E−07 (1.17E−06)$^{+}$	**0.00E+00** (**0.00E+00**)$^{\approx}$	**0.00E+00** (**0.00E+00**)$^{\approx}$	1.33E−07 (5.71E−07)$^{+}$	2.14E−05 (2.10E−05)$^{+}$	**0.00E+00** (**0.00E+00**)
f_{23}	2	**0.00E+00** (**0.00E+00**)$^{\approx}$	**0.00E+00** (**0.00E+00**)$^{\approx}$	**0.00E+00** (**0.00E+00**)$^{\approx}$	**0.00E+00** (**0.00E+00**)$^{\approx}$	**0.00E+00** (**0.00E+00**)$^{+}$	**0.00E+00** (**0.00E+00**)
$w/t/l$		12/8/3	9/10/4	15/7/1	16/7/0	15/5/3	–/–/–

indicate the results, where the "+" means the first algorithm is significantly better than the second, the "−" means the first algorithm is significantly worse than the second, and the "≈" means that there is no significant difference between the two algorithms. In this experiment, the value of MaxFES of all algorithms is set to $2000 \times D$, where D is the number of dimension, and the significance level (α) is set to 0.05. The control parameters for the competitive algorithms are set the same as in their original literatures.

Table 2 summarizes the results of the mean number of FES and SR of each benchmark functions, where “NA” means that no runs of the corresponding algorithm found the solution below the predefined accuracy level. As seen, SaDE converges faster than the other algorithms only on one function (f_{10}), and it hardly reaches to the predefined accuracy level on f_8 within the MaxFES. EPSDE shows faster convergence on 4 functions including 3 unimodal functions (f_3, f_5, and f_6) and one high-dimensional multimodal function (f_{15}). JADE requires less FES to reach the predefined accuracy level on f_7, but its success rate is 97% while that of the others all are 100%. For CoDE, it could hardly converge faster than the other competitors on any function. SHADE costs less FES to converge to the predefined accuracy level only on one function (f_{18}). Our proposed approach DELU shows faster convergence speed on 4 unimodal functions (f_1, f_2, f_4, and f_8), 7 high-dimensional multimodal functions (f_9, $f_{11}-f_{14}$, f_{16}, and f_{17}), and all low-dimensional multimodal functions ($f_{19}-f_{23}$) with 100% success rate. Furthermore, from the results of total average FES listed in the last row of Table 2, we can find that DELU costs the lowest FES to reach the predefined accuracy level, and followed by EPSDE. The average number of FES of all the 23 benchmark functions consumed by DELU is 2.91E+04, while that of SaDE, EPSDE, JADE, CoDE, and SHADE are 4.70E+04, 3.73E+04, 5.29E+04, 6.39E+04, and 4.39E+04, respectively. This means that DELU is able to save 38.1%, 22.0%, 45.0%, 54.5%, and 33.8% of FES in comparison to SaDE, EPSDE, JADE, CoDE, and SHADE, respectively. Especially, DELU successfully reaches the predefined accuracy level for all benchmark functions, thus obtains the highest SR. CoDE achieves the second highest SR (99.9%) as it gets 97% SR for f_{16}. The SR of JADE, EPSDE, and SHADE is 99.6%, 98.6%, and 99.7%, respectively. SaDE obtains the lowest SR (93.3%) as it fails to solve one function (f_8).

To further compare the overall performances of the six algorithms on all benchmark functions, the empirical distribution of normalized success performance SP(SP=FES/SR) has been plotted. We first calculate the SP of the six algorithms on each test function according to the results in Table 2, and then normalize the SP by dividing all SPs by the SP of the best algorithm on the respective function. Results of all functions are used where at least one algorithm was successful in at least one run. Small values of SP and large values of the empirical distribution in graphs are preferable. The first one that reaches the top of the graph will be regarded as the best algorithm [42]. From Fig. 3, we can observe that DELU outperforms other approaches on overall performance on all benchmark functions.

Fig. 4 shows the mean convergence graphs of SaDE, EPSDE, JADE, CoDE, SHADE, and DELU on some selected functions (f_1, f_8, f_{14}, and f_{16}). Clearly, DELU always converges faster than the other competitors on these four functions. For unimodal function f_1, DELU has the fastest convergence speed and followed by EPSDE. For function f_8(Rosenbrock) [50], it is multimodal when $D > 3$. It makes the algorithm easy to find the local minimum but difficult to find the global optimum, because its global optimum locates in a long narrow parabolic shaped flat valley. However, due to the invalid region exclusion operation in DELU, it prevents the algorithm from falling into the local optimum to a certain extent. Therefore, DELU can converge to the global optimum steadily while the other three algorithms all get trapped into the local optimum. For multimodal function f_{14}, DELU shows the fastest convergence in the beginning of the optimization process and followed by EPSDE, and they finally get almost the same results. For multimodal function f_{16}, DELU is obviously faster than the other five algorithms.

Table 3 presents the results of mean function error value and the corresponding standard deviation obtained by SaDE, EPSDE, JADE, CoDE, SHADE, and DELU for all benchmark functions, where

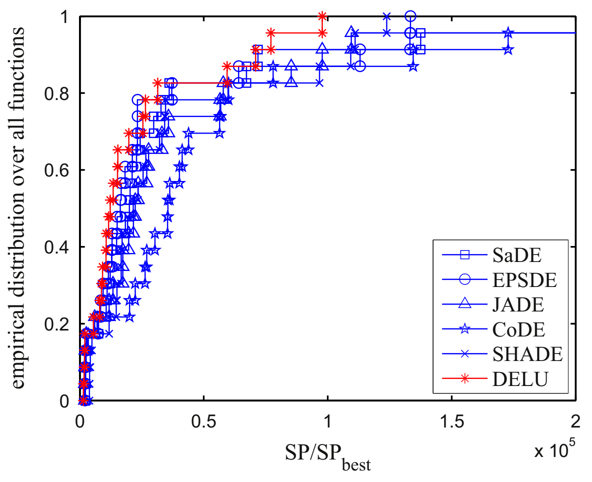
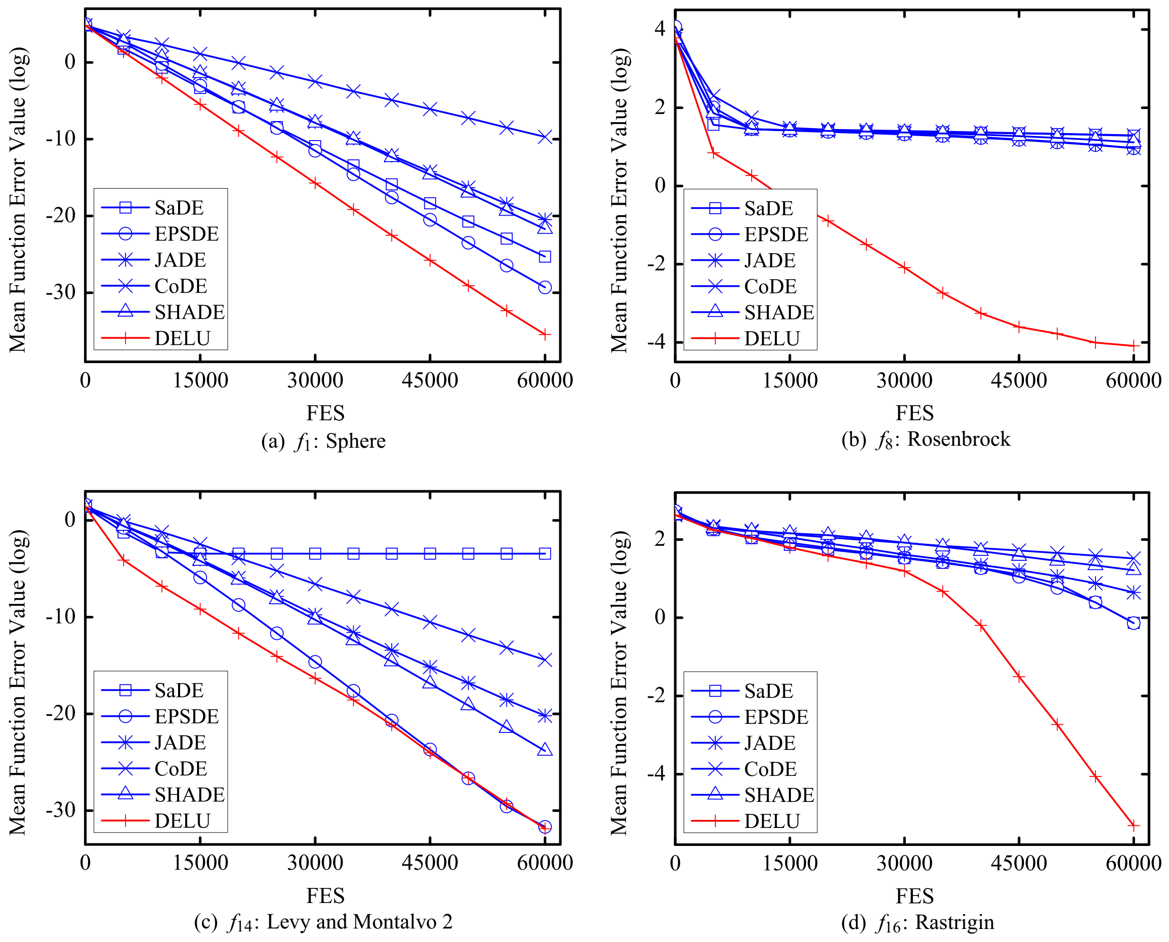


Fig. 3. Empirical distribution of normalized success performance on all 23 benchmark functions.

“Mean Error” indicates the mean value of the best function error of the 30 independent runs, and “Std Dev” indicates the corresponding standard deviation. According to the Wilcoxon Signed Rank Test, the results are summarized as “$w/t/l$” in the last row of the table indicates that DELU wins on w functions, ties on t functions, and loses on l functions, compared with its corresponding competitor. From the results, it can be seen that, our proposed DELU is superior to SaDE, EPSDE, JADE, CoDE, and SHADE on the majority of benchmark functions. DELU outperforms SaDE on 12 functions, ties on 8 functions, and only loses on 3 functions (f_3, f_5, and f_{10}). For high-dimensional unimodal and multimodal functions, EPSDE shows better performance than DELU on 4 functions (f_3, f_5, f_{15}, and f_{17}). But DELU achieves better results than EPSDE on 9 out of the other 14 high-dimensional functions. For low-dimensional multimodal functions, EPSDE and DELU almost have the same performance except for f_{20}. Compared to DELU, although JADE wins on one function (f_{15}) and also obtains the same results on all low-dimensional multimodal functions except for f_{20}, DELU wins on 15 functions. CoDE shows the same performance as DELU on f_6, f_{12}, f_{19}, f_{21}, and f_{23}, but it is not superior to DELU on any function, while DELU outperforms it on 16 functions. For SHADE, it performs better than DELU on three functions (f_5, f_{15}, and f_{18}), while DELU is superior to SHADE on 15 functions.

Table 4 shows the mean and standard deviation of the final best function error achieved by SaDE, EPSDE, JADE, CoDE, SHADE, and DELU for functions f_1-f_{18} at $D=50$. As can be seen, our proposed DELU performs better than the other five competitors on the majority of benchmark functions. To be specific, DELU outperforms SaDE on 14 out of 18 functions, while SaDE performs better than DELU only on one function (f_5). EPSDE is superior to DELU on 3 functions including 2 unimodal functions (f_2 and f_5) and one multimodal function (f_{17}), but DELU shows better performance than EPSDE on 13 out of 18 functions. JADE achieves better results than DELU on 3 functions (f_2, f_5, and f_{17}), while DELU outperforms JADE on 15 out of 18 functions. CoDE is not superior to DELU on any function, while DELU obtains the better performance than CoDE on 17 function. Compared to SHADE, DELU wins on 10 functions, ties on 3 functions, and loses on 5 functions (f_2, f_5, f_{14}, f_{15}, and f_{17}).

In summary, based on the above results and analysis, DELU is the best algorithm in comparison with the five state-of-the-art DE variants. It is capable of providing better quality solutions with less FES than the other five competitors on most of the benchmark functions.



(a) f_1: Sphere (b) f_8: Rosenbrock (c) f_{14}: Levy and Montalvo 2 (d) f_{16}: Rastrigin

Fig. 4. Mean convergence curves of SaDE, EPSDE, JADE, CoDE, SHADE, and DELU on functions f_1, f_8, f_{14}, and f_{16}.

Table 4
Mean function error values and standard deviation of SaDE, EPSDE, JADE, CoDE, SHADE, and DELU within the MaxFEs for functions f_1–f_{18} at D=50.

Fun	D	SaDE Mean Error(Std Dev)	EPSDE Mean Error(Std Dev)	JADE Mean Error(Std Dev)	CoDE Mean Error(Std Dev)	SHADE Mean Error(Std Dev)	DELU Mean Error(Std Dev)
f_1	50	9.21E−26 (9.49E−26)$^{+}$	1.70E−32 (5.67E−32)$^{+}$	5.40E−16 (1.42E−15)$^{+}$	1.57E−12 (1.40E−12)$^{+}$	1.12E−33 (1.04E−33)$^{+}$	**4.05E−34** (**4.50E−34**)
f_2	50	1.22E−26 (1.24E−26)$^{+}$	3.57E−32 (1.70E−31)$^{-}$	5.54E−16 (1.24E−15)$^{+}$	1.46E−13 (7.58E−14)$^{+}$	**5.03E−34** (**5.23E−34**)$^{-}$	1.95E−27 (1.33E−27)
f_3	50	1.64E−16 (8.95E−17)$^{+}$	3.27E−18 (1.49E−17)$^{+}$	**6.46E−27** (**1.29E−26**)$^{-}$	1.55E−07 (7.15E−08)$^{+}$	3.42E−17 (3.52E−17)$^{+}$	1.12E−19 (9.84E−20)
f_4	50	**0.00E+00** (**0.00E+00**)$^{\approx}$	1.63E−16 (8.11E−17)$^{+}$	1.55E−16 (7.76E−17)$^{+}$	4.62E−15 (4.04E−15)$^{+}$	1.11E−16 (0.00E+00)$^{+}$	**0.00E+00** (**0.00E+00**)
f_5	50	4.28E−25 (6.16E−25)$^{-}$	**1.34E−30** (**4.58E−30**)$^{-}$	5.51E−25 (1.19E−24)$^{-}$	1.26E−12 (1.19E−12)$^{+}$	3.07E−29 (4.12E−29)$^{-}$	2.87E−18 (7.54E−18)
f_6	50	**0.00E+00** (**0.00E+00**)$^{\approx}$	7.00E−01 (1.26E+00)$^{+}$	6.30E+00 (6.07E+00)$^{+}$	**0.00E+00** (**0.00E+00**)$^{\approx}$	**0.00E+00** (**0.00E+00**)$^{\approx}$	**0.00E+00** (**0.00E+00**)
f_7	50	9.15E−02 (5.88E−02)$^{+}$	3.00E+02 (6.05E+01)$^{+}$	1.32E+01 (3.21E+01)$^{+}$	4.32E−02 (4.36E−02)$^{+}$	9.83E−03 (1.81E−03)$^{+}$	**4.65E−03** (**4.71E−03**)
f_8	50	3.81E+01 (2.16E+00)$^{+}$	3.03E+01 (2.15E+00)$^{+}$	9.90E+00 (3.18E+00)$^{+}$	3.86E+01 (4.03E−01)$^{+}$	2.51E+01 (1.31E+00)$^{+}$	**2.60E−05** (**1.82E−05**)
f_9	50	7.61E−03 (1.69E−02)$^{+}$	3.04E−03 (5.74E−03)$^{+}$	2.71E−03 (5.97E−03)$^{+}$	9.35E−13 (4.92E−13)$^{+}$	**0.00E+00** (**0.00E+00**)$^{\approx}$	**0.00E+00** (**0.00E+00**)
f_{10}	50	6.79E−01 (1.73E+00)$^{+}$	4.58E−01 (3.27E−01)$^{\approx}$	1.16E+00 (3.57E+00)$^{+}$	9.79E−01 (2.02E−01)$^{+}$	9.43E−01 (1.26E−01)$^{+}$	**4.36E−01** (**1.83E−01**)
f_{11}	50	2.37E+01 (4.99E+01)$^{+}$	7.64E+02 (2.67E+03)$^{+}$	4.97E+02 (2.00E+02)$^{+}$	1.89E+01 (2.53E+01)$^{+}$	2.15E+02 (6.17E+01)$^{+}$	**0.00E+00** (**0.00E+00**)
f_{12}	50	**0.00E+00** (**0.00E+00**)$^{\approx}$	1.89E−02 (1.03E−01)$^{+}$	2.94E+00 (1.03E+00)$^{+}$	3.14E−05 (4.62E−12)$^{+}$	2.26E−01 (3.51E−01)$^{+}$	**0.00E+00** (**0.00E+00**)
f_{13}	50	9.17E−30 (9.26E−30)$^{+}$	8.74E−31 (2.70E−30)$^{+}$	6.22E−03 (1.97E−02)$^{+}$	1.28E−16 (1.15E−16)$^{+}$	5.01E−32 (9.37E−32)$^{\approx}$	**3.17E−32** (**2.63E−32**)
f_{14}	50	1.10E−03 (3.47E−03)$^{+}$	7.01E−04 (3.84E−03)$^{+}$	3.20E−03 (7.15E−03)$^{+}$	5.42E−16 (3.49E−16)$^{+}$	**2.92E−32** (**1.81E−33**)$^{-}$	2.26E−29 (4.02E−29)
f_{15}	50	8.40E−01 (6.12E−01)$^{+}$	1.49E−14 (4.32E−15)$^{\approx}$	1.41E+00 (3.75E−01)$^{+}$	1.74E−07 (9.59E−08)$^{+}$	**7.11E−15** (**0.00E+00**)$^{-}$	1.82E−14 (5.02E−15)
f_{16}	50	4.99E+00 (4.75E+00)$^{+}$	4.77E+01 (1.34E+01)$^{+}$	9.95E−02 (3.15E−01)$^{+}$	8.86E+01 (1.10E+01)$^{+}$	2.06E+01 (1.46E+00)$^{+}$	**1.64E−06** (**3.89E−07**)
f_{17}	50	2.49E−02 (3.21E−02)$^{+}$	1.05E−32 (4.55E−33)$^{-}$	1.38E−32 (5.69E−33)$^{-}$	1.89E−15 (1.56E−15)$^{+}$	**9.53E−33** (**5.66E−34**)$^{-}$	6.65E−20 (4.21E−20)
f_{18}	50	2.20E−03 (4.63E−03)$^{+}$	1.46E−03 (8.02E−03)$^{+}$	2.33E−02 (7.00E−02)$^{+}$	7.16E−14 (7.83E−14)$^{+}$	3.92E−21 (5.78E−22)$^{+}$	**2.31E−22**(**1.93E−22**)
$w/t/l$		14/3/1	13/2/3	15/0/3	17/1/0	10/3/5	–/–/–

5.3. Effect of the local abstract convex underestimate strategy on advanced DE variants

In this section, in order to demonstrate the effectiveness of the proposed local abstract convex underestimate strategy, we incorporate it into some advanced DE variants: SaDE [42], JADE [43], and CoDE [44]. The variants of these above considered algorithms using local abstract convex underestimate strategy are denoted as SaDELU, JADELU, and CoDELU, respectively. To make a fair comparison, all parameters of the above advanced DEs and their corresponding local abstract convex underestimate strategy-based variants are kept the same as used in their original paper. The

Table 5
Mean function error values and standard deviation of SaDE, JADE, CoDE, and their corresponding variants with local abstract convex underestimate strategy SaDELU, JADELU, CoDELU within the MaxFEs for all benchmark functions.

Fun	D	SaDE Mean Error(Std Dev)	SaDELU Mean Error(Std Dev)	JADE Mean Error(Std Dev)	JADELU Mean Error(Std Dev)	CoDE Mean Error(Std Dev)	CoDELU Mean Error(Std Dev)
f_1	30	1.29E−23 (3.07E−23)	**2.10E−24 (1.95E−24)**$^{+}$	3.01E−20 (8.74E−20)	**9.96E−24 (1.99E−23)**$^{+}$	2.16E−10 (1.73E−10)	**5.38E−15 (3.87E−15)**$^{+}$
f_2	30	6.59E−25 (8.39E−25)	**1.27E−28 (1.23E−28)**$^{+}$	1.16E−21 (1.59E−21)	**9.95E−25 (1.05E−24)**$^{+}$	2.83E−11 (2.61E−11)	**5.68E−17 (4.55E−17)**$^{+}$
f_3	30	**8.70E−16 (5.17E−16)**	5.28E−15 (3.01E−15)$^{-}$	3.11E−10 (3.36E−10)	**1.62E−12 (9.12E−13)**$^{+}$	3.86E−06 (1.32E−06)	**2.77E−08 (8.62E−09)**$^{+}$
f_4	30	**0.00E+00 (0.00E+00)**	**0.00E+00 (0.00E+00)**$^{\approx}$	**0.00E+00 (0.00E+00)**	**0.00E+00 (0.00E+00)**$^{\approx}$	1.30E−14 (1.13E−14)	**0.00E+00 (0.00E+00)**$^{+}$
f_5	30	8.85E−23 (4.35E−22)	**3.72E−26 (2.82E−26)**$^{+}$	1.83E−18 (6.07E−18)	**1.47E−21 (2.46E−21)**$^{+}$	**4.04E−10 (3.04E−10)**	6.73E−07 (1.29E−09)$^{-}$
f_6	30	**0.00E+00 (0.00E+00)**	**0.00E+00 (0.00E+00)**$^{\approx}$	**0.00E+00 (0.00E+00)**	**0.00E+00 (0.00E+00)**$^{\approx}$	**0.00E+00 (0.00E+00)**	**0.00E+00 (0.00E+00)**$^{\approx}$
f_7	30	9.27E−02 (1.30E−01)	**4.96E−07 (7.11E−07)**$^{+}$	6.16E−03 (3.13E−02)	**5.83E−03 (2.69E−02)**$^{\approx}$	5.69E−04 (7.53E−04)	**4.85E−07 (8.01E−07)**$^{+}$
f_8	30	5.07E+01 (3.36E+01)	**4.61E−06 (4.68E−06)**$^{+}$	9.29E+00 (1.06E+00)	**5.71E−05 (5.45E−05)**$^{+}$	1.97E+01 (5.80E−01)	**2.62E−03 (2.78E−03)**$^{+}$
f_9	30	2.22E−03 (4.73E−03)	**0.00E+00 (0.00E+00)**$^{+}$	9.70E−15 (5.25E−14)	**0.00E+00 (0.00E+00)**$^{+}$	2.47E−07 (7.34E−07)	**1.25E−07 (2.29E−07)**$^{\approx}$
f_{10}	30	**7.51E−04 (7.06E−04)**	8.41E−04 (7.19E−04)$^{\approx}$	3.24E+00 (1.23E+00)	**1.12E−01 (6.12E−03)**$^{+}$	**1.97E+00 (4.24E−01)**	2.43E+00 (4.13E−01)$^{-}$
f_{11}	30	**0.00E+00 (0.00E+00)**	**0.00E+00 (0.00E+00)**$^{\approx}$	1.07E+01 (2.20E+01)	**0.00E+00 (0.00E+00)**$^{+}$	1.66E−02 (3.13E−02)	**0.00E+00 (0.00E+00)**$^{+}$
f_{12}	30	**0.00E+00 (0.00E+00)**	**0.00E+00 (0.00E+00)**$^{\approx}$	**0.00E+00 (0.00E+00)**	**0.00E+00 (0.00E+00)**$^{\approx}$	**0.00E+00 (0.00E+00)**	**0.00E+00 (0.00E+00)**$^{\approx}$
f_{13}	30	2.47E−27 (4.90E−27)	**4.80E−31 (3.48E−31)**$^{+}$	3.69E−21 (1.55E−20)	**5.21E−27 (6.37E−27)**$^{+}$	1.80E−13 (3.36E−13)	**8.77E−18 (5.32E−18)**$^{+}$
f_{14}	30	1.46E−03 (3.80E−03)	**4.28E−30 (3.46E−30)**$^{+}$	5.07E−22 (2.22E−21)	**9.17E−25 (1.80E−24)**$^{+}$	1.31E−13 (1.51E−13)	**2.62E−19 (2.49E−19)**$^{+}$
f_{15}	30	2.40E−01 (4.45E−01)	**3.49E−12 (3.30E−12)**$^{+}$	**4.19E−11 (4.49E−11)**	4.59E−09 (3.14E−09)$^{-}$	3.75E−06 (1.88E−06)	**1.79E−07 (4.40E−08)**$^{+}$
f_{16}	30	4.11E−02 (1.82E−01)	**3.55E−16 (7.94E−16)**$^{+}$	4.76E+00 (8.78E−01)	**6.12E−15 (4.31E−15)**$^{+}$	3.30E+01 (5.81E+00)	**1.84E−06 (7.87E−07)**$^{+}$
f_{17}	30	3.46E−03 (1.89E−02)	**1.33E−25 (1.08E−25)**$^{+}$	5.65E−21 (1.54E−20)	**4.60E−21 (1.43E−20)**$^{\approx}$	**1.44E−12 (1.26E−12)**	1.89E−12 (2.77E−12)$^{\approx}$
f_{18}	30	1.83E−03 (4.16E−03)	**4.94E−19 (1.10E−18)**$^{+}$	**3.60E−20 (7.81E−20)**	1.18E−17 (9.46E−18)$^{-}$	2.45E−11 (2.36E−11)	**1.29E−11 (2.04E−11)**$^{+}$
f_{19}	4	**0.00E+00 (0.00E+00)**	**0.00E+00 (0.00E+00)**$^{\approx}$	**0.00E+00 (0.00E+00)**	**0.00E+00 (0.00E+00)**$^{\approx}$	**0.00E+00 (0.00E+00)**	**0.00E+00 (0.00E+00)**$^{\approx}$
f_{20}	4	9.38E−06 (1.95E−05)	**5.04E−07 (9.86E−07)**$^{+}$	1.37E−03 (5.08E−03)	**5.29E−05 (3.12E−05)**$^{+}$	**9.60E−06 (4.06E−05)**	2.78E−05 (2.70E−05)$^{-}$
f_{21}	2	**0.00E+00 (0.00E+00)**	**0.00E+00 (0.00E+00)**$^{\approx}$	**0.00E+00 (0.00E+00)**	**0.00E+00 (0.00E+00)**$^{\approx}$	**0.00E+00 (0.00E+00)**	**0.00E+00 (0.00E+00)**$^{\approx}$
f_{22}	2	4.67E−07 (1.17E−06)	**0.00E+00 (0.00E+00)**$^{+}$	**0.00E+00 (0.00E+00)**	**0.00E+00 (0.00E+00)**$^{\approx}$	1.33E−07 (5.71E−07)	**0.00E+00 (0.00E+00)**$^{+}$
f_{23}	2	**0.00E+00 (0.00E+00)**	**0.00E+00 (0.00E+00)**$^{\approx}$	**0.00E+00 (0.00E+00)**	**0.00E+00 (0.00E+00)**$^{\approx}$	**0.00E+00 (0.00E+00)**	**0.00E+00 (0.00E+00)**$^{\approx}$
$w/t/l$		–/–/–	14/8/1	–/–/–	12/9/2	–/–/–	13/7/3

mean function error values and the corresponding standard deviations obtained within the MaxFES are recorded. In addition, the Wilcoxon Signed Rank Test is also used to compare the results between two algorithms. In this experiment, the MaxFES is set to $2000 \times D$.

Table 5 summarizes the results obtained by SaDE, JADE, CoDE and their corresponding local abstract convex underestimate strategy-based variants SaDELU, JADELU, and CoDELU for all benchmark functions, where the "$w/t/l$" listed in the last row of the table is the competition results between DE variants and their corresponding local abstract convex underestimate strategy-based variants that are obtained by the Wilcoxon Signed Rank Test. The results indicate that, in the majority of the benchmark function, the local abstract convex underestimate strategy-based DE methods achieve significantly better results compared with their corresponding original DE methods. Specifically, SaDELU exhibits significantly better performance than SaDE on 14 out of 23 functions, while SaDELU achieves a significantly better performance than SaDELU only on one function (f_3). The performance of SaDE has no significant difference from that of SaDELU on 8 functions. The benefit from local abstract convex underestimate strategy is obvious in most of the high-dimensional unimodal functions (f_1, f_2, f_5, f_7, and f_8), 7 high-dimensional multimodal functions (f_9 and $f_{13}-f_{18}$), and 2 low-dimensional multimodal functions (f_{20} and f_{22}). JADE is also remarkably enhanced by the proposed local abstract convex underestimate strategy. JADELU exhibits either similar or better performance on 21 out of 23 functions. JADELU is significantly better than JADE on 5 high-dimensional unimodal functions (f_1-f_3, f_5, and f_8), 6 high-dimensional multimodal functions (f_9-f_{11}, f_{13}, f_{14}, and f_{16}), and one low-dimensional multimodal function (f_{20}), while JADE obtains better results than JADELU only on two functions (f_{15} and f_{18}). For the other functions, there is no significant difference between JADE and JADELU. The proposed local abstract convex underestimate strategy enhances CoDE and demonstrates either similar or significantly better performance. To be specific, CoDELU significantly outperforms CoDE on 13 functions including 6 low-dimensional unimodal functions (f_1-f_4, f_7, and f_8), 6 high-dimensional multimodal functions (f_{11}, $f_{13}-f_{16}$, and f_{18}), and one low-dimensional function (f_{22}). CoDE exhibits significantly better performance than CoDELU only on 3 functions (f_5, f_{10}, and f_{20}). For the other 7 functions, the performance of the two algorithms is not statistically different.

Table 6 shows the results achieved by SaDE, SaDELU, JADE, JADELU, CoDE, and CoDELU for benchmark functions f_1-f_{18} at $D=50$. It is clear that the DE approaches which are based on the local abstract convex underestimate strategy also consistently outperform their original DE methods in the majority of the benchmark functions. SaDELU exhibits either significantly similar or better performance on 17 out of 18 functions. SaDE is significantly superior to SaDELU only on one function (f_7). Particularly, SaDELU is significantly better than SaDE on 13 functions including 5 unimodal functions (f_1-f_3, f_5, and f_8), and 8 multimodal functions (f_9, f_{11}, and $f_{13}-f_{18}$). JADELU either enhances JADE or performs equally well on 15 out of 18 functions. In more detail, for the unimodal functions, JADELU significantly outperforms JADE on f_1, f_2, f_5, f_6, and f_8. For the multimodal functions, JADELU performs significantly better than JADE on f_9-f_{11}, $f_{13}-f_{16}$, and f_{18}. JADE is significantly better than JADELU only on f_3, f_{12}, and f_{17}. CoDELU shows either similar or better performance on 15 out of 18 functions. Specially, CoDELU exhibits significantly better performance than CoDE on 5 unimodal functions (f_1, f_2, f_4, f_7, and f_8) and 7 multimodal functions (f_9, $f_{11}-f_{13}$, and $f_{15}-f_{17}$), while CoDE outperforms CoDELU only on 3 functions (f_3, f_5, and f_{18}).

Overall, from the results shown in Tables 5 and 6, we can conclude that the proposed local abstract convex underestimate strategy is also capable of improving the performance of recently presented advanced DE variants.

5.4. Comparison of DELU with three non-DE algorithms

In this section, the proposed DELU is compared with three advanced non-DE algorithms. The involved algorithms in comparison are listed as follows:

(1) *CLPSO*: The comprehensive learning particle swarm optimizer [51].

Table 6
Mean function error values and standard deviation of SaDE, JADE, CoDE, and their corresponding variants with local abstract convex underestimate strategy SaDELU, JADELU, CoDELU within the MaxFEs for functions $f_1 - f_{18}$ at $D=50$.

Fun	D	SaDE Mean Error(Std Dev)	SaDELU Mean Error(Std Dev)	JADE Mean Error(Std Dev)	JADELU Mean Error(Std Dev)	CoDE Mean Error(Std Dev)	CoDELU Mean Error(Std Dev)
f_1	50	9.21E−26 (9.49E−26)	**1.22E−28 (1.50E−28)**$^{+}$	5.40E−16 (1.42E−15)	**1.50E−22 (2.57E−22)**$^{+}$	1.57E−12 (1.40E−12)	**6.91E−13 (3.59E−13)**$^{+}$
f_2	50	1.22E−26 (1.24E−26)	**3.60E−29 (2.33E−29)**$^{+}$	5.54E−16 (1.24E−15)	**3.62E−27 (7.94E−27)**$^{+}$	1.46E−13 (7.58E−14)	**1.82E−23 (2.39E−23)**$^{+}$
f_3	50	1.64E−16 (8.95E−17)	**1.97E−18 (1.68E−18)**$^{+}$	**6.46E−27 (1.29E−26)**	2.11E−15 (2.44E−15)$^{-}$	**1.55E−07 (7.15E−08)**	1.47E−06 (3.50E−07)$^{-}$
f_4	50	**0.00E+00 (0.00E+00)**	**0.00E+00 (0.00E+00)**$^{\approx}$	1.55E−16 (7.76E−17)	**1.22E−16 (5.97E−16)**$^{\approx}$	4.62E−15 (4.04E−15)	**1.99E−16 (2.29E−16)**$^{+}$
f_5	50	4.28E−25 (6.16E−25)	**1.16E−29 (8.35E−30)**$^{+}$	5.51E−25 (1.19E−24)	**1.21E−26 (7.81E−27)**$^{+}$	**1.26E−12 (1.19E−12)**	1.38E−11 (1.12E−11)$^{-}$
f_6	50	**0.00E+00 (0.00E+00)**	**0.00E+00 (0.00E+00)**$^{\approx}$	6.30E+00 (6.07E+00)	**0.00E+00 (0.00E+00)**$^{+}$	**0.00E+00 (0.00E+00)**	**0.00E+00 (0.00E+00)**$^{\approx}$
f_7	50	**9.15E−02 (5.88E−02)**	9.88E−01 (1.48E+00)$^{-}$	**1.32E+01 (3.21E+01)**	1.52E+01 (3.65E+01)$^{\approx}$	4.32E−02 (4.36E−02)	**1.11E−03 (1.56E−03)**$^{+}$
f_8	50	3.81E+01 (2.16E+00)	**6.06E−06 (1.02E−05)**$^{+}$	9.90E+00 (3.18E+00)	**6.66E−06 (5.73E−06)**$^{+}$	3.86E+01 (4.03E−01)	**9.71E−07 (5.71E−07)**$^{+}$
f_9	50	7.61E−03 (1.69E−02)	**1.48E−03 (3.31E−03)**$^{+}$	2.71E−03 (5.97E−03)	**2.45E−12(5.47E−12)**$^{+}$	9.35E−13 (4.92E−13)	**3.14E−13 (1.50E−13)**$^{+}$
f_{10}	50	6.79E−01 (1.73E+00)	**4.97E−01 (6.95E−01)**$^{\approx}$	1.16E+00 (3.57E+00)	**5.02E−02 (8.70E−02)**$^{+}$	9.79E−01 (2.02E−01)	**5.51E−01 (1.47E−01)**$^{\approx}$
f_{11}	50	2.37E+01 (4.99E+01)	**0.00E+00 (0.00E+00)**$^{+}$	4.97E+02 (2.00E+02)	**0.00E+00 (0.00E+00)**$^{+}$	1.89E+01 (2.53E+01)	**0.00E+00 (0.00E+00)**$^{+}$
f_{12}	50	**0.00E+00 (0.00E+00)**	**0.00E+00 (0.00E+00)**$^{\approx}$	**2.94E+00 (1.03E+00)**	5.67E+00 (4.62E+00)$^{-}$	3.14E−05 (4.62E−12)	**3.29E−06 (5.18E−07)**$^{+}$
f_{13}	50	9.17E−30 (9.26E−30)	**1.65E−31 (2.68E−31)**$^{+}$	6.22E−03 (1.97E−02)	**2.19E−25 (4.91E−25)**$^{+}$	1.28E−16 (1.15E−16)	**2.62E−21 (4.91E−21)**$^{+}$
f_{14}	50	1.10E−03 (3.47E−03)	**3.18E−23 (3.43E−23)**$^{+}$	3.20E−03 (7.15E−03)	**1.09E−21 (6.91E−22)**$^{+}$	5.42E−16 (3.49E−16)	**4.01E−16 (3.03E−16)**$^{\approx}$
f_{15}	50	8.40E−01 (6.12E−01)	**7.38E−12 (7.95E−12)**$^{+}$	1.41E+00 (3.75E−01)	**5.48E−11 (1.26E−11)**$^{+}$	1.74E−07 (9.59E−08)	**2.87E−09 (1.55E−09)**$^{+}$
f_{16}	50	4.99E+00 (4.75E+00)	**3.11E−15 (1.70E−15)**$^{+}$	9.95E−02 (3.15E−01)	**3.55E−15 (5.02E−15)**$^{+}$	8.86E+01 (1.10E+01)	**3.51E−05 (1.82E−06)**$^{+}$
f_{17}	50	2.49E−02 (3.21E−02)	**2.71E−30 (2.50E−30)**$^{+}$	**1.38E−32 (5.69E−33)**	1.61E−24 (3.60E−24)$^{-}$	1.89E−15 (1.56E−15)	**3.93E−20 (4.35E−20)**$^{+}$
f_{18}	50	2.20E−03 (4.63E−03)	**4.83E−22 (2.17E−22)**$^{+}$	2.33E−02 (7.00E−02)	**2.42E−21 (9.48E−22)**$^{+}$	**7.16E−14 (7.83E−14)**	7.47E−11 (3.37E−11)$^{-}$
$w/t/l$		–/–/–	13/4/1	–/–/–	13/2/3	–/–/–	12/3/3

(2) *CMA-ES*: The evolution strategy with covariance matrix adaptation [52].
(3) *GL-25*: The global and local real-coded genetic algorithm based on parent-centric crossover operators [53].

CLPSO, proposed by Liang et al. [51], is a particle swarm optimization (PSO) variant, in which the personal historical best information of all the particles is used to update a particle's velocity. CMA-ES, as a very famous and efficient evolution strategy, is proposed by Hansen et al. [52]. In CMA-ES, a multivariate normal distribution is used to sample the new solutions, and the distribution is updated by using the covariance matrix adaptation (CMA) method. GL-25, proposed by Garcia-Martinez [53], is a variant of real-coded genetic algorithm, which applies parent-centric real-parameter crossover operators to improve the global and local search. In the following experiments, the parameter settings of the above three algorithms are kept the same as in their original papers. For all algorithms, the stopping rule is the number of FES reaches the MaxFES. In this comparison, the value of MaxFES is set to $5000 \times D$.

Table 7 presents the mean function error values (Mean Error) achieved by CLPSO, CMA-ES, GL-25, and DELU for all benchmark functions, where "$w/t/l$" summarized in the last row of the table is the competition results among DELU and other three non-DE algorithms that are obtained by the Wilcoxon Signed Rank Test. It is obvious that our proposed DELU shows significantly better performance than other three approaches on the majority of benchmark functions. To be specific, DELU significantly outperforms CLPSO on 19 functions, while CLPSO does not achieves better results than DELU on any function. CMA-ES performs significantly better than DELU only on f_7, while DELU significantly outperforms CMA-ES on 21 out of 23 functions. DELU shows significantly better performance than GL-25 on 14 out of 23 functions. GL-25 is significantly superior to DELU on 3 high-dimensional unimodal functions (f_2, f_3, and f_5) and almost shows the same performance as DELU on all low-dimensional multimodal functions except for f_{20}. But GL-25 cannot significantly outperform DELU on any high-dimensional multimodal function.

For functions $f_1 - f_{18}$ at $D=50$, Table 8 shows that, in the majority of these functions, the proposed DELU provides significant results compared with other three non-DE algorithms. Specifically,

Table 7
Mean function error values of CLPSO, CMA-ES, GL-25, and DELU within the MaxFEs for all benchmark functions.

Fun	D	CLPSO Mean Error	CMA-ES Mean Error	GL-25 Mean Error	DELU Mean Error
f_1	30	9.13E−14^{+}	1.97E−29^{+}	4.78E−87^{+}	**0.00E+00**
f_2	30	9.79E−15^{+}	3.75E−28^{+}	**2.66E−78**$^{-}$	8.17E−50
f_3	30	4.48E−09^{+}	2.03E−14^{+}	**2.98E−28**$^{-}$	4.94E−23
f_4	30	3.37E−16^{+}	4.81E−17^{+}	**0.00E+00**$^{\approx}$	**0.00E+00**
f_5	30	9.33E−14^{+}	1.50E−24^{+}	**4.38E−85**$^{-}$	2.29E−34
f_6	30	**0.00E+00**$^{\approx}$	**0.00E+00**$^{\approx}$	**0.00E+00**$^{\approx}$	**0.00E+00**
f_7	30	4.73E+00^{+}	**2.95E−27**$^{-}$	3.14E−02^{+}	1.05E−06
f_8	30	1.95E+01^{+}	2.66E−01^{+}	2.21E+01^{+}	**1.18E−07**
f_9	30	2.40E−09^{+}	1.40E−03^{+}	1.61E−15^{+}	**0.00E+00**
f_{10}	30	1.54E−01^{+}	2.50E+02^{+}	2.95E+00^{+}	**3.32E−11**
f_{11}	30	**0.00E+00**$^{\approx}$	5.19E+03^{+}	4.46E+03^{+}	**0.00E+00**
f_{12}	30	**0.00E+00**$^{\approx}$	1.30E+01^{+}	3.14E−05^{+}	**0.00E+00**
f_{13}	30	1.13E−17^{+}	9.13E−01^{+}	8.82E−31^{+}	**1.57E−32**
f_{14}	30	5.94E−17^{+}	1.10E−03^{+}	1.28E−30^{+}	**1.35E−32**
f_{15}	30	9.76E−08^{+}	1.93E+01^{+}	1.09E−13^{+}	**7.55E−15**
f_{16}	30	9.41E−07^{+}	2.20E+02^{+}	3.07E+01^{+}	**0.00E+00**
f_{17}	30	3.49E−16^{+}	1.73E−02^{+}	1.26E+02^{+}	**1.57E−32**
f_{18}	30	2.52E−14^{+}	2.20E−03^{+}	1.97E+03^{+}	**1.35E−32**
f_{19}	4	**0.00E+00**$^{\approx}$	2.46E−02^{+}	**0.00E+00**$^{\approx}$	**0.00E+00**
f_{20}	4	3.15E−04^{+}	3.59E−03^{+}	2.19E−04^{+}	**0.00E+00**
f_{21}	2	**0.00E+00**$^{\approx}$	5.44E−02^{+}	**0.00E+00**$^{\approx}$	**0.00E+00**
f_{22}	2	**0.00E+00**$^{\approx}$	3.99E−01^{+}	**0.00E+00**$^{\approx}$	**0.00E+00**
f_{23}	2	**0.00E+00**$^{\approx}$	1.67E+01^{+}	**0.00E+00**$^{\approx}$	**0.00E+00**
$w/t/l$		19/4/0	21/1/1	14/6/3	–/–/–

CLPSO cannot significantly outperform DELU on any function, while DELU is significantly superior to CLPSO on 17 out of 18 functions. Compared to CMA-ES, DELU performs significantly better on 14 functions including 5 unimodal functions ($f_1 - f_3$, f_5, and f_8) and 9 multimodal functions ($f_{10} - f_{18}$). CMA-ES shows significantly better performance than DELU only on one function (f_7). GL-25 achieves significantly better result than DELU only on one function (f_5), while DELU significantly outperforms GL-25 on 16 functions including 6 unimodal functions ($f_1 - f_4$, f_7, and f_8) and all multimodal functions ($f_9 - f_{18}$).

In order to compare the performance of multiple algorithms on all the benchmark functions, the Friedman test [54] is adopted. The mean rankings of the four comparison algorithms are

Table 8
Mean function error values of CLPSO, CMA-ES, GL-25, and DELU within the MaxFEs for functions f_1-f_{18} at $D=50$.

Fun	D	CLPSO Mean Error	CMA-ES Mean Error	GL-25 Mean Error	DELU Mean Error
f_1	50	$5.41E-07^{+}$	$5.11E-29^{+}$	$1.17E-17^{+}$	**4.24E−42**
f_2	50	$9.28E-08^{+}$	$1.34E-29^{+}$	$1.74E-17^{+}$	**1.92E−32**
f_3	50	$4.11E-05^{+}$	$4.10E-14^{+}$	$6.95E-07^{+}$	**7.87E−29**
f_4	50	$2.54E-05^{+}$	$\mathbf{0.00E+00}^{\approx}$	$5.45E-15^{+}$	**0.00E+00**
f_5	50	$5.71E-07^{+}$	$7.07E+01^{+}$	$\mathbf{3.03E-21}^{-}$	7.64E−21
f_6	50	$\mathbf{0.00E+00}^{\approx}$	$\mathbf{0.00E+00}^{\approx}$	$\mathbf{0.00E+00}^{\approx}$	**0.00E+00**
f_7	50	$7.92E+01^{+}$	$\mathbf{1.28E-26}^{-}$	$8.11E+01^{+}$	1.77E−04
f_8	50	$4.50E+01^{+}$	$9.31E-01^{+}$	$4.31E+01^{+}$	**2.59E−06**
f_9	50	$6.43E-06^{+}$	$\mathbf{0.00E+00}^{\approx}$	$4.52E-13^{+}$	**0.00E+00**
f_{10}	50	$4.60E+00^{+}$	$4.18E+02^{+}$	$2.71E+01^{+}$	**2.28E−02**
f_{11}	50	$5.64E-03^{+}$	$8.88E+03^{+}$	$1.54E+04^{+}$	**0.00E+00**
f_{12}	50	$3.14E-05^{+}$	$1.30E+01^{+}$	$7.54E-02^{+}$	**0.00E+00**
f_{13}	50	$3.57E-11^{+}$	$5.35E-01^{+}$	$3.86E-23^{+}$	**9.42E−33**
f_{14}	50	$3.16E-10^{+}$	$1.46E-03^{+}$	$4.20E-25^{+}$	**1.35E−32**
f_{15}	50	$2.25E-04^{+}$	$1.93E+01^{+}$	$3.80E-10^{+}$	**6.13E−15**
f_{16}	50	$2.27E-02^{+}$	$3.69E+02^{+}$	$2.30E+02^{+}$	**2.45E−07**
f_{17}	50	$1.09E-09^{+}$	$4.15E-03^{+}$	$1.54E+02^{+}$	**3.25E−30**
f_{18}	50	$1.38E-07^{+}$	$2.42E-29^{+}$	$3.94E+03^{+}$	**3.08E−32**
$w/t/l$		17/1/0	14/3/1	16/1/1	–/–/–

summarized in Table 9. As seen, for all benchmark functions at $D=30$ (including the five low-dimensional multimodal functions), the highest mean ranking is obtained by DELU, which is superior to the other three algorithms. GL-25 is the runner up, CLPSO is the third, and CMA-ES is the last. For functions f_1-f_{18} at $D=50$, the performance of the four algorithms can be sorted by mean ranking into the following order: DELU, CMA-ES, GL-25, and CLSO. The proposed DELU obtains the highest mean ranking as it outperforms the other three algorithms.

Fig. 5 shows the mean convergence process of the four contestant algorithms on some selected functions (f_2, f_7, f_{10}, and f_{13}). As seen, the proposed DELU always converges faster than the other three algorithms on all selected multimodal functions (f_{10} and f_{13}). For GL-25, due to its parent-centric crossover operators, it is very efficient for the unmimodal function, so it converges faster than others on unimodal function f_2, and followed by DELU. CMA-ES shows the fastest convergence speed on unimodal function f_7 as it uses the self-adaptation of arbitrary mutation distribution to sample the new solutions. DELU is the runner up on function f_7, while CLPSO and GL-25 get trapped in local minima.

Based on the above results and analysis, we can find that the proposed DELU obtains better results and a faster convergence speed than the three advanced non-DE algorithms for most of the benchmark functions.

5.5. Comparison of DELU with three surrogate-assisted evolutionary algorithms

As the proposed DELU is a function approximation technique, DELU is also compared with three state-of-the-art surrogate-assisted evolutionary algorithms, i.e., GPEME [16], GS-SOMA [55], and SAGA-GLS [56]. GPEME, proposed by Liu et al. [16], uses a surrogate model-aware evolutionary search framework. GS-SOMA is proposed by Zhou et al. [55]. In GS-SOMA, diverse surrogate models are synergistically unified in the evolutionary search. SAGA-GLS, proposed by Zhou et al. [56], uses computationally cheap hierarchical surrogate models to reduce the expensive function evaluations. As shown in Table 1, GPEME has been tested on f_8, f_9, f_{15}, and 2 functions of CEC 2005 [57] in [16], GS-SOMA on f_8, f_9, f_{15}, and 7 functions of CEC 2005 in [55], and SAGA-GLS on f_1, f_8, f_9, f_{15}, and f_{16} in [56]. Their results are listed in Tables 10–12. All values of them are from their original papers.

Table 10 shows the mean function error values and standard deviation obtained by GPEME and DELU in 1000 FES over 20 independent runs. For f_8, DELU outperforms GPEME at $D=20$, 30, and 50. Although GPEME achieves better result than DELU for f_9 at $D=20$, DELU is superior to GPEME at $D=30$ and 50. For f_{15}, DELU also obtains better results than GPEME at $D=30$ and 50, while GPEME performs better than DELU only at $D=20$. For the other 2 functions in [57], GPEME achieves better result than DELU on F_{10}, while DELU reaches better result than GPEME on F_{19}. In total, DELU outperforms GPEME on 8 out of 11 functions, while GPEME is better than DELU only on 3 functions.

Table 11 presents the mean function error values and standard deviation achieved by GS-SOMA and DELU in 8000 FES over 20 independent runs. It is clear that DELU outperforms GS-SOMA on 7 out of 10 functions (f_8, f_9, f_{15}, and shifted and rotated functions F_{10}, F_{13}, F_{15}, and F_{19} in [57]). Specifically, DELU is better than GS-SOMA by 2 orders of magnitude for f_{15} in terms of mean function error. GS-SOMA is superior to GPME only on 3 functions (F_{11}, F_{16}, and F_{23} in [57]).

Table 12 summarizes the mean function error values reached by SAGA-GLS and DELU within 6000 FES over 20 independent runs. From the results, we can find that SAGA-GLS outperforms DELU only on f_1, while DELU shows better performance than SAGA-GLS on 4 out of 5 functions. Note that although SAGA-GLS is superior to DELU by 4 orders of magnitude only on unimodal function f_1, DELU outperforms SAGA-GLS by 2 orders of magnitude on unimodal function f_8 and multimodal function f_{15}. For multimodal function f_9, DELU achieves better result than SAGA-GLS by 3 orders of magnitude.

5.6. Test on CEC 2013 shifted and rotated benchmark functions

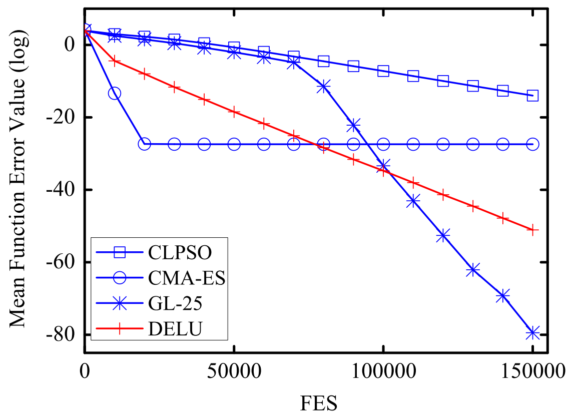
In this section, the CEC 2013 shifted and rotated benchmark functions are conducted to further verify the performance of DELU. Functions F_1-F_5 are unimodal functions. Functions F_6-F_{20} are basic multimodal functions. Functions $F_{21}-F_{28}$ are composition functions. Detailed description of these functions can be found in [58].

In this experiment, DELU is also compared with SaDE [42], EPSDE [45], JADE [43], CoDE [44], and SHADE [48] on all the CEC 2013 benchmark functions. For these algorithms, the same parameter settings as in their original papers are used. The mean function error values $(f(x)-f(x^o))$ and the corresponding standard deviation are recorded, where x is the best solution found so far within the MaxFES, and x^o is the global optimum of the function. In this comparison, the MaxFES is set to $10000\times D$.

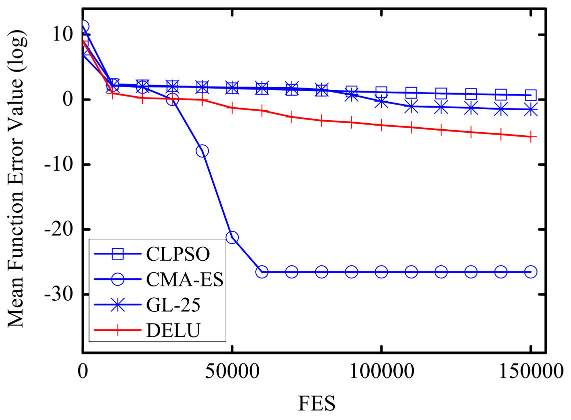
Table 13 summarizes the results achieved by the six compared algorithms for the CEC 2013 benchmark sets at $D=30$, where "Mean Error" and "Std Dev" indicate the mean function values and the corresponding standard deviation, respectively, and the "$w/t/l$" listed in the last row of the table is the competition results between DELU and other competitors according to the Wilcoxon Signed Rank Test. The results reveal that the proposed DELU achieves better results than other contestants on most of the benchmark functions. To be specific, DELU outperforms SaDE on 19 out of 28 functions, while SaDE performs better than DELU only on

Table 9
Mean rankings of CLPSO, CMA-ES, GL-25, and DELU obtained by Friedman test.

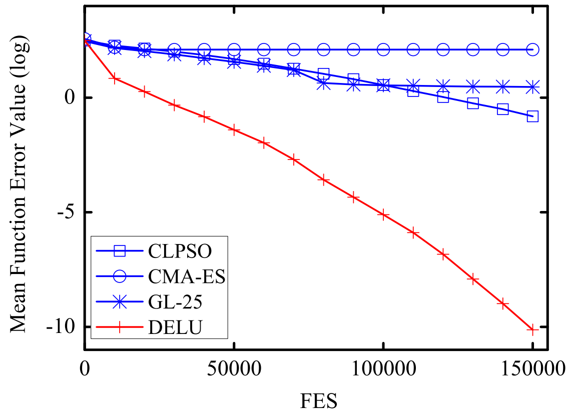
$D=30$		$D=50$	
Algorithms	Mean rankings	Algorithms	Mean rankings
DELU	**3.52**	DELU	**3.75**
GL-25	2.65	CMA-ES	2.19
CLPSO	2.24	GL-25	2.08
CMA-ES	1.59	CLPSO	1.97



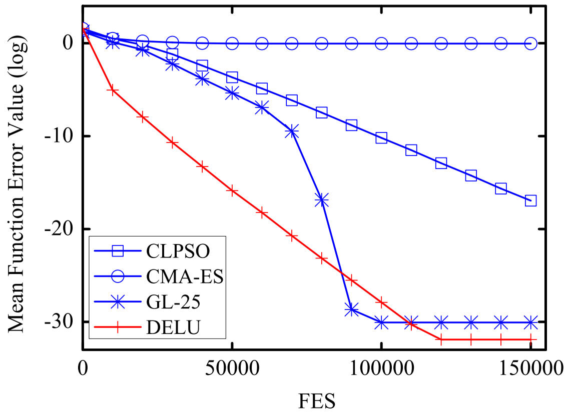
(a) f_2: SumSquares



(b) f_7: Zakharov



(c) f_{10}: Schaffer 2



(d) f_{13}: Levy and Montalvo 1

Fig. 5. Mean convergence curves of CLPSO, CMA-ES, GL-25, and DELU on functions f_2, f_7, f_{10}, and f_{13}.

Table 10
Mean function error values and standard deviation of GPEME and DELU within 1000 FEs.

Fun	D	GPEME Mean Error(Std Dev)	DELU Mean Error(Std Dev)
f_8	20	2.24E+01(8.16E+01)	**2.14E+01(2.57E+00)**
f_8	30	4.62E+01(2.55E+01)	**3.84E+01(1.32E+01)**
f_8	50	2.58E+02(8.02E+01)	**1.13E+02(6.49E+01)**
f_9	20	**3.07E−02(6.82E−02)**	3.43E−02(6.95E−02)
f_9	30	9.97E−01(1.08E−01)	**4.73E−01(3.31E−01)**
f_9	50	3.66E+01(1.32E+01)	**4.16E−01(4.80E−01)**
f_{15}	20	**1.99E−01(5.77E−01)**	4.90E−01(6.63E−01)
f_{15}	30	3.01E+00(9.25E−01)	**1.63E+00(8.19E−01)**
f_{15}	50	1.32E+01(1.58E+00)	**7.93E−01(8.72E−01)**
F_{10} in [57]	30	**3.08E+02(3.64E+01)**	4.84E+02(4.18E+01)
F_{19} in [57]	30	9.49E+02(2.57E+01)	**9.39E+02(2.47E+01)**

Table 11
Mean function error values and standard deviation of GS-SOMA and DELU within 8000 FEs.

Fun	D	GS-SOMA Mean Error(Std Dev)	DELU Mean Error(Std Dev)
f_8	30	4.63E+01 (2.92E+01)	**2.91E+01 (7.78E−01)**
f_9	30	2.20E−03 (4.60E−03)	**1.37E−03 (1.72E−03)**
f_{15}	30	3.58E+00 (5.09E−01)	**4.02E−02 (2.95E−02)**
F_{10} in [57]	30	2.04E+02 (1.60E+02)	**9.57E+01 (2.19E+01)**
F_{11} in [57]	30	**2.90E+01 (3.05E+00)**	4.11E+01 (8.59E−01)
F_{13} in [57]	30	1.80E+01 (1.05E+00)	**1.52E+01 (1.02E+00)**
F_{15} in [57]	30	4.87E+02 (3.06E+01)	**4.12E+02 (2.70E+01)**
F_{16} in [57]	30	**2.05E+02 (1.17E+02)**	2.56E+02 (7.51E+00)
F_{19} in [57]	30	9.32E+02 (1.75E+01)	**8.87E+02 (1.61E+01)**
F_{23} in [57]	30	**6.50E+02 (7.85E+01)**	6.66E+02 (2.06E+02)

Table 12
Mean function error values of SAGA-GLS and DELU within 6000 FEs.

Fun	D	SAGA-GLS Mean Error	DELU Mean Error
f_1	20	**7.94E−10**	6.07E−06
f_8	20	1.91E+03	**1.18E+01**
f_9	20	1.26E+00	**8.33E−03**
f_{15}	20	1.50E−01	**3.13E−03**
f_{16}	20	6.31E+02	**8.18E+01**

one function (F_2). EPSDE obtains better results than DELU only on F_{14}, while DELU is better than EPSDE on 22 out of 28 functions. JADE exhibits better performance than DELU on 4 functions (F_2, F_{14}, F_{22}, and F_{23}), while DELU is superior to JADE on 16 out of 28 functions. Compared with CoDE, DELU shows better performance on 20 functions, while CoDE achieves better results than DELU only on 3 functions (F_4, F_9, and F_{22}). SHADE outperforms DELU on 7 functions including 3 unimodal functions (F_2–F_4) and 4 multi-modal functions (F_6, F_{12}, F_{14}, and F_{19}). DELU performs better than SHADE on 14 out of 28 functions.

Table 14 presents the results obtained by SaDE, EPSDE, JADE, CoDE, SHADE, and DELU for the CEC 2013 benchmark sets at $D=50$. As can be seen, the proposed DELU obtains better results than the other five algorithms on the majority of benchmark functions. In more detail, DELU outperforms SaDE on 20 out of 28 functions, while SaDE performs better than DELU only on 2 functions (F_6 and F_{22}). Compared to EPSDE, DELU shows better performance on 23 out of 28 functions, while EPSDE is better than

Table 13
Mean function error values and standard deviation of SaDE, EPSDE, JADE, CoDE, SHADE, and DELU within the MaxFEs for CEC 2013 benchmark sets at $D=30$.

Fun	D	SaDE Mean Error(Std Dev)	EPSDE Mean Error(Std Dev)	JADE Mean Error(Std Dev)	CoDE Mean Error(Std Dev)	SHADE Mean Error(Std Dev)	DELU Mean Error(Std Dev)
F_1	30	**0.00E+00 (0.00E+00)**$^{\approx}$	2.27E−14 (6.94E−14)$^{+}$	**0.00E+00 (0.00E+00)**$^{\approx}$	**0.00E+00 (0.00E+00)**$^{\approx}$	**0.00E+00 (0.00E+00)**$^{\approx}$	**0.00E+00 (0.00E+00)**
F_2	30	3.87E+04 (2.36E+04)$^{-}$	1.58E+07 (8.92E+06)$^{+}$	8.07E+03 (6.08E+03)$^{-}$	8.36E+04 (6.07E+04)$^{+}$	**6.71E+03 (6.57E+03)**$^{-}$	7.34E+04 (1.80E+04)
F_3	30	2.23E+06 (5.10E+06)$^{+}$	2.35E+08 (2.84E+08)$^{+}$	1.85E+05 (7.25E+05)$^{+}$	5.67E+05 (1.39E+06)$^{+}$	**2.71E+02 (4.79E+02)**$^{-}$	2.91E+03 (3.33E+03)
F_4	30	2.86E+02 (2.02E+02)$^{+}$	4.21E+04 (1.36E+04)$^{+}$	8.45E+03 (1.60E+04)$^{+}$	3.61E−02 (3.91E−02)$^{-}$	**1.28E−06 (1.40E−06)**$^{-}$	6.30E+00 (5.86E+00)
F_5	30	**0.00E+00 (0.00E+00)**$^{\approx}$	1.25E−13 (4.58E−14)$^{+}$	9.09E−14 (4.63E−14)$^{+}$	3.79E−15 (2.08E−14)$^{+}$	1.14E−13 (0.00E+00)$^{+}$	**0.00E+00 (0.00E+00)**
F_6	30	6.94E+00 (6.08E+00)$^{+}$	9.41E+00 (1.45E+01)$^{+}$	8.80E−01 (4.82E+00)$^{+}$	2.98E+00 (7.95E+00)$^{+}$	**1.02E−13 (3.60E−14)**$^{-}$	3.63E−02 (6.72E−02)
F_7	30	2.07E+01 (1.14E+01)$^{+}$	6.44E+01 (1.82E+01)$^{+}$	5.39E+00 (5.36E+00)$^{+}$	8.81E+00 (6.05E+00)$^{+}$	2.79E+00 (3.04E+00)$^{+}$	**5.40E−01 (6.37E−01)**
F_8	30	2.09E+01 (4.46E−02)$^{\approx}$	2.09E+01 (5.68E−02)$^{\approx}$	2.09E+01 (1.06E−01)$^{\approx}$	**2.08E+01 (1.10E−01)**$^{\approx}$	2.08E+01 (2.38E−01)$^{\approx}$	**2.08E+01 (9.92E−02)**
F_9	30	1.53E+01 (2.88E+00)$^{\approx}$	2.91E+01 (2.00E+00)$^{+}$	2.62E+01 (1.82E+00)$^{+}$	**1.43E+01 (3.73E+00)**$^{-}$	2.82E+01 (1.71E+00)$^{+}$	1.48E+01 (2.21E+00)
F_{10}	30	1.30E−01 (8.42E−02)$^{+}$	9.99E−02 (5.21E−02)$^{+}$	4.02E−02 (2.70E−02)$^{+}$	3.32E−02 (1.67E−02)$^{+}$	5.77E−02 (3.15E−02)$^{+}$	**2.17E−02 (9.76E−03)**
F_{11}	30	2.98E−01 (7.90E−01)$^{+}$	3.79E−15 (1.44E−14)$^{+}$	**0.00E+00 (0.00E+00)**$^{\approx}$	3.32E−02 (1.82E−01)$^{+}$	**0.00E+00 (0.00E+00)**$^{\approx}$	**0.00E+00 (0.00E+00)**
F_{12}	30	4.69E+01 (1.18E+01)$^{+}$	6.02E+01 (1.64E+01)$^{+}$	2.50E+01 (4.96E+00)$^{\approx}$	3.52E+01 (1.02E+01)$^{+}$	**2.00E+01 (2.82E+00)**$^{-}$	2.52E+01 (3.05E+00)
F_{13}	30	8.30E+01 (2.52E+01)$^{+}$	8.25E+01 (2.13E+01)$^{+}$	4.96E+01 (1.16E+01)$^{+}$	7.59E+01 (2.12E+01)$^{+}$	4.89E+01 (1.48E+01)$^{+}$	**4.70E+01 (9.98E+00)**
F_{14}	30	7.86E−01 (1.10E+00)$^{+}$	1.42E−01 (2.01E−01)$^{-}$	2.22E−02 (2.11E−02)$^{-}$	2.24E+00 (2.24E+00)$^{+}$	**1.67E−02 (2.74E−02)**$^{-}$	5.02E−01 (5.35E−01)
F_{15}	30	4.65E+03 (1.02E+03)$^{+}$	6.02E+03 (8.94E+02)$^{+}$	3.30E+03 (3.27E+02)$^{+}$	3.33E+03 (7.08E+02)$^{+}$	3.19E+03 (5.15E+02)$^{+}$	**3.01E+03 (4.93E+02)**
F_{16}	30	2.22E+00 (2.65E−01)$^{+}$	2.47E+00 (2.17E−01)$^{+}$	1.97E+00 (7.03E−01)$^{+}$	4.18E−01 (1.74E−01)$^{+}$	8.35E−01 (9.68E−02)$^{+}$	**2.80E−01 (1.52E−01)**
F_{17}	30	3.05E+01 (1.28E−01)$^{\approx}$	3.04E+01 (2.07E−04)$^{\approx}$	3.04E+01 (1.06E−14)$^{\approx}$	3.04E+01 (4.67E−03)$^{\approx}$	**3.04E+01 (0.00E+00)**$^{\approx}$	3.04E+01 (1.77E−03)
F_{18}	30	1.34E+02 (2.22E+01)$^{+}$	1.33E+02 (1.23E+01)$^{+}$	7.64E+01 (6.44E+00)$^{+}$	6.55E+01 (1.11E+01)$^{+}$	7.28E+01 (4.69E+00)$^{+}$	**6.26E+01 (1.40E+01)**
F_{19}	30	3.53E+00 (6.52E−01)$^{+}$	1.83E+00 (1.97E−01)$^{+}$	1.45E+00 (1.12E−01)$^{\approx}$	1.52E+00 (2.63E−01)$^{+}$	**1.37E+00 (1.49E−01)**$^{-}$	1.44E+00 (1.71E−01)
F_{20}	30	1.07E+01 (4.61E−01)$^{\approx}$	1.25E+01 (9.90E−01)$^{+}$	1.03E+01 (5.20E−01)$^{\approx}$	1.07E+01 (5.69E−01)$^{+}$	1.12E+01 (4.86E−01)$^{+}$	**1.02E+01 (1.03E−01)**
F_{21}	30	3.09E+02 (6.16E+01)$^{+}$	3.38E+02 (8.16E+01)$^{+}$	3.19E+02 (7.03E+01)$^{+}$	3.07E+02 (8.77E+01)$^{+}$	2.94E+02 (6.69E+01)$^{+}$	**2.90E+02 (3.16E+01)**
F_{22}	30	1.38E+02 (1.31E+02)$^{+}$	1.11E+02 (4.49E+01)$^{\approx}$	**9.51E+01 (3.01E+01)**$^{-}$	1.13E+02 (2.04E+01)$^{-}$	1.06E+02 (3.49E−01)$^{+}$	1.16E+02 (3.57E+00)
F_{23}	30	4.40E+03 (1.18E+03)$^{+}$	6.43E+03 (8.50E+02)$^{+}$	**3.29E+03 (4.56E+02)**$^{-}$	3.61E+03 (6.43E+02)$^{\approx}$	3.31E+03 (4.21E+02)$^{+}$	3.60E+03 (2.52E+02)
F_{24}	30	2.21E+02 (6.72E+00)$^{+}$	2.71E+02 (7.22E+00)$^{+}$	2.13E+02 (1.49E+01)$^{+}$	2.19E+02 (8.05E+00)$^{+}$	2.08E+02 (9.44E+00)$^{\approx}$	**2.08E+02 (2.59E+00)**
F_{25}	30	2.64E+02 (8.56E+00)$^{+}$	2.95E+02 (4.85E+00)$^{+}$	2.74E+02 (9.98E+00)$^{+}$	2.54E+02 (6.34E+00)$^{+}$	2.71E+02 (1.18E+01)$^{+}$	**2.28E+02 (2.85E+01)**
F_{26}	30	2.00E+02 (1.52E−03)$^{\approx}$	2.01E+02 (4.80E−01)$^{\approx}$	2.28E+02 (5.78E+01)$^{+}$	2.04E+02 (2.04E+01)$^{+}$	**2.00E+02 (1.45E−04)**$^{\approx}$	2.00E+02 (6.11E−02)
F_{27}	30	5.42E+02 (6.84E+01)$^{+}$	1.03E+03 (5.05E+01)$^{+}$	6.97E+02 (2.54E+02)$^{+}$	6.04E+02 (9.00E+01)$^{+}$	4.12E+02 (1.23E+02)$^{+}$	**3.65E+02 (3.66E+01)**
F_{28}	30	**3.00E+02 (0.00E+00)**$^{\approx}$	**3.00E+02 (0.00E+00)**$^{\approx}$	**3.00E+02 (0.00E+00)**$^{\approx}$	**3.00E+02 (0.00E+00)**$^{\approx}$	**3.00E+02 (0.00E+00)**$^{\approx}$	**3.00E+02 (0.00E+00)**
$w/t/l$		19/8/1	22/5/1	16/8/4	20/5/3	14/7/7	–/–/–

DELU only on one function (F_{26}). Although JADE is superior to DELU on 8 functions including 2 unimodal functions (F_2 and F_3), 4 multimodal functions (F_7, F_{12}, F_{14}, and F_{19}), and 2 composition functions (F_{22} and F_{23}), DELU shows better performance than JADE on 15 out of 28 functions. Compared with CoDE, DELU achieves better performance on 21 out of 28 functions, while CoDE is better

Table 14
Mean function error values and standard deviation of SaDE, EPSDE, JADE, CoDE, SHADE, and DELU within the MaxFEs for CEC 2013 benchmark sets at $D=50$.

Fun	D	SaDE Mean Error(Std Dev)	EPSDE Mean Error(Std Dev)	JADE Mean Error(Std Dev)	CoDE Mean Error(Std Dev)	SHADE Mean Error(Std Dev)	DELU Mean Error(Std Dev)
F_1	50	**0.00E+00 (0.00E+00)**$^{\approx}$	2.27E−13 (0.00E+00)$^{+}$	1.82E−13 (9.59E−14)$^{+}$	**0.00E+00 (0.00E+00)**$^{\approx}$	1.82E−13 (9.59E−14)$^{+}$	**0.00E+00 (0.00E+00)**
F_2	50	1.85E+05 (6.15E+04)$^{+}$	8.06E+07 (1.79E+07)$^{+}$	2.74E+04 (1.80E+04)$^{-}$	2.43E+05 (7.96E+04)$^{+}$	**2.19E+04 (9.81E+03)**$^{-}$	1.52E+05 (5.91E+04)
F_3	50	3.20E+07 (3.77E+07)$^{+}$	5.50E+09 (2.08E+09)$^{+}$	3.45E+06 (5.77E+06)$^{-}$	1.50E+07 (1.06E+07)$^{+}$	**2.57E+06 (5.75E+06)**$^{-}$	4.36E+06 (3.47E+06)
F_4	50	9.69E+02 (6.60E+02)$^{+}$	8.36E+04 (2.16E+04)$^{+}$	4.18E+03 (1.32E+04)$^{+}$	1.68E−01 (1.46E−01)$^{+}$	**4.74E−05 (5.18E−05)**$^{-}$	1.31E−03 (1.16E−03)
F_5	50	**0.00E+00 (0.00E+00)**$^{\approx}$	4.32E−13 (1.17E−13)$^{+}$	1.36E−13 (4.79E−14)$^{+}$	6.82E−14 (5.87E−14)$^{+}$	1.36E−13 (4.79E−14)$^{+}$	**0.00E+00 (0.00E+00)**
F_6	50	**4.17E+01 (7.15E+00)**$^{-}$	4.34E+01 (7.93E−13)$^{\approx}$	4.34E+01 (3.61E−14)$^{\approx}$	4.34E+01 (3.61E−14)$^{\approx}$	4.34E+01 (9.59E−14)$^{\approx}$	4.34E+01 (0.00E+00)
F_7	50	4.13E+01 (9.80E+00)$^{\approx}$	1.08E+02 (3.02E+01)$^{+}$	**1.63E+01 (5.97E+00)**$^{-}$	3.27E+01 (6.44E+00)$^{-}$	1.76E+01 (1.18E+01)$^{-}$	4.16E+01 (1.22E+01)
F_8	50	2.11E+01 (2.56E−02)$^{\approx}$	2.11E+01 (4.33E−02)$^{\approx}$	2.11E+01 (2.53E−02)$^{\approx}$	2.10E+01 (9.12E−02)$^{\approx}$	2.09E+01 (2.27E−01)$^{\approx}$	**2.09E+01 (5.49E−02)**
F_9	50	3.73E+01 (4.58E+00)$^{+}$	6.22E+01 (7.31E+00)$^{+}$	5.41E+01 (2.29E+00)$^{+}$	3.25E+01 (6.71E+00)$^{+}$	5.60E+01 (2.35E+00)$^{+}$	**3.05E+01 (2.69E+00)**
F_{10}	50	1.31E−01 (5.95E−02)$^{+}$	1.20E−01 (6.79E−02)$^{+}$	4.42E−02 (3.18E−02)$^{+}$	6.75E−02 (3.23E−02)$^{+}$	4.70E−02 (1.88E−02)$^{+}$	**3.97E−02 (2.04E−02)**
F_{11}	50	2.69E+00 (1.56E+00)$^{+}$	6.25E−14 (1.80E−14)$^{+}$	**0.00E+00 (0.00E+00)**$^{\approx}$	6.96E−01 (6.72E−01)$^{+}$	5.68E−15 (1.80E−14)$^{+}$	**0.00E+00 (0.00E+00)**
F_{12}	50	1.11E+02 (2.08E+01)$^{+}$	1.87E+02 (4.95E+01)$^{+}$	**5.05E+01 (4.01E+00)**$^{-}$	9.65E+01 (2.17E+01)$^{+}$	5.08E+01 (5.85E+00)$^{-}$	5.65E+01 (6.57E+00)
F_{13}	50	2.49E+02 (4.42E+01)$^{+}$	2.06E+02 (2.41E+01)$^{+}$	1.24E+02 (2.89E+01)$^{+}$	2.09E+02 (5.55E+01)$^{+}$	1.15E+02 (3.23E+01)$^{+}$	**1.12E+02 (2.23E+01)**
F_{14}	50	1.14E+01 (9.67E+00)$^{+}$	5.79E+01 (1.10E+02)$^{+}$	**1.87E−02 (1.21E−02)**$^{-}$	2.07E+01 (1.20E+01)$^{+}$	2.75E−02 (1.29E−02)$^{-}$	6.92E−02 (2.14E−02)
F_{15}	50	1.05E+04 (9.32E+02)$^{+}$	1.35E+04 (8.29E+02)$^{+}$	7.13E+03 (2.02E+02)$^{+}$	7.33E+03 (8.92E+02)$^{+}$	7.12E+03 (5.51E+02)$^{+}$	**6.92E+03 (3.01E+02)**
F_{16}	50	3.05E+00 (4.34E−01)$^{+}$	3.26E+00 (3.49E−01)$^{+}$	1.64E+00 (5.87E−01)$^{+}$	1.16E+00 (2.27E−01)$^{+}$	1.22E+00 (1.19E−01)$^{+}$	**8.29E−01 (3.30E−01)**
F_{17}	50	5.28E+01 (1.45E+00)$^{+}$	5.08E+01 (9.66E−04)$^{\approx}$	5.08E+01 (4.02E−14)$^{\approx}$	5.23E+01 (6.29E−01)$^{+}$	**5.08E+01 (3.61E−14)**$^{\approx}$	5.08E+01 (2.70E−05)
F_{18}	50	2.23E+02 (7.17E+01)$^{+}$	3.27E+02 (1.97E+01)$^{+}$	1.52E+02 (7.27E+00)$^{+}$	1.36E+02 (2.80E+01)$^{+}$	1.39E+02 (1.78E+01)$^{+}$	**1.26E+02 (8.42E+00)**
F_{19}	50	9.75E+00 (7.55E−01)$^{+}$	5.80E+00 (4.18E−01)$^{+}$	2.83E+00 (1.81E−01)$^{-}$	3.35E+00 (5.29E−01)$^{+}$	**2.59E+00 (2.46E−01)**$^{-}$	3.19E+00 (5.35E−01)
F_{20}	50	2.02E+01 (7.37E−01)$^{+}$	2.33E+01 (1.05E+00)$^{+}$	1.99E+01 (4.54E−01)$^{+}$	2.04E+01 (4.19E−01)$^{+}$	1.97E+01 (3.66E−01)$^{\approx}$	**1.96E+01 (5.20E−01)**
F_{21}	50	9.79E+02 (1.51E+02)$^{+}$	1.00E+03 (2.96E+02)$^{+}$	7.88E+02 (4.22E+02)$^{+}$	6.96E+02 (4.41E+02)$^{+}$	1.00E+03 (2.96E+02)$^{+}$	**5.12E+02 (4.39E+02)**
F_{22}	50	2.27E+01 (6.45E+00)$^{-}$	9.53E+02 (4.43E+02)$^{+}$	3.54E+01 (6.68E+01)$^{-}$	3.44E+01 (1.05E+01)$^{-}$	**1.12E+01 (8.86E−01)**$^{-}$	7.18E+01 (9.98E+01)
F_{23}	50	9.46E+03 (2.31E+03)$^{+}$	1.36E+04 (5.42E+02)$^{+}$	7.08E+03 (6.50E+02)$^{-}$	7.60E+03 (6.99E+02)$^{+}$	7.58E+03 (5.61E+02)$^{+}$	7.16E+03 (1.08E+03)
F_{24}	50	2.77E+02 (5.87E+00)$^{+}$	3.45E+02 (7.76E+00)$^{+}$	2.66E+02 (9.08E+00)$^{+}$	2.65E+02 (1.70E+01)$^{+}$	2.40E+02 (9.28E+00)$^{+}$	**2.32E+02 (7.70E+00)**
F_{25}	50	3.39E+02 (8.93E+00)$^{+}$	3.84E+02 (9.17E+00)$^{+}$	3.55E+02 (2.23E+01)$^{+}$	**3.06E+02 (1.60E+01)**$^{-}$	3.40E+02 (2.35E+01)$^{+}$	3.13E+02 (1.11E+01)
F_{26}	50	2.75E+02 (9.73E+01)$^{+}$	**2.14E+02 (3.96E+00)**$^{-}$	3.56E+02 (1.07E+02)$^{+}$	3.19E+02 (8.44E+01)$^{+}$	2.49E+02 (8.01E+01)$^{+}$	2.32E+02 (7.17E+01)
F_{27}	50	1.09E+03 (4.33E+01)$^{\approx}$	1.83E+03 (4.74E+01)$^{+}$	1.31E+03 (3.38E+02)$^{+}$	1.21E+03 (1.54E+02)$^{+}$	1.08E+03 (4.09E+02)$^{+}$	**1.05E+03 (1.20E+02)**
F_{28}	50	4.00E+02 (7.58E−14)$^{\approx}$	4.00E+02 (3.18E−13)$^{\approx}$	**4.00E+02 (0.00E+00)**$^{\approx}$	4.00E+02 (5.99E−14)$^{\approx}$	9.98E+02 (1.26E+03)$^{+}$	**4.00E+02 (0.00E+00)**
$w/t/l$		20/6/2	23/4/1	15/5/8	21/4/3	16/4/8	–/–/–

than DELU only on 3 functions (F_7, F_{22}, and F_{25}). SHADE outperforms DELU on 8 functions including 3 unimodal functions ($F_2 - F_4$), 4 multimodal functions (F_7, F_{12}, F_{14}, and F_{19}), and one composition function (F_{22}), while DELU is superior to SHADE 16 out of 28 functions.

5.7. Parameter study

In the proposed DELU, to obtain the underestimate value of the trial individual, the N neighboring individuals of the trial individual that are measured by the Euclidean distance are used to construct the supporting hyperplanes. The number N of the neighboring individuals of the trial individual needs to be optimized. In this section, benchmark functions $f_1 - f_{18}$ with 10–D are used to investigate the impact of the parameter N on DELU. DELU runs with four different N of 2, 3, 4, 5, and 6. The mean number of function evaluations (FES) required to reach the function error below the predefined accuracy level (δ) within the MaxFES, and the success rate (SR) are recorded. In this comparison, δ is set to 1E − 5, and the MaxFES is set to $1000 \times D$.

Table 15 summarizes the mean number of FES and SR required to reach the predefined accuracy level with five different N. From the total average results listed in the last row of the table, we can find that DELU succeeds on all functions except for f_{12} with similar FES by using five different N. Therefore, the convergence speed of DELU is less sensitive to the parameter N. However, the space complexity of the algorithm increases along with the parameter N increasing, as more neighboring individuals will be used to construct the supporting hyperplanes if N increases. Thus, we only construct the supporting hyperplanes for the 2 ($N=2$) neighboring individuals of the trial individual in DELU which is more preferable.

5.8. Discussions

The DE algorithm is a fast, robust, and simple global optimization algorithm that has been applied in various fields. However, in DE, a large number of function evaluations are needed to find optimal solutions while expensive costs may have to be paid for function evaluations, especially for expensive-to-evaluate problems. In addition, DE is good at exploring the search space and locating the region of global minimum, but it is slow at exploiting the solution. The method of abstract convex underestimate can estimate the function value and exclude the regions where the global optimum cannot reside by constructing the abstract convex relaxed model of the objective function. Therefore, in this work, the local abstract convex underestimate strategy is introduced into DE to guide the selection operation of DE, reduce the number of function evaluations, and find the balance the exploration and exploitation. Thus, the DELU algorithm which is based on the local abstract convex underestimate strategy is proposed. From the experimental results and analysis, we can summarize that

(1) The proposed DELU is effective and efficient. It can obtain the global, or near-global optimum for unimodal and multimodal problems.
(2) The overall performance of DELU is superior to or highly competitive with five state-of-the-art DE variants, three non-DE algorithms, and three surrogate-assisted evolutionary algorithms.
(3) Due to the local abstract convex underestimate strategy, DELU is able to reduce the number of function evaluations and balance the exploration and exploitation.
(4) The proposed local abstract convex underestimate strategy is also capable of improving the performance of some recently presented advanced DE variants.
(5) According to the nonparametric Wilcoxon Signed Rank Test, the proposed DELU is significantly better than, or highly competitive with other competitors for the majority of benchmark functions.

Two main drawbacks of the abstract convex underestimate strategy are that it may take time to construct the supporting hyperplanes and that it needs computer memory to store the underestimate value. However, to reduce the complexity produced by the abstract convex underestimate strategy, the supporting hyperplanes are only constructed for the 2 neighboring individuals of the trial individual in the proposed local abstract convex underestimate strategy, and the neighboring individuals are selected according to the Euclidean distance from the trial individual to other individuals. Specifically, all supporting hyperplanes are deleted after the selection operation at each iteration. Moreover, the computational cost for the evaluation of the functions of expensive-to-evaluate problems is far greater than the cost for the

Table 15
Results of DELU uses different number of neighboring individuals (N) of the trial for supporting hyperplanes construction on test functions $f_1 - f_{18}$ at $D=10$.

Fun	D	$N=2$		$N=3$		$N=4$		$N=5$		$N=6$	
		FES	SR (%)	FES	SR (%)	FES	SR (%)	FES	SR (%)	FES	SR (%)
f_1	10	1.02E+03	100	1.13E+03	100	1.06E+03	100	1.04E+03	100	1.14E+03	100
f_2	10	5.13E+03	100	5.04E+03	100	5.73E+03	100	5.16E+03	100	5.52E+03	100
f_3	10	1.31E+04	100	1.30E+04	100	1.31E+04	100	1.30E+04	100	1.40E+04	100
f_4	10	2.16E+03	100	2.80E+03	100	2.64E+03	100	2.53E+03	100	2.37E+03	100
f_5	10	1.14E+04	100	1.13E+04	100	1.13E+04	100	1.12E+04	100	1.27E+04	100
f_6	10	3.34E+03	100	3.29E+03	100	3.60E+03	100	3.28E+03	100	3.49E+03	100
f_7	10	1.22E+04	100	1.20E+04	100	1.18E+04	100	1.15E+04	100	1.09E+04	100
f_8	10	2.88E+04	100	3.13E+04	100	3.05E+04	100	3.10E+04	100	2.95E+04	100
f_9	10	8.32E+03	100	8.45E+03	100	8.22E+03	100	7.92E+03	100	8.32E+03	100
f_{10}	10	2.69E+04	100	2.62E+04	100	2.59E+04	100	2.57E+04	100	2.49E+04	100
f_{11}	10	9.03E+03	100	9.53E+03	100	1.01E+04	100	1.04E+04	100	9.82E+03	100
f_{12}	10	6.26E+03	87	6.14E+03	90	6.19E+03	87	6.22E+03	90	6.36E+03	90
f_{13}	10	4.67E+03	100	4.98E+03	100	5.05E+03	100	5.21E+03	100	4.81E+03	100
f_{14}	10	3.94E+03	100	4.18E+03	100	4.27E+03	100	4.31E+03	100	3.87E+03	100
f_{15}	10	1.26E+04	100	1.24E+04	100	1.24E+04	100	1.23E+04	100	1.30E+04	100
f_{16}	10	8.96E+03	100	9.40E+03	100	9.71E+03	100	9.64E+03	100	9.08E+03	100
f_{17}	10	7.18E+03	100	7.17E+03	100	7.11E+03	100	7.23E+03	100	6.99E+03	100
f_{18}	10	7.73E+03	100	7.74E+03	100	7.81E+03	100	7.88E+03	100	8.04E+03	100
Total average		9.60E+03	99.3	9.78E+03	99.4	9.80E+03	99.3	9.75E+03	99.4	9.72E+03	99.4

construction of supporting hyperplanes. Therefore, the proposed algorithms are effective and particularly efficient for expensive-to-evaluate problems.

6. Conclusions

This paper presented a DELU algorithm, which reduces the number of function evaluations and balances the exploration and exploitation of DE. In DELU, a local abstract convex underestimate strategy based on the underestimate technique used in CAM is introduced into the selection operation of DE. With the underestimate value as guide, the algorithm does not need to evaluate all trial individuals. Moreover, some invalid regions where the global optimum cannot reside can also be safely excluded according to the underestimate, thereby eliminating the unnecessary searching in invalid regions, avoiding the premature convergence, and achieving the desired convergence speed. In addition, convergence speed is further accelerated by employing the direction of supporting hyperplanes.

The performance of DELU was verified by comparing it with five state-of-the-art DE variants and three advanced non-DE variants over a suite of 23 bound-constrained benchmark functions. Numerical experimental results suggest that DELU requires less function evaluations in obtaining better quality solutions, which are more stable with the relatively smaller standard deviation of function error, and has faster convergence speed. Specially, through incorporating the proposed local abstract convex underestimate strategy into some advanced DE variants, we find that the performances of the advanced DE variants are also improved. Experimental results also show that DELU is better than three state-of-the-art surrogate-assisted evolutionary algorithms. In addition, compared with the five state-of-the-art DE variants on the CEC 2013 shifted and rotated benchmark sets, DELU achieves better results on the majority of benchmark functions.

The proposed local abstract convex underestimate strategy provides inspirations and insights into other population algorithms. It can perform as an effective method to solve complex optimization problems, such as the protein-folding problem that has caught the attention of researchers all over the world. For example, the underestimate strategy can be used to reduce the number of evaluations of force fields to improve prediction efficiency, thus guaranteeing the prediction of protein conformation at a more accurate level. In the future, we intend to extend our work to the protein-folding problem and other complex optimization problems.

Acknowledgements

This work was supported by National Nature Science Foundation of China (No. 61075062, 61573317), Natural Science Foundation of Zhejiang Province (No. LY13F030008), and Public Welfare Project of Science Technology Department of Zhejiang Province (No. 2014C33088). The authors would like to thank Prof. P.N. Suganthan for providing the source code of SaDE, JADE, CoDE, EPSDE, SHADE, CLPSO, CMA-ES, and GL-25, and gratefully acknowledge the anonymous reviewers for their insight comments and useful suggestions.

References

[1] Floudas CA, Gounaris CE. A review of recent advances in global optimization. J Glob Optim 2009;45(1):3–38.
[2] Storn R, Price K. Differential evolution—a simple and efficient heuristic for global optimization over continuous spaces. J Glob Optim 1997;11(4):341–59.
[3] Michalewicz Z. Genetic algorithms + data structures = evolution programs. London: Springer; 1996.
[4] Langdon WB, Poli R. Evolving problems to learn about particle swarm optimizers and other search algorithms. IEEE Trans Evol Comput 2007;11(5):561–78.
[5] Rechenberg I. Evolutions strategie: optimierung technischer systeme nach prinzipien der biologischen evolution. Stuttgart, Germany: Fromman-Holzboog; 1973.
[6] Fogel L, Owens J, Walsh M. Artificial intelligence through simulated evolution. New York: Wiley; 1966.
[7] Das S, Suganthan PN. Differential evolution: a survey of the state-of-the-art. IEEE Trans Evol Comput 2011;15(1):4–31.
[8] Tasgetiren MF, Pan QK, Suganthan PN, Buyukdagli O. A variable iterated greedy algorithm with differential evolution for the no-idle permutation flowshop scheduling problem. Comput Oper Res 2013;40(7):1729–43.
[9] Bhattacharya A, Chattopadhyay PK. Hybrid differential evolution with biogeography-based optimization for solution of economic load dispatch. IEEE Trans Power Syst 2010;25(4):1955–64.
[10] Sotiroudis SP, Goudos SK, Gotsis KA, Siakavara K, Sahalos JN. Application of a composite differential evolution algorithm in optimal neural network design for propagation path-loss prediction in mobile communication systems. IEEE Antennas Wirel Propag Lett 2013;12:364–7.
[11] Sharma S, Rangaiah GP. An improved multi-objective differential evolution with a termination criterion for optimizing chemical processes. Comput Chem Eng 2013;56(9):155–73.
[12] Venu MK, Mallipeddi R, Suganthan PN. Fiber Bragg grating sensor array interrogation using differential evolution. Optoelectron Adv Mater – Rapid Commun 2008;2(11):682–5.
[13] Sudha S, Baskar S, Amali SMJ, Krishnaswamy S. Protein structure prediction using diversity controlled self-adaptive differential evolution with local search. Soft Comput 2014;19(6):1635–46.
[14] Adra SF, Dodd TJ, Griffin IA, Fleming PJ. Convergence acceleration operator for multiobjective optimization. IEEE Trans Evol Comput 2009;13(4):825–47.
[15] Noman N, Iba H. Accelerating differential evolution using an adaptive local search. IEEE Trans Evol Comput 2008;12:107–25.
[16] Liu B, Zhang Q, Gielen G. A Gaussian process surrogate model assisted evolutionary algorithm for medium scale expensive optimization problems. IEEE Trans Evol Comput 2014;18(2):180–92.
[17] Zhou Z, Ong YS, Lim MH, Lee BS. Memetic algorithm using multi-surrogates for computationally expensive optimization problems. Soft Comput 2007;11(10):957–71.
[18] Müller J, Shoemaker CA, Piché R. SO-MI: a surrogate model algorithm for computationally expensive nonlinear mixed-integer black-box global optimization problems. Comput Oper Res 2013;40(5):1383–400.
[19] Glaz B, Friedmann PP, Liu L. Surrogate based optimization of helicopter rotor blades for vibration reduction in forward flight. Struct Multidiscip Optim 2008;35(4):341–63.
[20] Hou S, Tan W, Zheng Y, Han X. Optimization design of corrugated beam guardrail based on RBF-MQ surrogate model and collision safety consideration. Adv Eng Softw 2014;78(12):28–40.
[21] Li F, Wang J, Brigham JC. Inverse calculation of insitu stress in rock mass using the surrogate-model accelerated random search algorithm. Comput Geotech 2014;61(9):24–32.
[22] Mtüller J, Piché R. Mixture surrogate models based on Dempster–Shafer theory for global optimization problems. J Glob Optim 2011;51(1):79–104.
[23] Rubinov AM. Abstract convexity and global optimization. Nonconvex optimization and its applications.Dordrecht: Kluwer Academic Publishers; 2000.
[24] Andramonov M, Rubinov A, Glover B. Cutting angle methods in global optimization. Appl Math Lett 1999;12(3):95–100.
[25] Rockafellar RT. Convex analysis.Princeton, New Jersey: Princeton University Press; 1970.
[26] Beliakov G. Geometry and combinatorics of the cutting angle method. Optimization 2003;52(4–5):379–94.
[27] Beliakov G, Kelarev A. Global non-smooth optimization in robust multivariate regression. Optim Methods Softw 2013;28(1):124–38.
[28] Ferrer A, Bagirov A, Beliakov G. Solving DC programs using the cutting angle method. J Glob Optim 2015;61(1):71–89.
[29] Beliakov G, Lim KF. Challenges of continuous global optimization in molecular structure prediction. Eur J Oper Res 2007;181(3):1198–213.
[30] Bagirov AM, Rubinov AM. Global minimization of increasing positively homogeneous functions over the unit simplex. Ann Oper Res 2000;98(1–4):171–87.
[31] Bagirov AM, Rubinov AM. Fast algorithm for the cutting angle method of global optimization. J Global Optim 2002;24(2):149–61.
[32] Beliakov G. Extended cutting angle method of global optimization. Pac J Optim 2008;4(1):152–76.
[33] Queipo NV, Haftka RT, Shyy W, Goel T, Vaidyanathan R, Tucker PK. Surrogate-based analysis and optimization. Prog Aerosp Sci 2005;41(1):1–28.
[34] Liu Y, Sun F. A fast differential evolution algorithm using k-nearest neighbour predictor. Expert Syst Appl 2011;38(4):4254–8.
[35] Jin S. An efficient evolutionary optimization with fitness approximation using neural networks (Masters thesis). Korea Advanced Institute of Science and Technology; 2007.
[36] Zhang Q, Liu W, Tsang E, Virginas B. Expensive multiobjective optimization by MOEA/D with Gaussian process model. IEEE Trans Evol Comput 2010;14(3):456–74.

[37] Park SY, Lee JJ. An efficient differential evolution using speeded-up k-nearest neighbor estimator. Soft Comput 2014;18(1):35–49.
[38] Wang L, Pan QK, Suganthan PN, Wang WH, Wang YM. A novel hybrid discrete differential evolution algorithm for blocking flow shop scheduling problems. Comput Oper Res 2010;37(3):509–20.
[39] Cai ZH, Gong WY, Ling CX, Zhang H. A clustering-based differential evolution for global optimization. Appl Soft Comput 2011;11(1):1363–79.
[40] Piotrowski AP, Napiorkowski JJ, Kiczko A. Differential evolution algorithm with separated groups for multi-dimensional optimization problems. Eur J Oper Res 2012;216(1):33–46.
[41] Li X, Yin M. Modified differential evolution with self-adaptive parameters method. J Comb Optim 2016;31(2):546–76.
[42] Qin AK, Huang VL, Suganthan PN. Differential evolution algorithm with strategy adaptation for global numerical optimization. IEEE Trans Evol Comput 2009;13(2):398–417.
[43] Zhang JQ, Sanderson AC. JADE: adaptive differential evolution with optional external archive. IEEE Trans Evol Comput 2009;13(5):945–58.
[44] Wang Y, Cai Z, Zhang Q. Differential evolution with composite trial vector generation strategies and control parameters. IEEE Trans Evol Comput 2011;15(1):55–66.
[45] Mallipeddi R, Suganthan PN, Pan QK, Tasgetiren MF. Differential evolution algorithm with ensemble of parameters and mutation strategies. Appl Soft Comput 2011;11(2):1679–96.
[46] Pan QK, Suganthan PN, Wang L, Gao L, Mallipeddi R. A differential evolution algorithm with self-adapting strategy and control parameters. Comput Oper Res 2011;38(1):394–408.
[47] Bagirov AM, Rubinov AM. Cutting angle method and a local search. J Glob Optim 2003;27(2–3):193–213.
[48] Tanabe R, Fukunaga A. Success-history based parameter adaptation for differential evolution. In: Proceedings of IEEE congress on evolutionary computation (CEC), Cancun, Mexico; 2013. p. 71–8.
[49] Corder GW, Foreman DI. Nonparametric statistics for non-statisticians: a step-by-step approach.Hoboken, New Jersey: John Wiley & Sons; 2009.
[50] Shang YW, Qiu YH. A note on the extended Rosenbrock function. Evol Comput 2006;14(1):119–26.
[51] Liang JJ, Qin AK, Suganthan PN, Baskar S. Comprehensive learning particle swarm optimizer for global optimization of multimodal functions. IEEE Trans Evol Comput 2006;10(3):281–95.
[52] Hansen N, Ostermeier A. Completely derandomized self-adaptation in evolution strategies. Evol Comput 2001;9(2):159–95.
[53] Garcia-Martinez C, Lozano M, Herrera F, Molina D, Sanchez AM. Global and local real-coded genetic algorithms based on parent-centric crossover operators. Eur J Oper Res 2008;185(3):1088–113.
[54] GarcÍa S, Fernández A, Luengo J, Herrera F. Advanced nonparametric tests for multiple comparisons in the design of experiments in computational intelligence and data mining: experimental analysis of power. Inf Sci 2010;180(10):2044–64.
[55] Lim D, Jin Y, Ong YS, Sendhoff B. Generalizing surrogate-assisted evolutionary computation. IEEE Trans Evol Comput 2010;14(3):329–55.
[56] Zhou Z, Ong YS, Nair PB, Keane AJ, Lum KY. Combining global and local surrogate models to accelerate evolutionary optimization. IEEE Trans Syst Man Cybern Part C: Appl Rev 2007;37(1):66–76.
[57] Suganthan PN, Hansen N, Liang JJ, Deb K, Chen YP, Auger A, et al. Problem definitions and evaluation criteria for the CEC 2005 special session on real-parameter optimization. Singapore: Nanyang Technological University; 2005. Technical report 2005005.
[58] Liang JJ, Qu BY, Suganthan PN, Hernández-DÍaz AG. Problem definitions and evaluation criteria for the CEC 2013 special session on real-parameter optimization. Zhengzhou, China/Singapore: Zhengzhou University/Nanyang Technological University; 2013. Technical report 201212.