Applied Soft Computing 48 (2016) 169–181

Contents lists available at ScienceDirect

Applied Soft Computing

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

Enhanced differential evolution using local Lipschitz underestimate strategy for computationally expensive optimization problems

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

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

ARTICLE INFO

Article history:
Received 14 July 2015
Received in revised form 15 June 2016
Accepted 30 June 2016
Available online 16 July 2016

Keywords:
Differential evolution
Evolutionary algorithm
Supporting hyperplane
Lipschitz underestimate
Global optimization

ABSTRACT

Differential evolution (DE) has been successfully applied in many scientific and engineering fields. However, one of the main problems in using DE is that the optimization process usually needs a large number of function evaluations to find an acceptable solution, which leads to an increase of computational time, particularly in the case of computationally expensive problems. The Lipschitz underestimate method (LUM), a deterministic global optimization technique, can be used to obtain the underestimate of the objective function by building a sequence of piecewise linear supporting hyperplanes. In this paper, an enhanced differential evolution using local Lipschitz underestimate strategy, called LLUDE, is proposed for computationally expensive optimization problems. LLUDE effectively combines the exploration of DE with the underestimation of LUM to obtain promising solutions with less function evaluations. To verify the performance of LLUDE, 18 well-known benchmark functions and one computationally expensive real-world application problem, namely, the protein structure prediction problem, are employed. Results obtained from these benchmark functions show that LLUDE is significantly better than or at least comparable to the state-of-the-art DE variants and non-DE algorithms. Furthermore, the results obtained from the protein structure prediction problem suggest that LLUDE is effective and efficient for computationally expensive problems.

1. Introduction

Differential evolution (DE) [1], proposed by Storn and Price, is a simple yet powerful population-based stochastic search algorithm for solving nonlinear, non-differentiable, and continuous space problems [2,3]. DE enriches a population of candidate solutions over several generations using three basic operations: mutation, crossover, and selection operators. Due to its advantage of simple structure, ease of use, speed and robustness, it is widely used in many real-world applications [4–6], such as power systems, communication, chemical engineering, optics, pattern recognition, and bioinformatics. However, like other population-based algorithms such as particle swarm optimization (PSO) [7,8], genetic algorithm (GA) [9,10], and ant colony optimization (ACO) [11,12], one of the main drawbacks of DE is its requirement of large number of function evaluations to find optimal solutions [13]. Unfortunately, due to the complexity of the objective function, the function evaluation is often very time-consuming, especially for computationally expensive optimization problems in real-world applications. For example, the protein structure prediction problem usually involves an energy function with thousands of degrees of freedom. Several minutes or even hours may be required to evaluate the energy function using a high-performance computer. Thus reducing the number of actual function evaluations in obtaining an acceptable solution is crucial for DE.

Numerous time-efficient function approximation surrogate models including Gaussian process models [14], kriging models [15], neural network models [16], polynomial regression models [17], radial basis function models [18], and support vector machines [19] have been developed to reduce the number of function evaluations in population-based algorithms. In these techniques, approximation models are used to estimate the objective function and the global optimum of the optimization problem is obtained using the estimated values. Liu et al. [20] employed a dimensionality reduction technique to reduce the problem to a low-dimensional space, as well as proposed a surrogate model-aware search mechanism to reduce the exact function evaluations. Jin and Sendhoff [21] grouped the population into a number of clusters. Only the individual closest to the cluster center is evaluated, while the fitness of other individuals are estimated using a neural network model. Takahama and Sakai [22] proposed a differential evolution with the estimated comparison method, which utilized

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

http://dx.doi.org/10.1016/j.asoc.2016.06.044
1568-4946/

kernel smoothers to estimate the function value. Mallipeddi and Lee [23] proposed an evolving surrogate model-based DE, in which a surrogate model constructed based on the individuals in the population to help DE to generate competitive offspring. Elsayed et al. [24] proposed a surrogate-assisted DE, in which a kriging model is used to approximate the objective function value.

Apart from the aforementioned surrogate model-based methods, many other function approximation techniques have been proposed to reduce the computational burden in population-based algorithms. Yuan et al. [25] introduced an existing linear-time algorithm to approximate the objective function. Liu and Sun [26] proposed a fast DE algorithm, in which a cheap k-nearest neighbor predictor was constructed through dynamic learning. Park and Lee [13] proposed an efficient differential evolution using a speeded-up k-nearest neighbor estimator, which improved the accuracy and efficiency of the k-nearest neighbor predictor to further reduce the exact computationally expensive evaluation. Pham [27] proposed a DE with nearest neighbor comparison, in which a nearest neighbor is used to judge whether or not a new individual is worth evaluating. Paschalidis et al. [28] proposed a semidefinite programming-based underestimation method, which used a class of general convex quadratic functions to estimate the objective function. However, the effectiveness of these function approximation methods highly depends on the accuracy of the approximation model. A time-consuming learning process is usually needed to obtain a high accuracy model. In addition, selecting a appropriate model is a challenge because no one model suits all kinds of problems [29].

The Lipschitz underestimate method (LUM) [30–33] is a deterministic global optimization technique. The objective function is approximated from below using a pointwise maximum of tangent planes (or supporting hyperplanes) in LUM. The underestimate is a piecewise affine convex function, and its global optimum can be determined through linear programming techniques. At each iteration, a new supporting hyperplane is added to the underestimate, which becomes a more accurate approximation of the objective function. In addition, those part of the domain where the global optimum cannot be found can be safely excluded using the underestimate. LUM is very efficient in addressing problems of relatively small dimension ($\leq$10) [31]. However, a large number of supporting hyperplanes need to be constructed to obtain a highly accurate underestimate for large-scale problems, thereby leading to high complexity. As a result, LUM is inappropriate for solving large-scale problems.

In this paper, an enhanced DE using local Lipschitz underestimate strategy (LLUDE) is proposed to reduce the number of function evaluations without deteriorating the performance. In LLUDE, the supporting hyperplanes are constructed only for some neighboring individuals of the trial individual which measured by Euclidean distances to calculate the underestimate value of the trial individual. Then the underestimate value is used to determine whether the trial individual is worth evaluating to avoid unnecessary function evaluations. Meanwhile, those part of the domain where the global optimum cannot reside are safely excluded according to the local minimum of the underestimate to improve the exploration efficiency. In addition, the generalized descent directions of the supporting hyperplanes can be employed for local enhancement to improve the exploitation capability further. Hence LLUDE can substantially reduce the number of function evaluations and achieve good search capabilities. The numerical results obtained from the 18 unconstrained benchmark functions and one real-world problem indicate that the proposed algorithm significantly better than or at least comparable to the state-of-the-art DE and non-DE variants mentioned in this paper.

The rest of this paper is organized as follows. Section 2 briefly describes the DE algorithm and LUM. Section 3 presents the proposed enhanced DE using local Lipschitz underestimate strategy. Experimental results and analysis are reported in Section 4. Section 5 concludes this paper.

2. Preliminary

2.1. Differential evolution

Differential evolution [1] is a population-based stochastic search algorithm for global optimization. It is able to solve nonlinear, non-differentiable, and multimodal objective problems [2]. Like other evolutionary algorithms, DE also includes initialization, mutation, crossover, and selection operators. A brief description of the traditional DE is given as follows:

First, at generation $g=0$, an initial population $P=\{x_1^0, x_2^0, \ldots, x_{N_p}^0\}$ is randomly sampled from the feasible solution space Ω, where N_P is the population size. At each generation g, DE creates a mutant vector for each target vector x_i^g, $i=1, 2, \ldots, N_P$ in the population P. The mutation operation is described as

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

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

Then a crossover operation comes into play after mutation. The trial vector is generated according to

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

where $j=1, 2, \ldots, N$, j_{rand} is a randomly chosen integer in $[1, N]$, $\text{rand}_j(0, 1)$ is a uniformly distributed random number between 0 and 1, which is generated for each j, and $C_R\in(0, 1)$ 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.

Finally, the selection operation is performed to determine whether the target vector x_i^g or the trial vector u_i^g enter the next generation

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

where $f(u_i^g)$ and $f(x_i^g)$ are the objective function value of u_i^g and x_i^g, respectively.

2.2. Lipschitz underestimate method

The Lipschitz underestimate method [30–33] is a deterministic global optimization method, which aims to obtain the global optimum of a Lipschitz function. In LUM, a piecewise affine underestimate of an objective function is constructed using a subset of supporting hyperplanes. At each iteration, a new supporting hyperplane is built at the global minima of the current underestimate, which is obtained by linear programming techniques. By adding the new supporting hyperplanes, the underestimate becomes more and more accurate. Consequently, the global minimum of the objective function can be found by enumerating all local minima of the underestimate. LUM is designed specifically for Lipschitz functions, and many practical problems can be formulated with Lipschitz functions [32].

Suppose that the D-dimensional objective function f to be minimized satisfies Lipschitz condition. According to the theory of Lipschitz underestimate [31,34], a lower approximation of f can be

constructed using a set of K supporting vectors of the given points $(x^k, f(x^k))$, $k = 1, 2, \ldots, K$, namely

$$H^K(x) = \max_{k \le K} \min_{i \in I} M\left(l_i^k + x_i\right) \tag{4}$$

where $l_i^k = f(x^k)/M - x_i^k$, $I = \{1, 2, \ldots, D+1\}$, $x_{D+1} = 1 - \sum_{i=1}^{D} x_i$, and M is the Lipschitz constant of f. In [34], every local minimizer of H^K has been proven to correspond to a support matrix L whose rows are the $D+1$ support vectors $l^{k_1}, l^{k_2}, \ldots, l^{k_{D+1}}$:

$$L = \begin{bmatrix} l_1^{k_1} & l_2^{k_1} & \cdots & l_{D+1}^{k_1} \\ l_1^{k_2} & l_2^{k_2} & \cdots & l_{D+1}^{k_2} \\ \vdots & \vdots & \ddots & \vdots \\ l_1^{k_{D+1}} & l_2^{k_{D+1}} & \cdots & l_{D+1}^{k_{D+1}} \end{bmatrix} \tag{5}$$

Let x^* be a local minimizer of H^K, and its value $d = H^K(x^*)$. The matrix L corresponding to x^* enjoys the following properties [31]:

(1) $\forall i, j \in I, i \neq j : l_i^{k_i} < l_i^{k_j}$;
(2) $\forall r \notin \{k_1, k_2, \cdots, k_{D+1}\}, \exists i \in I : L_{ii} = l_i^{k_i} \geq l_i^r$;
(3) $d = \frac{Trace(L)+1}{\sum_{i=1}^{D+1} \frac{1}{M}}$;
(4) $x_i* = \frac{d}{M} - l_i^{k_i}$.

Property (1) shows that the diagonal elements of matrix L are dominated by their respective columns, and Property (2) indicates that no support vector l^r (which is not part of L) strictly dominates the diagonal of L.

3. Differential evolution with local Lipschitz underestimate strategy (LLUDE)

In this paper, we propose LLUDE, an enhanced DE using local Lipschitz underestimate strategy, in order to reduce the number of function evaluations and improve the search capabilities of DE. Unlike LUM, LLUDE does not need to build a highly accurate underestimate of the objective function using numerous supporting hyperplanes. LLUDE constructs the supporting hyperplanes only for some neighboring individuals (measured according to Euclidean distances) of the trial individual. Then the underestimate value of the trial individual can be obtained through the supporting hyperplanes of its neighboring individuals. According to the underestimate value, we can judge whether the trial individual is worth evaluating. Also, the part of the domain where the global optimum cannot be found can be excluded according to the local minimum of the underestimate, and the generalized descent directions of the supporting hyperplanes can be used to guide the local enhancement. In particular, all the supporting hyperplanes are deleted to free the computer memory at the end of each iteration.

3.1. Local Lipschitz underestimate-based judgement

Suppose the trial individual is generated according to Eqs. (1) and (2). Then the Euclidean distances from the trial individual to all individuals in the population are calculated. All individuals are sorted in ascending order based on the Euclidean distances. The supporting vectors of the top n individuals x_{NN}^t, $t = 1, 2, \ldots, n$ (i.e., the n nearest neighboring individuals of the trial individual) are constructed as follow:

$$l_{NN}^t = \left(\frac{f(x_{NN}^t)}{M} - x_{NN,1}^t, \ldots, \frac{f(x_{NN}^t)}{M} - x_{NN,D+1}^t\right) \tag{6}$$

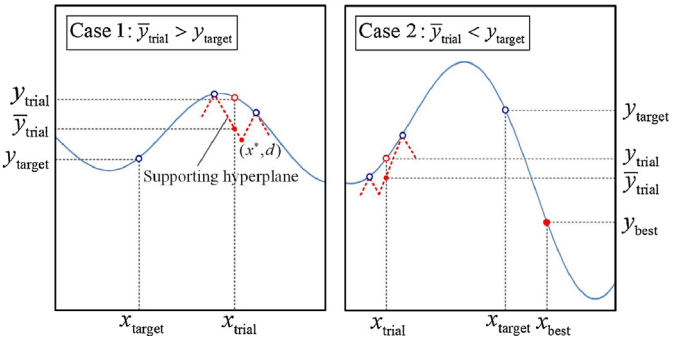


Fig. 1. Illustration of the local Lipschitz underestimate-based judgement.

where $x_{NN,D+1}^t = 1 - \sum_{i=1}^{D} x_{NN,i}^t$, and $f(x_{NN}^t)$ is the objective function value of x_{NN}^t. Based on these n supporting vectors, the underestimate value $\bar{y}_{trial}$ of the trial individual x_{trial} can be obtained by

$$\bar{y}_{trial} = \max_{t \le n} \min_{i \in I} M\left(l_{NN,i}^t + x_{trial,i}\right) \tag{7}$$

Then the comparison between $\bar{y}_{trial}$ and the objective function value y_{target} of the target individual x_{target} will help judge whether or not the trial individual is worth evaluating. Here, two cases may occur in the judgement. Fig. 1 illustrates the judgement for a minimization problem of a single variable function $y(x) = f(x)$:

- Case 1: $\bar{y}_{trial} > y_{target}$. Since the underestimate is always below the objective function, namely $\bar{y}_{trial} < y_{trial}$, hence, $y_{trial} > \bar{y}_{trial} > y_{target}$, then we can judge that x_{trial} worse than x_{target}, and discard x_{trial} without evaluating x_{trial}.
- Case 2: $\bar{y}_{trial} < y_{target}$. In this case, we cannot judge whether x_{trial} is better than x_{target} or not. Hence, we compare $\bar{y}_{trial}$ with the objective function value y_{best} of the best individual x_{best} found so far. If $\bar{y}_{trial} > y_{best}$, then $y_{trial} > \bar{y}_{trial} > y_{best}$, so we can safely exclude x_{trial} from consideration, without evaluating x_{trial}.

In the above cases, the trial vector is completely judged according to its underestimate value that obtained by the supporting hyperplanes of its n nearest neighboring individuals. For maximization problems, the comparison is reversed.

3.2. Invalid region exclusion

Fig. 1 shows that the intersection of the supporting hyperplanes form an underestimate region H^K that corresponds to a subregion S of the search domain Ω ($S \subset \Omega$). Since the local minimum d of H^K that calculated according to property (3) of L is the lower bound of S, so we can conclude that S is an invalid region where the global optimum cannot be included when d is larger than the best value f_{best} found so far. As each H^K corresponds to a support matrix L, and the local minimum d of each H^k is unique. Thus, according to property (1) of L, the subregion S can be denoted as

$$\left(x_j * - x_j^{k_j}\right) < \left(x_i * - x_i^{k_j}\right), \quad \forall i, j \in I, i \neq j \tag{8}$$

where x^* is the local minimizer of H^K that can be calculated according to property (4) of L. When the subregion S is an invalid region, we only need to record its corresponding x^*, and we can conclude that a point x^{k_j} belongs to the subregion S if its all elements satisfy inequality (8).

Therefore, the judgement based on inequality (8) needs to be executed if invalid regions exist to judge whether the trial individual is located in the invalid regions after its generation at each iteration. If the trial individual is included in the invalid regions, the process directly proceeds to the next iteration. By excluding some

invalid regions through the above process, the search can focus on the promising region, thereby improving the search efficiency.

3.3. Generalized descent direction-based local enhancement

According to the theory of Lipschitz underestimate [31], the direction of the supporting hyperplane is generally descending. Hence, the local minimizer of the underestimate may have a lower function value than the trial individual at a high probability. When the underestimate value of the trial individual is lower than the objective function value of the target individual and the trial individual is better than the target individual, a comparison between the local minimizer of the underestimate and the trial individual is performed to obtain better individuals for the new generation, namely

$$x_i^{g+1} = \begin{cases} x*, & \text{if } f(x*) \le f(x_i^g) \\ x_{\text{trial}}, & \text{otherwise} \end{cases} \tag{9}$$

where x^* is the local minimizer of the underestimate H^K that is calculated through property (4) of the support matrix L. Specifically, if x^* obtains a lower objective function value than the target individual x_i^g, replace x_i^g with x^*, otherwise, x_i^g is replaced by x_{trial}.

3.4. Algorithm description

The goal of LLUDE is to reduce the number of function evaluations and enhance the search capability to obtain a desirable solution. The LLUDE algorithm is described as follows:

Step 1. *Initialization*: Set the value of C_R, F, N_P, and the maximum number of function evaluations (MaxFEs).

Step 2. *Generate trial individual*: Run the mutation operation (Eq. (1)) and the crossover operation (Eq. (2)) of DE to generate the trial individual.

Step 3. *Check trial individual*: Check whether or not the trial individual is included in the invalid regions according to inequality (8) when the invalid regions exist. If yes, return to step 2, otherwise, proceed to step 4.

Step 4. *Calculate underestimate*: Calculate the Euclidean distances from the trial individual to all individuals. Sort all individuals in ascending order according to the Euclidean distances. Construct supporting hyperplanes for the top n individuals according to Eq. (6) to form an underestimate region includes the trial individual. Then calculate the underestimate value of the trial individual according to Eq. (7).

Step 5. *Underestimate-based selection operation*: Compare the underestimate value of the trial individual with the objective function value of the corresponding target individual. If the underestimate value of the trial individual is equal to or larger than the objective function value of the target individual, we can know that the trial individual is worse than the target individual, so the target individual is retained without evaluating the trial individual and the process continues to step 6. If the underestimate value of the trial individual is smaller than the function value of the target individual but larger than the best function value found so far, the target individual is retained without evaluating the trial individual and proceed to step 6. Otherwise, employ the selection operation of DE.

Step 6. *Exclude invalid regions*: Calculate the local minimum of the underestimate region which includes the trial individual according to property (3) of L. If the local minimum is equal to or larger than the function value of the best individual in the current population, consider the subregion of the domain corresponded to the underestimate region as an invalid region where the global optimum cannot be found.

Step 7. *Local enhancement*: If the trial individual is better than the target individual, calculate the local minimizer of the underestimate region according to property (4) of L. If the point in the objective function that corresponded to the local minimizer is better than the trial individual, the point replaces the trial individual.

Step 8. *Check convergence*: Delete all the supporting hyperplanes and repeat steps 2–7 until the stopping criterion is met.

The above procedure shows that the proposed LLUDE is a simple and straightforward global optimization algorithm. It can guide DE to achieve an acceptable global optimum with fewer function evaluations. The computation time taken to build the local underestimate of the objective function can be ignored compared with the computational time of the actual function evaluations for computationally expensive problems. This is because we only construct the supporting hyperplanes for the n nearest neighboring individuals of the trial individual, and all these supporting hyperplanes are deleted at the end of each iteration. Moreover, the computational time needed in evaluating the function of computationally expensive optimization problems is often very large, that the time taken for the local Lipschitz underestimate will be comparatively small. The procedure of LLUDE is shown in Appendix B.

3.5. Runtime complexity analysis

The runtime of a basic DE is $O(N_P \cdot D \cdot G_{\max})$, where $G_{\max}$ is the maximum number of generations. The proposed LLUDE differs from the basic DE mainly in the local Lipschitz underestimate strategy.

Three operations are included in the local Lipschitz underestimate strategy. For the local Lipschitz underestimate-based judgement, the n nearest neighboring individuals of the trial individual are selected to construct the supporting hyperplanes. The runtime complexity of computing the distances between the trial individual and all individuals is $O(N_P \cdot D)$ since we use Euclidean distance. The procedure of finding the n nearest neighboring individuals can be completed in $O(N_P \cdot n)$ time. The construction of the supporting hyperplanes for the n nearest neighboring individuals of the trial individual requires $O(n \cdot (D+1))$ runtime. The process of calculating the underestimate value of the trial individual also can be completed in $O(n \cdot (D+1))$ time. Hence, over $G_{\max}$ generations, the overall runtime complexity of the local Lipschitz underestimate-based judgement is $O(\max(N_P \cdot D \cdot G_{\max}, N_P \cdot n \cdot G_{\max}, n \cdot (D+1) \cdot G_{\max}))$.

For the invalid region exclusion, we need to calculate the local minimum of the underestimate region and determine the best solution found so far, respectively. If the invalid region exist, we also need to judge whether the trial individual is included in the invalid region. The process of calculating the local minimum of the underestimate region can be completed in $O(1)$ time. The determination of the best solution found so far requires $O(N_P - 1)$ runtime. The runtime complexity of judging whether the trial individual is located in the invalid region is $O(\log_D T \cdot (D+1))$ (T is the number of leaf nodes) since we use the n-ary tree data structure to save the underestimate value. Hence, over $G_{\max}$ generations, the overall runtime complexity is $O(\max((N_P - 1) \cdot G_{\max}, \log_D T \cdot (D+1) \cdot G_{\max}))$.

For the generalized descent direction-based local enhancement, we only need to calculate the local minimum of the underestimate region. Thus, over $G_{\max}$ generations, the local enhancement requires $O((D+1) \cdot G_{\max})$ runtime.

The invalid region exclusion and generalized descent direction-based local enhancement are performed only when certain conditions are met. If all the above three operations are performed at every generation, the overall runtime complexity of LLUDE is $O(\max(N_P \cdot D \cdot G_{\max},\ N_P \cdot n \cdot G_{\max},\ \log_D T \cdot (D+1) \cdot G_{\max}))$. In LLUDE, only two neighboring individuals ($n = 2$) of the trial individual are used to construct the supporting hyperplanes. Thus, the number T of leaf nodes in the data structure is also less than D. If $D > n$,

Table 1
18 benchmark functions used in the experimental studies.

Name	Function	Search range	Global optimum
Sphere	$f_1(x) = \sum_{i=1}^{D} x_i^2$	$[-100, 100]$	0
SumSquares	$f_2(x) = \sum_{i=1}^{D} i x_i^2$	$[-10, 10]$	0
Schwefel 2.22	$f_3(x) = \sum_{i=1}^{D} \lvert x_i \rvert + \prod_{i=1}^{D} \lvert x_i \rvert$	$[-10, 10]$	0
Exponential	$f_4(x) = -\exp\left(-0.5 \sum_{i=1}^{D} x_i^2\right)$	$[-1, 1]$	-1
Tablet	$f_5(x) = 10^6 x_1^2 + \sum_{i=2}^{D} x_i^2$	$[-100, 100]$	0
Step	$f_6(x) = \sum_{i=1}^{D} (x_i + 0.5)^2$	$[-100, 100]$	0
Zakharov	$f_7(x) = \sum_{i=1}^{D} x_i^2 + \left(\sum_{i=1}^{D} 0.5 i x_i\right)^2 + \left(\sum_{i=1}^{D} 0.5 i x_i\right)^4$	$[-5, 10]$	0
Rosenbrock	$f_8(x) = \sum_{i=1}^{D-1} \left(100(x_{i+1} - x_i^2)^2 + (x_i - 1)^2\right)$	$[-30, 30]$	0
Griewank	$f_9(x) = 1 + \frac{1}{4000} \sum_{i=1}^{D} x_i^2 - \prod_{i=1}^{D} \cos\left(\frac{x_i}{\sqrt{i}}\right)$	$[-600, 600]$	0
Schaffer 2	$f_{10}(x) = \sum_{i=1}^{D-1} \left(x_i^2 + x_{i+1}^2\right)^{0.25} \left(\sin^2(50(x_i^2 + x_{i+1}^2)^{0.1}) + 1\right)$	$[-100, 100]$	0
Schwefel 2.26	$f_{11}(x) = -\sum_{i=1}^{D} \left(x_i \sin(\sqrt{\lvert x_i \rvert})\right)$	$[-500, 500]$	$-418.983D$
Himmelblau	$f_{12}(x) = \frac{1}{D} \sum_{i=1}^{D} \left(x_i^4 - 16x_i^2 + 5x_i\right)$	$[-100, 100]$	-78.3323
Levy and Montalvo 1	$f_{13}(x) = \frac{\pi}{D} \left(10\sin^2(\pi y_1) + \sum_{i=1}^{D-1} (y_i - 1)^2 \left(1 + 10\sin^2(\pi y_i + 1)\right) + (y_D - 1)^2\right), y_i = 1 + \frac{1}{4}(x_i + 1)$	$[-10, 10]$	0
Levy and Montalvo 2	$f_{14}(x) = 0.1 \left(\sin^2(3\pi x_1) + \sum_{i=1}^{D-1} (x_i - 1)^2 \left(1 + \sin^2(3\pi x_{i+1}) + (x_D - 1)^2 \left(1 + \sin^2(2\pi x_D)\right)\right)\right)$	$[-5, 5]$	0
Ackley	$f_{15}(x) = -20 \exp\left(-0.02\sqrt{D^{-1} \sum_{i=1}^{D} x_i^2}\right) - \exp\left(D^{-1} \sum_{i=1}^{D} \cos(2\pi x_i)\right) + 20 + e$	$[-30, 30]$	0
Rastrigin	$f_{16}(x) = 10D + \sum_{i=1}^{D} (x_i^2 - 10\cos(2\pi x_i))$	$[-5, 5]$	0
Penalized 1	$f_{17}(x) = \frac{\pi}{D} \{\sum_{i=1}^{D-1} (y_i - 1)^2 [1 + \sin(\pi y_{i+1})] + (y_D - 1)^2 + (10\sin^2(\pi y_1))\} + \sum_{i=1}^{D} u(x_i, 10, 100, 4)$, $y_i = 1 + \frac{x_i + 1}{4}$; $u(x_i, a, k, m) = \begin{cases} k(x_i - a)^m, & x_i > a \\ 0, & -a \le x_i \le a \\ k(-x_i - a)^m, & x_i < -a \end{cases}$	$[-50, 50]$	0
Penalized 2	$f_{18}(x) = 0.1\{\sin^2(3\pi x_1) + \sum_{i=1}^{D-1} (x_i - 1)^2 [1 + \sin^2(3\pi x_{i+1})] + (x_D - 1)^2 [1 + \sin^2(2\pi x_D)]\} + \sum_{i=1}^{D} u(x_i, 5, 100, 4)$	$[-50, 50]$	0

the overall runtime complexity of LLUDE is $O(N_P \cdot D \cdot G_{\max})$. In this case, LLUDE does not impose any serious burden on the runtime complexity.

4. Experimental studies

4.1. Benchmark functions and parameter settings

To verify the performance of the proposed LLUDE, a set of 18 well-known benchmark functions are used in the following experiments. All these functions have been considered in [35,36]. Among these functions, f_1–f_8 are unimodal functions. f_6 is a step function that has one minimum and is discontinuous. f_8 is the Rosenbrock function, it becomes multimodal when $D > 3$. f_9–f_{18} are multimodal functions, which the local minima increases exponentially with the dimension. All the above functions used in our experiments are problems to be minimized. The descriptions of these benchmark functions are presented in Table 1.

For LLUDE, the following parameters are used unless a change is mentioned: scaling factor $F = 0.5$, crossover rate $CR = 0.9$, and population size $N_P = 50$. The number n of the neighboring individuals of the trial individual that are used to construct the supporting hyperplanes in LLUDE is set to 2. For each algorithm and each function, 30 independent runs are executed. All experiments are implemented by MATLAB 8.2 and Microsoft Visual Studio 2010 on a computer with the Intel Core i7 2.4 GHZ CPU, 8 GB RAM, and Microsoft Windows 7 operating system.

4.2. Performance criteria

Five performance criteria are selected to evaluate the performance of the proposed algorithm. These criteria are described as follows:

(1) Function evaluations (FEs): The number of function evaluations is recorded when the best function value found so far is below the predefined threshold value within the maximum function evaluations (MaxFEs).
(2) Success rate (SR): If the best function value found within the MaxFEs is below the predefined threshold value, then it is deemed a successful run. The SR is calculated as the number of successful runs divided by the total number of runs.
(3) Performance ratio (PR): Firstly, the success performance of each algorithm (a) on each test problem (p) are calculated by

$$s_{a,p} = \frac{\text{Mean}(FEs)}{SR}$$

Then the performance ratio can be defined as

$$r_{a,p} = \frac{s_{a,p}}{\min\{s_{a,p} : a \in A\}}$$

where A is the set of comparison algorithms. If $r_{a,p}$ is equal to 1, the algorithm (a) is considered the best algorithm in the overall performance of FEs and SR for the test problem (p).

Table 2
Results of mean FEs, SR and PR of SaDE, JADE, CoDE and LLUDE.

Fun	*D*	Threshold	SaDE			JADE			CoDE			LLUDE		
			FEs	SR	PR	FEs	SR	PR	FEs	SR	PR	FEs	SR	PR
f_1	30	1.00E−05	20,297	1.00	1.58	23,557	1.00	1.83	40,155	1.00	3.13	**12,841**	**1.00**	**1.00**
f_2	30	1.00E−05	18,410	1.00	1.57	21,600	1.00	1.85	36,270	1.00	3.10	**11,704**	**1.00**	**1.00**
f_3	30	1.00E−05	**24,368**	**1.00**	**1.00**	35,787	1.00	1.47	56,421	1.00	2.32	33,225	1.00	1.36
f_4	30	0.99999	10,993	1.00	1.52	13,377	1.00	1.84	22,362	1.00	3.08	**7,253**	**1.00**	**1.00**
f_5	30	1.00E−05	**21,117**	**1.00**	**1.00**	27,007	1.00	1.28	41,286	1.00	1.96	29,250	1.00	1.39
f_6	30	1.00E−05	**10,657**	**1.00**	**1.00**	11,790	1.00	1.11	20,070	1.00	1.88	12,474	1.00	1.17
f_7	30	1.00E−05	137,487	1.00	2.38	**56,107**	0.97	**1.00**	77,979	1.00	1.35	84,941	1.00	1.47
f_8	30	1.00E−05	NA	0.00	4.87	97,793	1.00	1.59	236,957	1.00	3.84	**61,651**	**1.00**	**1.00**
f_9	30	1.00E−05	21,740	0.73	1.87	26,893	0.97	1.74	43,878	1.00	2.75	**15,949**	**1.00**	**1.00**
f_{10}	30	1.00E−05	**71,833**	**1.00**	**1.00**	279,660	1.00	3.89	172,776	1.00	2.41	79,772	1.00	1.11
f_{11}	30	−12569.48	36,050	1.00	1.65	82,707	0.97	3.90	59,982	1.00	2.74	**21,878**	**1.00**	**1.00**
f_{12}	30	−78.33229	22,873	1.00	1.14	56,503	1.00	2.81	35,565	1.00	1.77	**20,112**	**1.00**	**1.00**
f_{13}	30	1.00E−05	12,260	1.00	1.15	17,340	1.00	1.63	26,307	1.00	2.48	**10,619**	**1.00**	**1.00**
f_{14}	30	1.00E−05	12,855	1.00	1.11	17,373	1.00	1.50	26,964	1.00	2.33	**11,580**	**1.00**	**1.00**
f_{15}	30	1.00E−05	**30,453**	0.93	1.07	33,080	1.00	1.08	56,826	1.00	1.85	30,656	**1.00**	**1.00**
f_{16}	30	1.00E−05	62,508	0.93	2.37	109,470	1.00	3.86	130,409	0.97	4.74	**28,389**	**1.00**	**1.00**
f_{17}	30	1.00E−05	14,852	1.00	1.42	19,710	1.00	1.88	30,324	1.00	2.90	**10,471**	**1.00**	**1.00**
f_{18}	30	1.00E−05	17,421	0.87	1.19	22,363	1.00	1.33	35,385	1.00	2.11	**16,774**	**1.00**	**1.00**
Total average			47,010	0.914	1.604	52,895	0.995	1.977	63,884	0.998	2.596	**27,752**	**1.000**	**1.083**

(4) Error: The function error of a solution x is defined as $f(x) - f(x^*)$, where x^* is the global optimum of the function. The average and standard deviation of the best error are recorded when the FEs reaches the MaxFEs.
(5) Convergence graphs: The convergence graphs show the mean convergence characteristics of the best solution over the total runs, in the respective experiments.

4.3. Comparison with three state-of-the-art DE

In this section, LLUDE is compared with three state-of-the-art DE algorithms, i.e., SaDE [37], JADE [38], and CoDE [39]. SaDE, proposed by Qin et al. [37], is a self-adaptive DE, in which both trial vector generation strategies and their associated control parameter settings are self-adapted according to the previous experiences. CoDE was proposed by [39]. In CoDE, several effective trial vector generation strategies are randomly combined with some suitable control parameter values to generate trial vectors. JADE, proposed by Zhang and Sanderson [38], is an adaptive DE with optional external archive. In JADE, a new mutation strategy "DE/current-to-*p*best" was proposed to improve the optimization performance and the control parameters for each individual are automatically updated according to their historical record of success. The control parameters for the above three DE algorithms are set the same as in the original literatures. For a fair comparison, the parameters self-adaptive technique proposed in SaDE is adopted in LLUDE.

4.3.1. Comparison of the convergence speed and SR

Table 2 summarizes the results of the mean FEs, SR, and PR of SaDE, JADE, CoDE, and LLUDE when they converge to the threshold value before meeting MaxFEs = 3.00E+05. The best results are marked in **boldface**, and "NA" indicates that no runs of the corresponding algorithm converged below the predefined threshold within the MaxFEs. As seen, LLUDE requires less FEs than other three algorithms for the majority of benchmark functions. To be specific, SaDE shows faster convergence speed than LLUDE on three unimodal functions (f_3, f_5, and f_6) and two multimodal functions (f_{10} and f_{15}). Particularly, SaDE fails to solve f_8, while LLUDE successfully converges to the threshold value with 1.0 SR. For f_{15}, although SaDE costs less FEs to reach the threshold value, it obtains the lowest SR. As PR is a criteria that measures the overall performance of FEs and SR, the PR of SaDE is lower than LLUDE for f_8. Hence SaDE only outperforms LLUDE on four functions in terms of PR. JADE converges faster than LLUDE on three functions (f_5, f_6, and f_7). For f_7, although JADE is better than LLUDE in terms of FEs and PR, its SR is lower than LLUDE. CoDE only performs better than LLUDE on one function (f_7). Moreover, based on the results of the total average FEs listed in the last row of the table, LLUDE costs the lowest FEs (i.e., 27,752) to converge to the threshold value, while SaDE, JADE and CoDE cost 47,010, 52,895, and 63,884 FEs, respectively. Namely, LLUDE saves 40.97%, 47.53%, and 56.56% of FEs compared to SaDE, JADE, and CoDE, respectively. In addition, LLUDE achieves the highest SR (i.e., 1.000) because it successfully solves all functions with 1.00 SR. SaDE fails to solve one function and obtains the lowest SR (i.e., 0.914). The SR of JADE and CoDE is 0.995 and 0.998, respectively. In terms of PR, LLUDE also outperforms other three algorithms, and followed by SaDE, JADE, and CoDE.

Additionally, the performance profiles [40] are also depicted to verify the superior performance of LLUDE in terms of PR. The performance profiles are a tool to compare the performance using a comparison goal. The goal is the PR in this comparison. Denoting the probability that the PR ($r_{a,p}$) of algorithm $a \in A$ is within a factor $\tau \geq 1$ of the best possible ratio as follow:

$$\rho_a(\tau) = \frac{1}{n_p} |\{p \in P : r_{a,p} \leq \tau\}$$

where P is the set of problems. The $\rho_a(\tau)$ function is the distribution function of the PR. Small values of τ and large values of $\rho_a(\tau)$ in the graph are preferred. That is, the first one that reaches the level of 1 with small values of τ is considered the best algorithm.

Fig. 2 shows the performance profiles of SaDE, JADE, CoDE, and LLUDE. From this figure, it is observed that LLUDE is the best algorithm as it reaches 1 with τ equals to 1.47, and JADE is the second as it reaches the probability of 1 with τ of 3.90. SaDE shows competitive performance with CoDE. However, CoDE is better than SaDE with lower value of τ.

Furthermore, the mean convergence curves of SaDE, JADE, CoDE, and LLUDE on some selected functions are presented in Fig. 3. As shown in the figure, it is obvious that the proposed LLUDE always converges faster than other three algorithms on these four functions. Specially, f_8 is Rosenbrock function, it becomes multimodal when $D > 3$. It is easy to find a local optimum but difficult to find the global minimum which locates in a long narrow parabolic shaped

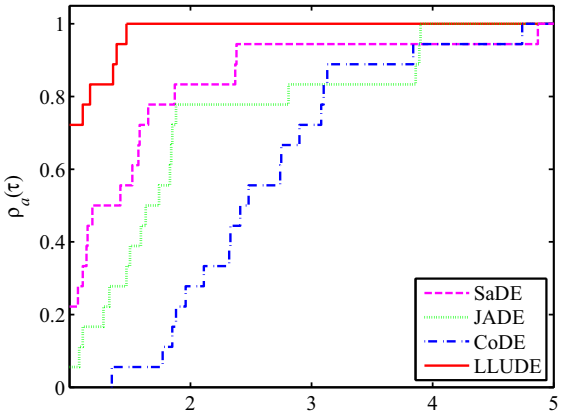


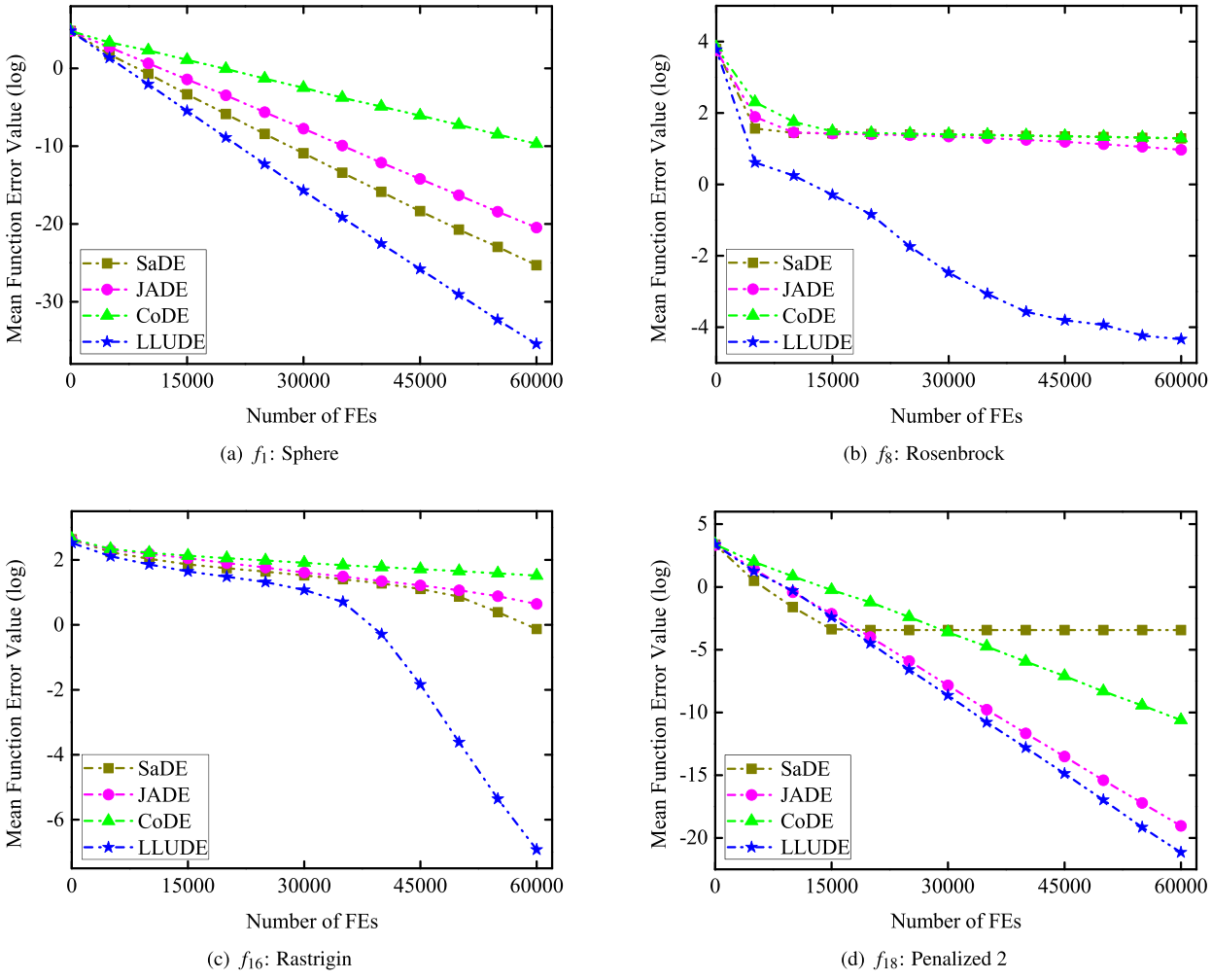
Fig. 2. Performance profiles of SaDE, JADE, CoDE, and LLUDE.

flat valley. However, only LLUDE can steadily converge to the global optimum, while other three algorithms are terminated as the premature convergence. For f_1, LLUDE is the fastest, and followed by SaDE. For f_{16}, LLUDE is significantly faster than other competitors. For f_{18}, although SaDE converges faster than LLUDE in the beginning of the evolutionary process, it gets trapped into a local minimum soon, while LLUDE is able to reach the solution with the fastest convergence speed.

4.3.2. Comparison of final solutions quality

Table 3 summarizes mean function error values and standard deviation obtained by SaDE, JADE, CoDE, and LLUDE within the MaxFEs= $2000 \times D$ for all benchmark functions at $D = 30$. In the table, "Mean Error" represents the mean function error value, and "Std Dev" indicates the corresponding standard deviation. To study the difference between any two stochastic algorithms in a more meaningful way, the paired Wilcoxon signed-rank test at 5% significance level is conducted on the experimental results. We mark with "+" when the first algorithm is significantly better than the second, with "≈" when there is no significant difference between the two algorithms, and with "−" when the first algorithm is significantly worse than the second. According to the Wilcoxon's test, the results summarized as "$w/t/l$" in the last row of the table represent that LLUDE wins on w functions, ties on t functions, and loses on l functions, compared with its corresponding competitor.

From the results listed in Table 3, we can find that LLUDE is superior to other three algorithms on most of the benchmark functions. All algorithms find the global solution for f_6 and f_{12}. From the results listed in the last row of the table, LLUDE significantly outperforms SaDE on 11 out of 18 functions, while SaDE performs significantly better than LLUDE only on 3 functions including 2 unimodal functions (f_3 and f_5) and one multimodal function (f_{10}). Compared to JADE and CoDE, LLUDE shows significantly better performance than them on 15 and 16 out of 18 functions, respectively, while JADE and CoDE does not significantly outperform LLUDE on any functions.



(a) f_1: Sphere

(b) f_8: Rosenbrock

(c) f_{16}: Rastrigin

(d) f_{18}: Penalized 2

Fig. 3. Mean convergence curves of SaDE, JADE, CoDE, and LLUDE on some selected benchmark functions (f_1, f_8, f_{16} and f_{18}).

Table 3
Mean function error values and standard deviation of SaDE, JADE, CoDE, and LLUDE on all benchmark functions for $D = 30$.

Fun	D	MaxFEs	SaDE	JADE	CoDE	LLUDE
			Mean error ± Std Dev	Mean error ± Std Dev	Mean error ± Std Dev	Mean error ± Std Dev
f_1	30	6.00E+04	1.29E−23 ± 3.07E−23+	3.01E−20 ± 8.74E−20+	2.16E−10 ± 1.73E−10+	**1.58E−37 ± 2.03E−37**
f_2	30	6.00E+04	6.59E−25 ± 8.39E−25+	1.16E−21 ± 1.59E−21+	2.83E−11 ± 2.61E−11+	**3.44E−30 ± 2.82E−30**
f_3	30	6.00E+04	**8.70E−16 ± 5.17E−16**−	3.11E−10 ± 3.36E−10+	3.86E−06 ± 1.32E−06+	2.83E−11 ± 3.97E−11
f_4	30	6.00E+04	**0.00E+00 ± 0.00E+00**≈	**0.00E+00 ± 0.00E+00**≈	1.30E−14 ± 1.13E−14+	**0.00E+00 ± 0.00E+00**
f_5	30	6.00E+04	**8.85E−23 ± 4.35E−22**−	1.83E−18 ± 6.07E−18+	4.04E−10 ± 3.04E−10+	5.29E−21 ± 7.57E−21
f_6	30	6.00E+04	**0.00E+00 ± 0.00E+00**≈	**0.00E+00 ± 0.00E+00**≈	**0.00E+00 ± 0.00E+00**≈	**0.00E+00 ± 0.00E+00**
f_7	30	6.00E+04	9.27E−02 ± 1.30E−01+	6.16E−03 ± 3.13E−02+	5.69E−04 ± 7.53E−04+	**1.48E−04 ± 2.36E−04**
f_8	30	6.00E+04	5.07E+01 ± 3.36E+01+	9.29E+00 ± 1.06E+00+	1.97E+01 ± 5.80E−01+	**1.31E−05 ± 1.56E−05**
f_9	30	6.00E+04	2.22E−03 ± 4.73E−03+	9.70E−15 ± 5.25E−14+	2.47E−07 ± 7.34E−07+	**0.00E+00 ± 0.00E+00**
f_{10}	30	6.00E+04	**7.51E−04 ± 7.06E−04**−	3.24E+00 ± 1.23E+00+	1.97E+00 ± 4.24E−01+	1.77E−02 ± 5.26E−03
f_{11}	30	6.00E+04	**0.00E+00 ± 0.00E+00**≈	1.07E+01 ± 2.20E+01+	1.66E−02 ± 3.13E−02+	**0.00E+00 ± 0.00E+00**
f_{12}	30	6.00E+04	**0.00E+00 ± 0.00E+00**≈	**0.00E+00 ± 0.00E+00**≈	**0.00E+00 ± 0.00E+00**≈	**0.00E+00 ± 0.00E+00**
f_{13}	30	6.00E+04	2.47E−27 ± 4.90E−27+	3.69E−21 ± 1.55E−20+	1.80E−13 ± 3.36E−13+	**1.13E−30 ± 9.39E−31**
f_{14}	30	6.00E+04	1.46E−03 ± 3.80E−03+	5.07E−22 ± 2.22E−21+	1.31E−13 ± 1.51E−13+	**1.35E−32 ± 2.25E−34**
f_{15}	30	6.00E+04	2.40E−01 ± 4.45E−01+	4.19E−11 ± 4.49E−11+	3.75E−06 ± 1.88E−06+	**7.55E−15 ± 1.49E−15**
f_{16}	30	6.00E+04	4.11E−02 ± 1.82E−01+	4.76E+00 ± 8.78E−01+	3.30E+01 ± 5.81E+00+	**6.27E−07 ± 5.35E−07**
f_{17}	30	6.00E+04	3.46E−03 ± 1.89E−02+	5.65E−21 ± 1.54E−20+	1.44E−12 ± 1.26E−12+	**4.37E−26 ± 5.15E−26**
f_{18}	30	6.00E+04	1.83E−03 ± 4.16E−03+	3.60E−20 ± 7.81E−20+	2.45E−11 ± 2.36E−11+	**3.93E−21 ± 4.40E−21**
$w/t/l$			11/4/3	15/3/0	16/2/0	−/−/−

Overall, LLUDE requires less function evaluations in obtaining better quality solutions, and has higher success rates, compared to SaDE, JADE and CoDE.

4.3.3. Scalability of LLUDE

In order to demonstrate the scalability of LLUDE, we further compare LLUDE with the three DE variants on all benchmark functions at $D = 50$ and 100.

Table 4 presents the results achieved by these four DE variants within the MaxFEs= $2000 \times D$ for all benchmark functions at $D = 50$. From the data, we can see that the proposed LLUDE also performs better than SaDE, JADE, and CoDE on the majority of functions. According to the results obtained by the Wilcoxon signed-rank test, SaDE significantly outperforms LLUDE on 3 functions, while LLUDE is significantly superior to SaDE on 11 out of 18 functions. JADE achieves significantly better results than LLUDE on 4 functions including 3 unimodal functions (f_2, f_3, and f_5) and one multimodal function f_{17}), while LLUDE shows significantly better performance than JADE on 14 out of 18 functions. CoDE does not significantly outperform LLUDE on any functions, while LLUDE is significantly better than CoDE on 17 out of 18 functions.

Table 5 shows the mean function error value and standard deviation obtained by SaDE, JADE, CoDE, and LLUDE within 300,000 FEs for all benchmark functions at $D = 100$. From the results, we can see that the proposed LLUDE also performs better than the three DE variants on the majority of functions. According to the results listed in the last row of the table, the LLUDE significantly outperforms SaDE on 12 out of 18 functions, while SaDE is significantly better than LLUDE only on 2 functions (f_2 and f_5). JADE is significantly superior to LLUDE on 5 functions including 3 unimodal functions and 2 multimodal functions, but LLUDE shows significantly better performance than JADE on 11 functions. LLUDE achieves significantly better results than CoDE on 14 out of 18 functions, while CoDE significantly outperforms LLUDE only on one function (f_7).

The above results and analysis demonstrate that LLUDE remains good at obtaining high quality solutions when the dimension increased to 50 and 100.

4.4. Comparison with three non-DE algorithms

In this section, LLUDE is compared with three non-DE algorithms, i.e., CLPSO [41], CMA-ES [42], and GL-25 [43]. CLPSO, proposed by Liang et al. [41], 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, a very famous and efficient evolution strategy, is proposed by Hansen et al. [42]. In CMA-ES, a multivariate normal distribution is used to sample the new solutions, and the distribution is updated using the covariance matrix adaptation (CMA) method. GL-25, proposed by Garcia-Martinez [43], is a variant of real-coded genetic algorithm that 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. Each algorithm stops when FEs reaches the MaxFEs. In this comparison, MaxFEs= 150, 000.

Table 6 reports the results achieved by CLPSO, CMA-ES, GL-25, and LLUDE for all benchmark functions at $D = 30$. As seen, LLUDE is significantly better than its three competitors on all functions except for f_2, f_3, f_5 and f_7. All algorithms achieve the global optimum for f_6. CLPSO does not significantly outperform LLUDE on any functions. It just ties on 3 functions compared with LLUDE. CMA-ES only significantly outperforms LLUDE for f_7, while LLUDE shows significantly better performance than CMA-ES on 16 out of 18 functions. GL-25 obtains significantly better results than LLUDE on 3 unimodal functions (f_2, f_3 and f_5), but it does not significantly outperform LLUDE on any multimodal functions.

The results obtained by CLPSO, CMA-ES, GL-25, and LLUDE for all benchmark functions at $D = 50$ are summarized in Table 7. From the results, we find that LLUDE exhibits better performance than the three competitors on most of the benchmark functions. To be specific, LLUDE significantly outperforms CLPSO on 17 out of 18 functions, while CLPSO is not superior to LLUDE on any functions. Although CMA-ES is significantly better than LLUDE on 2 functions (f_3 and f_7), LLUDE shows significantly better performance than CMA-ES on 13 out of 18 functions. Compared to GL-25, LLUDE performs significantly better results on 16 out of 18 functions, while GL-25 significantly outperforms LLUDE only on one function (f_5).

4.5. Test on CEC 2015 shifted and rotated benchmarks

To further verify the performance of LLUDE, the CEC 2015 shifted and rotated benchmark functions are utilized in the following experiment. The definitions and properties of them can be found in [44].

In this experiment, we compare LLUDE with SaDE [37], JADE [38], and CoDE [39]. For each algorithm and each function, the

Table 4
Mean function error values and standard deviation of SaDE, JADE, CoDE, and LLUDE on all benchmark functions for $D = 50$.

Fun	D	MaxFEs	SaDE	JADE	CoDE	LLUDE
			Mean error ± Std Dev	Mean error ± Std Dev	Mean error ± Std Dev	Mean error ± Std Dev
f_1	50	1.00E+05	9.21E−26 ± 9.49E−26+	8.32E−58 ± 2.63E−57+	1.57E−12 ± 1.40E−12+	**5.07E−60 ± 6.67E−60**
f_2	50	1.00E+05	1.22E−26 ± 1.24E−26+	**1.80E−40 ± 5.68E−40**−	1.46E−13 ± 7.58E−14+	1.95E−27 ± 1.33E−27
f_3	50	1.00E+05	1.64E−16 ± 8.95E−17−	**6.46E−27 ± 1.29E−26**−	1.55E−07 ± 7.15E−08+	1.40E−10 ± 9.20E−11
f_4	50	1.00E+05	**0.00E+00 ± 0.00E+00**≈	1.55E−16 ± 7.76E−17+	4.62E−15 ± 4.04E−15+	**0.00E+00 ± 0.00E+00**
f_5	50	1.00E+05	4.28E−25 ± 6.16E−25−	**4.52E−52 ± 1.43E−51**−	1.26E−12 ± 1.19E−12+	2.96E−16 ± 2.35E−16
f_6	50	1.00E+05	**0.00E+00 ± 0.00E+00**≈	6.30E+00 ± 6.07E+00+	**0.00E+00 ± 0.00E+00**≈	**0.00E+00 ± 0.00E+00**
f_7	50	1.00E+05	9.15E−02 ± 5.88E−02+	1.32E+01 ± 3.21E+01+	4.32E−02 ± 4.36E−02+	**6.72E−03 ± 5.72E−03**
f_8	50	1.00E+05	3.81E+01 ± 2.16E+00+	9.90E+00 ± 3.18E+00+	3.86E+01 ± 4.03E−01+	**2.09E−05 ± 1.73E−05**
f_9	50	1.00E+05	7.61E−03 ± 1.69E−02+	2.71E−03 ± 5.97E−03+	9.35E−13 ± 4.92E−13+	**0.00E+00 ± 0.00E+00**
f_{10}	50	1.00E+05	6.79E−01 ± 1.73E+00≈	1.16E+00 ± 3.57E+00+	9.79E−01 ± 2.02E−01+	**6.55E−01 ± 1.67E−01**
f_{11}	50	1.00E+05	2.37E+01 ± 4.99E+01+	4.97E+02 ± 2.00E+02+	1.89E+01 ± 2.53E+01+	**0.00E+00 ± 0.00E+00**
f_{12}	50	1.00E+05	**0.00E+00 ± 0.00E+00**≈	2.94E+00 ± 1.03E+00+	3.14E−05 ± 4.62E−12+	**0.00E+00 ± 0.00E+00**
f_{13}	50	1.00E+05	**9.17E−30 ± 9.26E−30**−	6.22E−03 ± 1.97E−02+	1.28E−16 ± 1.15E−16+	2.61E−21 ± 6.19E−22
f_{14}	50	1.00E+05	1.10E−03 ± 3.47E−03+	3.20E−03 ± 7.15E−03+	5.42E−16 ± 3.49E−16+	**7.64E−20 ± 8.53E−21**
f_{15}	50	1.00E+05	8.40E−01 ± 6.12E−01+	1.41E+00 ± 3.75E−01+	1.74E−07 ± 9.59E−08+	**1.41E−09 ± 6.40E−10**
f_{16}	50	1.00E+05	4.99E+00 ± 4.75E+00+	9.95E−02 ± 3.15E−01+	8.86E+01 ± 1.10E+01+	**1.10E−06 ± 6.95E−07**
f_{17}	50	1.00E+05	2.49E−02 ± 3.21E−02+	**1.38E−32 ± 5.69E−33**−	1.89E−15 ± 1.56E−15+	9.29E−18 ± 5.37E−18
f_{18}	50	1.00E+05	2.20E−03 ± 4.63E−03+	2.33E−02 ± 7.00E−02+	7.16E−14 ± 7.83E−14+	**7.80E−17 ± 2.69E−17**
$w/t/l$			11/4/3	14/0/4	17/1/0	−/−/−

Table 5
Mean function error values and standard deviation of SaDE, JADE, CoDE, and LLUDE on all benchmark functions for $D = 100$.

Fun	D	MaxFEs	SaDE	JADE	CoDE	LLUDE
			Mean error ± Std Dev	Mean error ± Std Dev	Mean error ± Std Dev	Mean error ± Std Dev
f_1	100	3.00E+05	1.10E−33 ± 2.25E−33+	9.71E−57 ± 2.47E−56+	1.72E−24 ± 9.12E−25+	**5.92E−60 ± 2.81E−60**
f_2	100	3.00E+05	9.04E−35 ± 1.46E−34−	**1.10E−56 ± 2.26E−56**−	1.05E−24 ± 8.04E−25+	1.35E−32 ± 3.03E−33
f_3	100	3.00E+05	8.45E−20 ± 1.40E−19≈	**9.79E−24 ± 2.01E−23**−	1.55E−14 ± 1.80E−14+	6.35E−20 ± 1.84E−20
f_4	100	3.00E+05	**0.00E+00 ± 0.00E+00**≈	9.99E−01 ± 2.68E−03+	2.22E−17 ± 4.68E−17+	**0.00E+00 ± 0.00E+00**
f_5	100	3.00E+05	1.86E−34 ± 4.48E−34−	**2.22E−55 ± 4.20E−55**−	3.14E−24 ± 4.53E−24+	3.89E−29 ± 1.05E−29
f_6	100	3.00E+05	**0.00E+00 ± 0.00E+00**≈	3.80E+00 ± 3.36E+00+	**0.00E+00 ± 0.00E+00**≈	**0.00E+00 ± 0.00E+00**
f_7	100	3.00E+05	6.43E+00 ± 2.38E+00+	5.07E+01 ± 1.60E+02+	**4.23E−01 ± 4.10E−01**−	3.78E+00 ± 3.32E−01
f_8	100	3.00E+05	1.53E+02 ± 3.97E+01+	5.40E+01 ± 3.62E+01+	1.42E+02 ± 5.90E+01+	**4.08E−02 ± 4.02E−02**
f_9	100	3.00E+05	2.95E−03 ± 6.33E−03+	**0.00E+00 ± 0.00E+00**≈	**0.00E+00 ± 0.00E+00**≈	**0.00E+00 ± 0.00E+00**
f_{10}	100	3.00E+05	1.64E+01 ± 1.43E+01+	4.77E+00 ± 1.61E+00+	6.00E+00 ± 7.13E+00+	**5.42E−03 ± 3.41E−03**
f_{11}	100	3.00E+05	9.48E+01 ± 1.83E+02+	1.19E+01 ± 3.74E+01+	1.42E+02 ± 1.22E+02+	**0.00E+00 ± 0.00E+00**
f_{12}	100	3.00E+05	2.83E−01 ± 2.31E−01+	8.48E−02 ± 1.37E−01+	3.96E−01 ± 3.32E−01+	**0.00E+00 ± 0.00E+00**
f_{13}	100	3.00E+05	3.11E−03 ± 9.83E−03+	**4.71E−33 ± 7.21E−49**−	7.87E−29 ± 5.87E−29+	1.56E−32 ± 3.83E−33
f_{14}	100	3.00E+05	7.68E−03 ± 1.72E−02+	1.35E−32 ± 2.88E−48≈	7.69E−03 ± 1.38E−02+	**1.35E−32 ± 5.16E−50**
f_{15}	100	3.00E+05	2.67E+00 ± 3.73E−01+	9.20E−01 ± 5.10E−01+	8.79E−02 ± 2.78E−01+	**1.64E−14 ± 2.51E−15**
f_{16}	100	3.00E+05	9.85E+00 ± 2.50E+00+	1.80E−01 ± 4.92E−02+	9.54E+01 ± 1.58E+01+	**5.12E−03 ± 4.61E−03**
f_{17}	100	3.00E+05	9.33E−03 ± 1.50E−02≈	**1.46E−32 ± 8.30E−33**−	3.11E−03 ± 9.83E−03≈	8.10E−03 ± 5.17E−03
f_{18}	100	3.00E+05	7.51E+01 ± 1.60E+02+	1.10E−03 ± 3.47E−03+	6.58E+01 ± 2.08E+02+	**1.19E−32 ± 9.27E−33**
$w/t/l$			12/4/2	11/2/5	14/3/1	−/−/−

Table 6
Mean function error values and standard deviation of CLPSO, CMA-ES, GL-25, and LLUDE on all benchmark functions for $D = 30$.

Fun	D	MaxFEs	CLPSO	CMA-ES	GL-25	LLUDE
			Mean error ± Std Dev	Mean error ± Std Dev	Mean error ± Std Dev	Mean error ± Std Dev
f_1	30	1.50E+05	9.13E−14 ± 3.33E−14+	1.97E−29 ± 2.07E−30+	4.78E−87 ± 1.69E−86+	**0.00E+00 ± 0.00E+00**
f_2	30	1.50E+05	9.79E−15 ± 5.44E−15+	3.75E−28 ± 4.61E−29+	**2.66E−78 ± 1.25E−77**−	6.74E−50 ± 8.53E−50
f_3	30	1.50E+05	4.48E−09 ± 1.48E−09+	2.03E−14 ± 9.76E−16+	**2.98E−28 ± 1.12E−27**−	3.87E−23 ± 1.51E−23
f_4	30	1.50E+05	3.37E−16 ± 7.98E−17+	4.81E−17 ± 5.60E−17+	**0.00E+00 ± 0.00E+00**≈	**0.00E+00 ± 0.00E+00**
f_5	30	1.50E+05	9.33E−14 ± 6.40E−14+	1.50E−24 ± 2.15E−25+	**4.38E−85 ± 2.40E−84**−	2.51E−34 ± 3.38E−34
f_6	30	1.50E+05	**0.00E+00 ± 0.00E+00**≈	**0.00E+00 ± 0.00E+00**≈	**0.00E+00 ± 0.00E+00**≈	**0.00E+00 ± 0.00E+00**
f_7	30	1.50E+05	4.73E+00 ± 1.20E+00+	**2.95E−27 ± 4.82E−28**−	3.14E−02 ± 8.77E−02+	1.02E−06 ± 1.24E−06
f_8	30	1.50E+05	1.95E+01 ± 2.62E+00+	2.66E−01 ± 1.01E+00+	2.21E+01 ± 6.19E−01+	**1.05E−07 ± 1.17E−07**
f_9	30	1.50E+05	2.40E−09 ± 3.67E−09+	1.40E−03 ± 3.70E−03+	1.61E−15 ± 4.55E−15+	**0.00E+00 ± 0.00E+00**
f_{10}	30	1.50E+05	1.54E−01 ± 2.09E−02+	2.50E+02 ± 1.23E+01+	2.95E+00 ± 1.00E+00+	**3.25E−11 ± 3.15E−11**
f_{11}	30	1.50E+05	**0.00E+00 ± 0.00E+00**≈	5.19E+03 ± 6.07E+02+	4.46E+03 ± 1.36E+03+	**0.00E+00 ± 0.00E+00**
f_{12}	30	1.50E+05	**0.00E+00 ± 0.00E+00**≈	1.30E+01 ± 2.50E+00+	3.14E−05 ± 3.97E−14+	**0.00E+00 ± 0.00E+00**
f_{13}	30	1.50E+05	1.13E−17 ± 6.31E−18+	9.13E−01 ± 1.06E+00+	8.82E−31 ± 4.10E−30+	**1.57E−32 ± 0.00E+00**
f_{14}	30	1.50E+05	5.94E−17 ± 3.31E−17+	1.10E−03 ± 3.35E−03+	1.28E−30 ± 5.05E−30+	**1.35E−32 ± 0.00E+00**
f_{15}	30	1.50E+05	9.76E−08 ± 2.38E−08+	1.93E+01 ± 1.97E−01+	1.09E−13 ± 1.91E−13+	**7.55E−15 ± 0.00E+00**
f_{16}	30	1.50E+05	9.41E−07 ± 6.48E−07+	2.20E+02 ± 5.64E+01+	3.07E+01 ± 2.71E+01+	**0.00E+00 ± 0.00E+00**
f_{17}	30	1.50E+05	3.49E−16 ± 1.59E−16+	1.73E−02 ± 3.93E−02+	1.26E+02 ± 1.09E+01+	**1.57E−32 ± 0.00E+00**
f_{18}	30	1.50E+05	2.52E−14 ± 1.22E−14+	2.20E−03 ± 4.47E−03+	1.97E+03 ± 1.93E+02+	**1.35E−32 ± 0.00E+00**
$w/t/l$			15/3/0	16/1/1	13/2/3	−/−/−

Table 7
Mean function error values and standard deviation of CLPSO, CMA-ES, GL-25, and LLUDE on all benchmark functions for $D = 50$.

Fun	D	MaxFEs	CLPSO	CMA-ES	GL-25	LLUDE
			Mean error ± Std Dev	Mean error ± Std Dev	Mean error ± Std Dev	Mean error ± Std Dev
f_1	50	1.50E+05	4.82E−07 ± 1.28E−07+	4.89E−29 ± 4.03E−30+	3.57E−18 ± 7.75E−18+	**2.07E−69 ± 1.38E−69**
f_2	50	1.50E+05	1.02E−07 ± 2.82E−08+	1.32E−27 ± 8.52E−29+	3.65E−19 ± 7.73E−19+	**4.66E−32 ± 4.80E−32**
f_3	50	1.50E+05	4.29E−05 ± 5.59E−06+	**1.19E−14 ± 1.09E−15**−	1.21E−06 ± 3.66E−06+	1.62E−14 ± 9.74E−15
f_4	50	1.50E+05	6.89E−07 ± 1.20E−06+	**0.00E+00 ± 0.00E+00**≈	1.36E−13 ± 4.12E−13+	**0.00E+00 ± 0.00E+00**
f_5	50	1.50E+05	5.35E−07 ± 1.51E−07+	4.11E+01 ± 2.72E+01+	**2.60E−18 ± 9.51E−18**−	5.35E−18 ± 1.93E−18
f_6	50	1.50E+05	**0.00E+00 ± 0.00E+00**≈	**0.00E+00 ± 0.00E+00**≈	**0.00E+00 ± 0.00E+00**≈	**0.00E+00 ± 0.00E+00**
f_7	50	1.50E+05	7.87E+01 ± 1.53E+01+	**2.71E−26 ± 4.67E−26**−	7.67E+01 ± 1.45E+01+	2.71E−04 ± 3.28E−04
f_8	50	1.50E+05	1.35E+02 ± 2.73E+01+	9.25E−01 ± 1.39E+00+	4.35E+01 ± 8.35E−01+	**1.61E−06 ± 2.26E−06**
f_9	50	1.50E+05	6.47E−06 ± 2.91E−06+	**0.00E+00 ± 0.00E+00**≈	1.69E−14 ± 2.29E−14+	**0.00E+00 ± 0.00E+00**
f_{10}	50	1.50E+05	4.93E+00 ± 3.00E−01+	4.24E+02 ± 1.67E+01+	2.30E+01 ± 4.26E+00+	**8.11E−05 ± 2.23E−05**
f_{11}	50	1.50E+05	4.36E−03 ± 1.33E−06+	9.39E+03 ± 1.21E+03+	1.37E+04 ± 3.12E+03+	**0.00E+00 ± 0.00E+00**
f_{12}	50	1.50E+05	3.14E−05 ± 4.54E−10+	1.23E+01 ± 1.57E+00+	1.13E−01 ± 2.53E−01+	**0.00E+00 ± 0.00E+00**
f_{13}	50	1.50E+05	3.53E−11 ± 1.05E−11+	8.94E−01 ± 8.04E−01+	7.98E−24 ± 1.18E−23+	**5.28E−28 ± 7.35E−28**
f_{14}	50	1.50E+05	2.96E−10 ± 1.13E−10+	4.39E−03 ± 5.67E−03+	4.43E−21 ± 9.57E−21+	**4.29E−26 ± 5.79E−26**
f_{15}	50	1.50E+05	2.47E−04 ± 4.60E−05+	1.93E+01 ± 1.55E−01+	3.57E−10 ± 5.85E−10+	**5.03E−13 ± 4.75E−13**
f_{16}	50	1.50E+05	2.23E−02 ± 8.68E−03+	3.65E+02 ± 1.02E+02+	2.96E+02 ± 1.07E+02+	**3.11E−07 ± 4.35E−07**
f_{17}	50	1.50E+05	1.47E−09 ± 4.87E−10+	2.43E−02 ± 4.21E−02+	1.57E+02 ± 7.19E+00+	**8.98E−26 ± 5.12E−26**
f_{18}	50	1.50E+05	1.32E−07 ± 4.08E−08+	1.10E−03 ± 3.47E−03+	3.79E+03 ± 4.04E+02+	**4.81E−25 ± 8.93E−26**
$w/t/l$			17/1/0	13/3/2	16/1/1	−/−/−

mean and standard deviation of the function error value are recorded when the FEs reaches MaxFEs. For this comparison, MaxFEs = $10000 \times D$.

Table 8 presents the results of the mean function error value and the corresponding standard deviation obtained by the four algorithms. Functions F_3–F_5 are simple multimodal function. Functions F_6–F_8 are hybrid function. Functions F_9–F_{15} are composition function. According to the results listed in last row of the table, we can find that the proposed LLUDE is significantly superior to SaDE, JADE, and CoDE on the majority of benchmark functions. In more detail, LLUDE is significantly better than SaDE on 11 out of 15 functions, while SaDE performs significantly better than LLUDE only on one function (F_7). Although JADE significantly outperforms LLUDE on 4 functions (F_5, F_6, F_{13}, and F_{14}), LLUDE is significantly better than JADE on 7 functions. Compared to CoDE, LLUDE shows significantly better performance on 8 functions, while CoDE is significantly superior to LLUDE on 4 functions (F_3, F_5, F_7, and F_{14}).

4.6. Real-world application

In this section, in order to verify the effectiveness of the proposed LLUDE, one computationally expensive real-world application problem is utilized, namely, the protein structure prediction problem. The protein structure prediction problem plays an important role in the modern computation biology. The problem aims to determine the native conformation of proteins, namely, given an amino acid sequence, predict its 3D structure. However, the prediction involves an extremely expensive-to-evaluate energy function which has thousands of degrees of freedom, and its energy hyperspaces is highly degenerated as it has massive local minima and large regions of unfeasible conformations. This makes the problem difficult to treat. Some related works on solving this problem using evolutionary algorithms can be found in [45–47], and some recent reviews can be found in [48–50].

The goal of the prediction is finding the global minimum of the energy function as the most stable protein structure is believed to be the one corresponding to the lowest value of the potential energy. In this study, we adopt the Rosetta energy function [51]:

$$\begin{aligned}\bar{F}_1(x^1, x^2, \cdots, x^N) = {} & W_{\text{repulsion}}E_{\text{repulsion}} + W_{\text{attraction}}E_{\text{attraction}} \\ & + W_{\text{slovation}}E_{\text{slovation}} + W_{\text{bb-schb}}E_{\text{bb-schb}} \\ & + W_{\text{bb-bbhb}}E_{\text{bb-bbhb}} + W_{\text{sc-schb}}E_{\text{sc-schb}} \\ & + W_{\text{pair}}E_{\text{pair}} + W_{\text{dunbrack}}E_{\text{dunbrack}} \\ & + W_{\text{rama}}E_{\text{rama}} + W_{\text{reference}}E_{\text{reference}} \end{aligned} \quad (10)$$

Table 8
Mean function error values and standard deviation of SaDE, JADE, CoDE, and LLUDE on the CEC 2015 benchmark set.

Fun	D	MaxFEs	SaDE	JADE	CoDE	LLUDE
			Mean error ± Std Dev	Mean error ± Std Dev	Mean error ± Std Dev	Mean error ± Std Dev
F_1	30	3.00E+05	1.78E+03 ± 1.43E+03+	6.23E+00 ± 1.55E+01+	1.58E+04 ± 1.16E+04+	**5.93E−01 ± 2.47E−01**
F_2	30	3.00E+05	2.38E−11 ± 7.22E−11+	3.41E−14 ± 1.17E−14+	6.02E−13 ± 9.88E−13+	**2.84E−14 ± 2.69E−14**
F_3	30	3.00E+05	2.05E+01 ± 5.99E−02+	2.03E+01 ± 2.86E−02≈	**2.00E+01 ± 9.98E−02**−	2.03E+01 ± 2.33E−02
F_4	30	3.00E+05	3.46E+01 ± 6.44E+00+	2.61E+01 ± 3.39E+00+	2.97E+01 ± 1.08E+01+	**2.59E+01 ± 3.28E+00**
F_5	30	3.00E+05	3.08E+03 ± 4.45E+02+	**1.70E+03 ± 2.30E+02**−	1.88E+03 ± 2.17E+02−	1.97E+03 ± 1.98E+02
F_6	30	3.00E+05	3.27E+03 ± 6.55E+03+	**9.59E+02 ± 3.78E+02**−	1.93E+03 ± 1.95E+03+	1.13E+03 ± 3.80E+02
F_7	30	3.00E+05	4.97E+00 ± 2.40E+00−	7.90E+00 ± 9.56E−01+	**2.69E+00 ± 3.06E−01**−	6.32E+00 ± 2.16E+00
F_8	30	3.00E+05	6.92E+02 ± 5.06E+02+	4.14E+03 ± 1.73E+04+	1.49E+02 ± 1.22E+02+	**1.45E+02 ± 1.15E+02**
F_9	30	3.00E+05	1.03E+02 ± 1.98E−01≈	1.03E+02 ± 2.17E−01≈	1.03E+02 ± 1.58E−01≈	**1.03E+02 ± 1.22E−01**
F_{10}	30	3.00E+05	9.15E+02 ± 4.22E+02+	1.93E+04 ± 5.77E+04+	6.91E+02 ± 1.72E+02+	**6.29E+02 ± 1.60E+02**
F_{11}	30	3.00E+05	4.44E+02 ± 1.03E+02+	4.17E+02 ± 6.74E+01+	4.05E+02 ± 7.82E+01+	**3.96E+02 ± 6.31E+01**
F_{12}	30	3.00E+05	1.05E+02 ± 6.95E−01≈	1.05E+02 ± 3.83E−01≈	1.05E+02 ± 4.17E−01≈	**1.05E+02 ± 3.13E−01**
F_{13}	30	3.00E+05	1.08E+02 ± 3.40E+00+	**9.56E+01 ± 3.21E+00**−	9.80E+01 ± 8.27E+00+	9.63E+01 ± 3.48E+00
F_{14}	30	3.00E+05	3.31E+04 ± 1.31E+03+	3.24E+04 ± 9.19E+02−	**3.23E+04 ± 1.24E+03**−	3.24E+04 ± 1.23E+03
F_{15}	30	3.00E+05	**1.00E+02 ± 0.00E+00**≈	**1.00E+02 ± 0.00E+00**≈	**1.00E+02 ± 0.00E+00**≈	**1.00E+02 ± 0.00E+00**
$w/t/l$			11/3/1	7/4/4	8/3/4	−/−/−

Table 9
Results of the minimum, mean, and standard deviation of RMSD achieved by DE, Rosetta and LLUDE.

PDB ID	Length	Type	Minimum RMSD (Å)			Mean ± Std RMSD (Å)			Reduced FEs
			DE	Rosetta	LLUDE	DE	Rosetta	LLUDE	(Percentage%)
1ENH	54	α	1.89	2.08	**1.64**	2.80 ± 0.78	2.65 ± 0.39	**2.42 ± 0.37**	28.6%
2MQK	65	α	2.25	2.37	**1.97**	3.24 ± 0.96	**2.85 ± 0.35**	3.02 ± 0.29	43.8%
4ICB	76	α	3.67	3.48	**3.36**	4.22 ± 0.36	4.18 ± 0.74	**4.12 ± 0.46**	49.4%
3GWL	106	α	8.02	**5.98**	6.31	9.01 ± 0.67	7.91 ± 1.33	**7.53 ± 0.75**	19.4%
1GB1	56	α/β	6.50	6.27	**5.77**	7.52 ± 0.60	6.79 ± 0.89	**6.18 ± 0.44**	26.7%
1I6C	39	β	4.12	4.22	**4.06**	5.55 ± 0.82	5.01 ± 0.52	**4.78 ± 0.30**	29.9%

where $x^i = (x_1^i, x_2^i, x_3^i)$, $i = 1, 2, \ldots, N$, N is the number of atoms. A detailed description of the energy items and its corresponding parameter settings in Eq. (10) can be found in [51].

In order to eliminate the effect of the nonlinear equation constraints and reduce the dimension of the energy function $\bar{F}_1$, it is transformed to the dihedral-angle space at first:

$$\bar{F}_2(\tau) = \bar{F}_2(\phi_i, \psi_i, \omega_i) \tag{11}$$

where $\tau = (\tau_1, \tau_2, \ldots \tau_N) = \{\phi_i, \psi_i, \omega_i | i = 1, 2, \ldots, N_{\mathrm{RES}}\}$, N_{RES} is the number of residues, and ϕ_i, ψ_i, ω_i are the dihedral angles of the atoms C–N–C_α–C, N–C_α–C–N, and C_α–C–N–C_α for the ith residue, respectively. Then the ultrafast shape recognition [52] method is employed to extract the feature point of the protein: the molecular centroid (ctd), the closest atom to ctd (cst), the farthest atom to ctd (fct), and the farthest atom to fct (ftf). By calculating the average Euclidean distance of all atoms to these four feature atoms, $\bar{F}_2$ can be transformed into

$$\bar{F}_3(\vec{M}) = \bar{F}_3(\mu^{\mathrm{ctd}}, \mu^{\mathrm{cst}}, \mu^{\mathrm{fct}}, \mu^{\mathrm{ftf}}) \tag{12}$$

where μ^{ctd}, μ^{cst}, μ^{fct}, and μ^{ftf} are the average atomic distance to ctd, cst, fct and ftf, respectively.

Through the above processes, the energy function $\bar{F}_1$ is transformed into $\bar{F}_3$, thus, the proposed approach LLUDE can be used to solve the problem.

In this experiment, we compare LLUDE with the traditional DE [1] and Rosetta [51], where DE and LLUDE use the function $\bar{F}_3$, and Rosetta uses the function $\bar{F}_1$. Six structurally-diverse protein sequences of varying lengths listed in Table 9 are selected to evaluate these three algorithms. For each algorithm, the fragment assembly [53] is used to achieve a high prediction accuracy of the native state, and the fragment library can be downloaded from ROBETTA full-chain protein structure prediction server (http://robetta.bakerlab.org). The parameters used for DE and LLUDE are the same as described in Section 4.1, and the parameters of Rosetta are kept the same as in the original paper. For Each algorithm and each target protein, 30 independent runs are conducted with 100,000 FEs as the termination criterion. The minimum, average, and standard deviation (Std) of the root-mean-squared-deviation (RMSD) obtained in the 30 runs are recorded to evaluate the performance of each algorithm.

Table 9 presents the results of DE, Rosetta and LLUDE for the 6 test proteins. Obviously, LLUDE obtains better performance than DE and Rosetta for all target proteins except for proteins with PDB IDs 2MQK and 3GWL. For the protein with PDB ID 2MQK, Rosetta obtains the lowest value in terms of mean RMSD, but LLUDE achieves the lowest value in terms of minimum RMSD. For the protein with PDB ID 3GWL, although the minimum RMSD obtained by Rosetta is lower than that of LLUDE, the mean RMSD achieved by LLUDE is lower than that of Rosetta. In addition, for most of the target proteins, LLUDE is more stable with the relatively smaller standard deviation of RMSD than DE and Rosetta. Furthermore, the reduced FEs of LLUDE compared with that of DE and Rosetta are

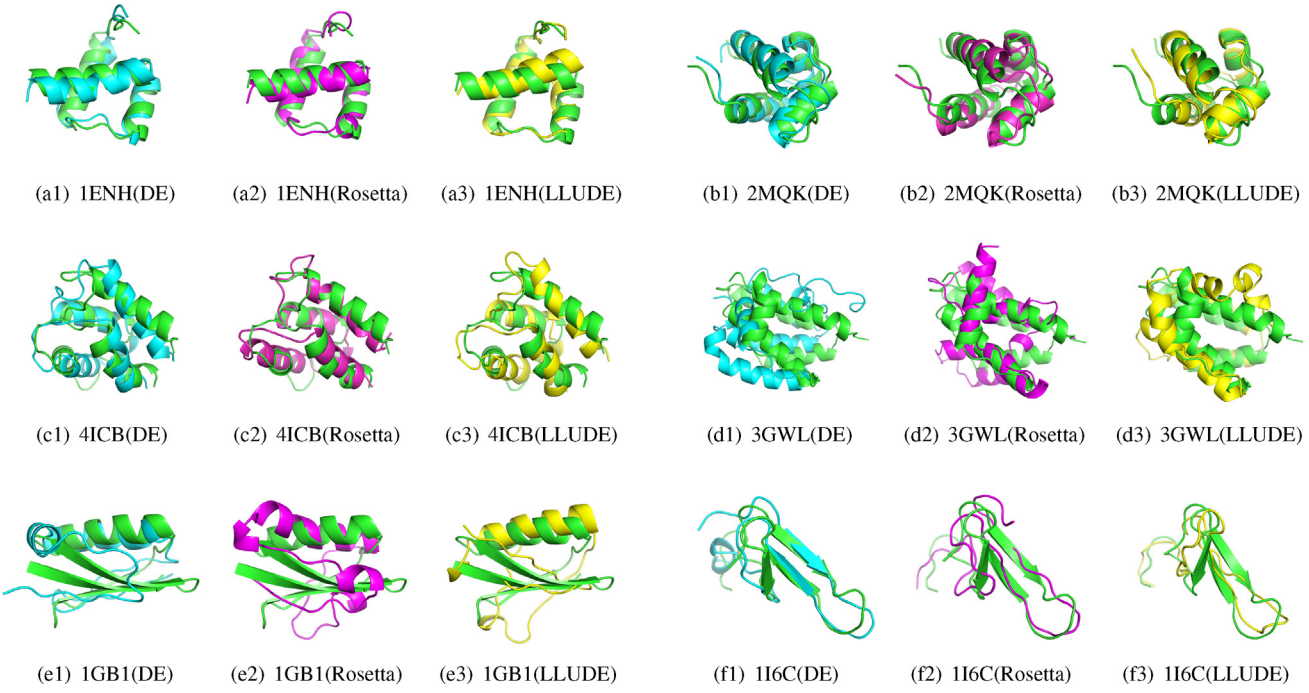


Fig. 4. Comparison of the structure obtained by DE (blue), Rosseta (pink) and LLUDE (yellow) with the native structure (green). (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

listed in the last column of Table 9. We find that LLUDE saves more than 19% of FEs compared with DE and Rosetta for each test protein. For proteins with PDB IDs 2MQK and 4ICB, 43.8% and 49.4% of FEs are saved by LLUDE, respectively.

Fig. 4 shows the structure with the lowest RMSD obtained by DE, Rosetta, and LLUDE for the 6 proteins, where the structure predicted by DE, Rosetta, and LLUDE are drawn in blue, pink, and yellow, respectively. The native structure is drawn in green. The more the predicted structure overlaps with the native structure, the higher accuracy the algorithm has. It is obvious that the structure predicted by the proposed LLUDE are closer to the native structure than DE and Rosetta for proteins with PDB IDs 1ENH, 2MQK, 4ICB, 1GB1, and 1I6C. For the protein with PDB ID 3GWL, the structure obtained by LLUDE is worse than that of Rosetta but much better than that of DE.

From the above results and analysis, we can conclude that the proposed LLUDE requires less function evaluation in obtaining more accurate structure with lower RMSD than DE and Rosetta for protein structure prediction problems.

5. Conclusion

DE is certainly a fast, robust, and simple global optimization algorithm that has been applied in various fields. However, one of the main issues in using DE is it needs a large number of function evaluations. The function evaluations usually require a large amount of computation time, especially for computationally expensive problems. In this paper, an enhanced DE variant called LLUDE has been proposed. In LLUDE, the supporting hyperplanes of some neighboring individuals of the trial individual are constructed to obtain the underestimate of the objective function. According to the underestimate, we can judge whether the trial individual is worth evaluating. Thereby significantly reducing the actual function evaluations. The underestimate value is also used to exclude some invalid regions of the search domain where the global optimum cannot be found, as well as to guide the local enhancement. Therefore, the reliability and search efficiency are enhanced.

The experimental studies are carried out on 18 well-known benchmark functions and the CEC 2015 shifted and rotated benchmark set. Experiments results show that LLUDE exhibits good performance for both unimodal and multimodal problems. The performance of LLUDE is compared with three state-of-the-art DE variants (i.e., SaDE, JADE, and CoDE) and three non-DE algorithms (i.e., CLPSO, CMA-ES, and GL-25). It can be concluded that LLUDE costs less function evaluations in obtaining better quality solutions, and it has higher success rates than the other DE variants for the majority of the benchmark functions.

We have also tested LLUDE on one well-known computationally expensive real-word application problem, namely, the protein structure prediction problem. The results of the 6 benchmark proteins show that the proposed LLUDE is an effective and efficient algorithm for computationally expensive problems. Compared with DE and Rosetta, LLUDE can obtain more accurate structures for most of the proteins whose RMSDs are relatively small.

The number n of the neighboring individuals of the trial individual should be researched in detail. Guidelines for adaptive selection of the parameter n in different stages of the algorithm will be investigated in our future research. In the future, we will also generalize our work to other EAs for other computationally expensive optimization problems.

Acknowledgements

This work was supported by the National Nature Science Foundation of China (No. 61075062, 61573317), Natural Science Foundation of Zhejiang Province (No. LY13F030008), Public Welfare Project of Science Technology Department of Zhejiang Province (No. 2014C33088), and Open Fund for the Key-Key Discipline of Zhejiang Province (No. 20151008, 20151015). The authors would like to thank the anonymous reviewers for their helpful insights, comments, and suggestions.

Appendix A. Abbreviations

DE	differential evolution
LUM	Lipschitz underestimate method
LLUDE	local Lipschitz underestimate-based differential evolution
PSO	particle swarm optimization
GA	genetic algorithm
ACO	ant colony optimization
SaDE	self-adaptive differential evolution
CoDE	composite differential evolution
JADE	adaptive differential evolution with optional external archive
CLPSO	comprehensive learning particle swarm optimizer
CMA-ES	evolution strategy with covariance matrix adaptation
GL-25	parent-centric crossover operators-based genetic algorithm
FEs	function evaluations
SR	success rate
PR	performance ratio
MaxFEs	maximum function evaluations
RMSD	root mean squared deviation

Appendix B. Procedure of LLUDE

```
Randomly initialize the population P at g = 0;
while (not satisfy stop rule) do
  for i = 1 to N_P do
    Generate trial vector x_trial according to Eqs. (1) and (2);
    if invalid regions IR ≠ ∅ then
      Check x_trial according to inequality (8);
      if not satisfy inequality (8) then
        Calculate the distances from x_trial to all x_i^g;
        Sort the distances of all x_i^g in an ascending order;
        Choose the top n individuals x_NN^t;
        Calculate l_NN^t of x_NN^t according to Eq. (6);
        Calculate ȳ_trial according to Eq. (7);
        if ȳ_trial > f(x_i^g) or ȳ_trial > f_best then
          x_i^{g+1} = x_i^g;
          Calculate d according to property (3) of L;
          if d > f_best then
            Calculate x* and add it to IR;
          end if
        else
          if f(x_trial) < f(x_i^g) then
            x_i^{g+1} = x_trial;
            Calculate x* according to property (4) of L;
            if f(x*) < f(x_trial)
              x_i^{g+1} = x*;
            end if
          end if
        end if
      end if
    end if
    Delete all l_NN;
  end for
  g = g + 1;
end while
return x_best;
```

References

[1] R. Storn, K. Price, Differential evolutionła simple and efficient heuristic for global optimization over continuous spaces, J. Glob. Opt. 11 (4) (1997) 341–359.
[2] S. Das, S.S. Mullick, P.N. Suganthan, Recent advances in differential evolution – an updated survey, Swarm Evol. Comput. 27 (6) (2016) 1–30.
[3] S. Das, P.N. Suganthan, Differential evolution: a survey of the state-of-the-art, IEEE Trans. Evol. Comput. 15 (1) (2011) 4–31.
[4] S. Sayah, A. Hamouda, A hybrid differential evolution algorithm based on particle swarm optimization for nonconvex economic dispatch problems, Appl. Soft Comput. 13 (2013) 1608–1619.

[5] N. Sharma, A. Anpalagan, Differential evolution aided adaptive resource allocation in OFDMA systems with proportional rate constraints, Appl. Soft Comput. 34 (2015) 39–50.
[6] M. Ghasemi, M. Taghizadeh, S. Ghavidel, A. Abbasian, Colonial competitive differential evolution: an experimental study for optimal economic load dispatch, Appl. Soft Comput. 40 (2016) 342–363.
[7] W.B. Langdon, R. Poli, Evolving problems to learn about particle swarm optimizers and other search algorithms, IEEE Trans. Evol. Comput. 11 (5) (2007) 561–578.
[8] M.S. Kiran, M. Gündüz, A recombination-based hybridization of particle swarm optimization and artificial bee colony algorithm for continuous optimization problems, Appl. Soft Comput. 13 (4) (2013) 2188–2203.
[9] Z. Michalewicz, Genetic Algorithms + Data Structures = Evolution Programs, Springer, London, 1996.
[10] J. Manicassamy, S.S. Kumar, M. Rangan, V. Ananth, T. Vengattaraman, P. Dhavachelvan, Gene suppressor: an added phase toward solving large scale optimization problems in genetic algorithm, Appl. Soft Comput. 35 (2015) 214–226.
[11] M. Dorigo, M. Birattari, T. Stutzle, Ant colony optimization, IEEE Comput. Intell. Mach. 1 (4) (2006) 28–39.
[12] T. Inkaya, S. Kayaligil, N.E. Özdemirel, Ant colony optimization based clustering methodology, Appl. Soft Comput. 28 (2015) 301–311.
[13] S.Y. Park, J.J. Lee, An efficient differential evolution using speeded-up *k*-nearest neighbor estimator, Soft Comput. 18 (1) (2014) 35–49.
[14] Z. Zhou, Y. Ong, P. Nair, A. Keane, K. Lum, Combining global and local surrogate models to accelerate evolutionary optimization, IEEE Trans. Syst. Man Cybern. C 37 (1) (2007) 66–76.
[15] T.W. Simpson, T.M. Mauery, J.J. Korte, F. Mistree, Kriging models for global approximation in simulation-based multidisciplinary design optimization, AIAA J. 39 (12) (2001) 2233–2241.
[16] K. Dalamagkidis, K.P. Valavanis, L. Piegl, Nonlinear model predictive control with neural network optimization for autonomous autorotation of small unmanned helicopters, IEEE Trans. Control Syst. Technol. 19 (4) (2011) 818–831.
[17] Z. Li, S. He, S. Zhang, Approximation Methods for Polynomial Optimization: Models, Algorithms, and Applications, Springer Science & Business Media, 2012.
[18] R.G. Regis, C.A. Shoemaker, Improved strategies for radial basis function methods for global optimization, J. Glob. Optim. 37 (1) (2007) 113–135.
[19] N. Deng, Y. Tian, C. Zhang, Support Vector Machines: Optimization Based Theory, Algorithms, and Extensions, CRC Press, Florida, 2012.
[20] B. Liu, Q. Zhang, G.G.E. Gielen, A Gaussian process surrogate model assisted evolutionary algorithm for medium scale expensive optimization problems, IEEE Trans. Evol. Comput. 18 (2) (2014) 180–192.
[21] Y. Jin, B. Sendhoff, Reducing fitness evaluations using clustering techniques and neural network ensembles, in: Genetic and Evolutionary Computation Conference (GECCO), Springer, Seattle, USA, 2004, pp. 688–699.
[22] T. Takahama, S. Sakai, A comparative study on kernel smoothers in differential evolution with estimated comparison method for reducing function evaluations, in: IEEE Congress on Evolutionary Computation (CEC), IEEE, Trondheim, Norway, 2009, pp. 1367–1374.
[23] R. Mallipeddi, M. Lee, An evolving surrogate model-based differential evolution algorithm, Appl. Soft Comput. 34 (2015) 770–787.
[24] S.M. Elsayed, T. Ray, R.A. Sarker, A surrogate-assisted differential evolution algorithm with dynamic parameters selection for solving expensive optimization problems, in: IEEE Congress on Evolutionary Computation (CEC), IEEE, Beijing, China, 2014, pp. 1062–1068.
[25] B. Yuan, B. Li, T. Weise, X. Yao, A new memetic algorithm with fitness approximation for the defect-tolerant logic mapping in crossbar-based nanoarchitectures, IEEE Trans. Evol. Comput. 18 (6) (2014) 846–859.
[26] Y. Liu, F. Sun, A fast differential evolution algorithm using *k*-nearest neighbour predictor, Expert Syst. Appl. 38 (4) (2011) 4254–4258.
[27] H.A. Pham, Reduction of function evaluation in differential evolution using nearest neighbor comparison, Vietnam J. Comput. Sci. 2 (2) (2015) 121–131.
[28] I.C. Paschalidis, Y. Shen, P. Vakili, S. Vajda, SDU: a semidefinite programming-based underestimation method for stochastic global optimization in protein docking, IEEE Trans. Autom. Control 52 (4) (2007) 664–676.
[29] J. M”uller, R. Piché, Mixture surrogate models based on Dempster–Shafer theory for global optimization problems, J. Global Optim. 51 (1) (2011) 79–104.
[30] G. Beliakov, K.F. Lim, Challenges of continuous global optimization in molecular structure prediction, Eur. J. Oper. Res 181 (3) (2007) 1198–1213.
[31] G. Beliakov, Extended cutting angle method of global optimization, Pac. J. Optim. 4 (1) (2008) 152–176.
[32] G. Beliakov, A. Ferrer, Bounded lower subdifferentiability optimization techniques: applications, J. Glob. Opt. 47 (2) (2010) 211–231.
[33] A. Auslender, A. Ferrer, M.A. Goberna, M.A. López, Comparative study of RPSALG algorithm for convex semi-infinite programming, Comput. Optim. Appl. 60 (1) (2015) 59–87.
[34] A. Ferrer, A. Bagirov, G. Beliakov, Solving DC programs using the cutting angle method, J. Glob. Opt. 61 (1) (2014) 71–89.
[35] H. Wang, S. Rahnamayan, H. Sun, M.G.H. Omran, Gaussian bare-bones differential evolution, IEEE Trans. Cybern. 43 (2) (2013) 634–647.
[36] S. Das, A. Konar, U.K. Chakraborty, A. Abraham, Differential evolution using a neighborhood based mutation operator, IEEE Trans. Evol. Comput. 13 (3) (2009) 526–553.
[37] A.K. Qin, V.L. Huang, P.N. Suganthan, Differential evolution algorithm with strategy adaptation for global numerical optimization, IEEE Trans. Evol. Comput. 13 (2) (2009) 398–417.
[38] J.Q. Zhang, A.C. Sanderson, JADE: adaptive differential evolution with optional external archive, IEEE Trans. Evol. Comput. 13 (5) (2009) 945–958.
[39] Y. Wang, Z. Cai, Q. Zhang, Differential evolution with composite trial vector generation strategies and control parameters, IEEE Trans. Evol. Comput. 15 (1) (2011) 55–66.
[40] H.J.C. Barbosa, H.S. Bernardino, A.M.S. Barreto, Using performance profiles to analyze the results of the 2006 CEC constrained optimization competition, in: IEEE Congress on Evolutionary Computation (CEC), IEEE, Barcelona, Spain, 2010, pp. 1–8.
[41] J.J. Liang, A.K. Qin, P.N. Suganthan, S. Baskar, Comprehensive learning particle swarm optimizer for global optimization of multimodal functions, IEEE Trans. Evol. Comput. 10 (3) (2006) 281–295.
[42] N. Hansen, A. Ostermeier, Completely derandomized selfadaptation in evolution strategies, Evol. Comput. 9 (2) (2001) 159–195.
[43] C. Garcia-Martinez, M. Lozano, F. Herrera, D. Molina, A.M. Sanchez, Global and local real-coded genetic algorithms based on parent-centric crossover operators, Eur. J. Oper. Res. 185 (3) (2008) 1088–1113.
[44] J.J. Liang, B.Y. Qu, P.N. Suganthan, Q. Chen, Problem definitions and evaluation criteria for the CEC 2015 competition on learning-based real-parameter single objective optimization, Zhengzhou University/Nanyang Technological University, Zhengzhou, China/Singapore, 2014, Technical Report.
[45] S. Sudha, S. Baskar, S.M.J. Amali, S. Krishnaswamy, Protein structure prediction using diversity controlled self-adaptive differential evolution with local search, Soft Comput. 19 (16) (2015) 1635–1646.
[46] F.L. Custódio, H.J.C. Barbosa, L.E. Dardenne, Full-atom ab initio protein structure prediction with a genetic algorithm using a similarity-based surrogate model, in: IEEE Congress on Evolutionary Computation (CEC), IEEE, Shanghai, China, 2010, pp. 1–8.
[47] F.L. Custódio, H.J.C. Barbosa, L.E. Dardenne, A multiple minima genetic algorithm for protein structure prediction, App. Soft Comput. 15 (2014) 88–99.
[48] C.A. Floudas, H. Fung, S. McAllister, M. Monnigmann, R. Rajgaria, Advances in protein structure prediction and de novo protein design, Chem. Eng. Sci. 61 (3) (2006) 966–988.
[49] K.A. Dill, J.L. MacCallum, The protein-folding problem, 50 years on, Science 338 (2012) 1042–1046.
[50] R.A. Friesner, R. Abel, D.A. Goldfeld, E.B. Miller, C.S. Murrett, Computational methods for high resolution prediction and refinement of protein structures, Curr. Opin. Struct. Biol. 23 (2) (2013) 177–184.
[51] C.A. Rohl, C.E. Strauss, K.M. Misura, D. Baker, Protein structure prediction using Rosetta, Method Enzymol. 383 (2004).
[52] P.J. Ballester, W.G. Richards, Ultrafast shape recognition for similarity search in molecular databases, Proc. Math. Phys. Eng. Sci. 463 (2081) (2007) 1307–1321.
[53] J. Handl, J. Knowles, R. Vernon, D. Baker, S.C. Lovell, The dual role of fragments in fragment-assembly methods for de novo protein structure prediction, Proteins 80 (2) (2012) 490–504.