Applied Soft Computing 36 (2015) 300–314

Contents lists available at ScienceDirect

Applied Soft Computing

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

Cooperative differential evolution with fast variable interdependence learning and cross-cluster mutation

Hongwei Ge[a,b], Liang Sun[a,*], Xin Yang[a], Shinichi Yoshida[c], Yanchun Liang[d]

[a] College of Computer Science and Technology, Dalian University of Technology, Dalian 116023, China
[b] Key Laboratory of Advanced Control and Optimization for Chemical Processes, East China University of Science and Technology, Shanghai 200237, China
[c] School of Information, Kochi University of Technology, Kochi 7828502, Japan
[d] College of Computer Science and Technology, Jilin University, Changchun 130012, China

ARTICLE INFO

Article history:
Received 6 June 2014
Received in revised form 20 June 2015
Accepted 20 July 2015
Available online 1 August 2015

Keywords:
Cooperative optimization
Differential evolution
Large scale optimization
Cross-cluster mutation

ABSTRACT

Cooperative optimization algorithms have been applied with success to solve many optimization problems. However, many of them often lose their effectiveness and advantages when solving large scale and complex problems, e.g., those with interacted variables. A key issue involved in cooperative optimization is the task of problem decomposition. In this paper, a fast search operator is proposed to capture the interdependencies among variables. Problem decomposition is performed based on the obtained interdependencies. Another key issue involved is the optimization of the subproblems. A cross-cluster mutation strategy is proposed to further enhance exploitation and exploration. More specifically, each operator is identified as exploitation-biased or exploration-biased. The population is divided into several clusters. For the individuals within each cluster, the exploitation-biased operators are applied. For the individuals among different clusters, the exploration-biased operators are applied. The proposed operators are incorporated into the original differential evolution algorithm. The experiments were carried out on CEC2008, CEC2010, and CEC2013 benchmarks. For comparison, six algorithms that yield top ranked results in CEC competition are selected. The comparison results demonstrated that the proposed algorithm is robust and comprehensive for large scale optimization problems.

1. Introduction

Optimization problems are rife in diverse fields such as mechanical engineering, compressed sensing, natural language processing, structure control, and bio-computing [1–5]. Researchers have to determine a set of model parameters or state-variables that provide the minimum or maximum value of a predefined cost or objective [6]. With the coming of internet of things (IoT) [7], many optimization problems are becoming more difficult, i.e., the problems are characterized by more variables with complicated interactions. Research on optimization problems has attracted the attention of researchers and many algorithms have been proposed. Though the existing optimizers have shown to be successful in solving moderate scale problems, many of them still suffer from the "curse of dimensionality", which means that their performance will deteriorate as the dimensionality of the problem increases [8,9]. Thus effective and efficient algorithms for large scale optimization have become essential requirements. In this paper, we aim at solving the large scale optimization problems and providing tools for scientists and engineers when they are solving real world problems from the involved disciplines.

Generally, the natural way to address the "curse of dimensionality" is to apply cooperative optimization, which can be regarded as an automatic approach to implement the divide-and-conquer strategy. A typical cooperative optimization algorithm can be summarized as follows [10]:

1. Problem decomposition: decompose a large scale problem into smaller scale subproblems.
2. Subproblem optimization: optimize each subproblem by means of a separate optimizer.
3. Cooperative coordination: combine the subsolutions to obtain an entire solution.

A key issue with regards to the cooperative optimization is the task of problem decomposition. An appropriate decomposition algorithm should group interacted variables together so that the interdependencies among different subproblems are minimized. Based on whether the variable interdependencies are considered or not, the decomposition algorithms can be classified into two

* Corresponding author. Tel.: +86 15998564404.
E-mail address: liangsun@dlut.edu.cn (L. Sun).

http://dx.doi.org/10.1016/j.asoc.2015.07.016
1568-4946/

categories. Generally, the algorithms performed without considering variable interdependencies are simple and effective for separable problems, but have difficulty in solving nonseparable problems [10–16]. On the other hand, the algorithms performed by considering the variable interdependencies provide opportunities to solve large scale nonseparable problems [17–24]. However, many of them either add extra computational burden to the algorithm or lack extensive variable interdependence learning.

Another key issue with regards to the cooperative optimization is the optimization of the subproblems. The widely used optimizers are inspired by nature phenomena, which include genetic algorithm (GA) [25], evolution programming (EP) [26,27], evolution strategy (ES) [28,29], differential evolution (DE) [6,30], ant colony optimization (ACO) [31], particle swarm optimization (PSO) [32–37], bacterial foraging optimization (BFO) [38], simulated annealing (SA) [39], tabu search (TS) [40], harmony search (HS) [35,36,40], etc. These optimizers facilitated research into the optimization of the subproblems. However, many of them are still not free from premature convergence for the complex multi-modal, rugged, and nonseparable problems.

As can be seen from the aforementioned, as far as cooperative optimization algorithms concerned, there still exists a big room to improve their performance through deeper studies. In this paper, we propose a variant of cooperative optimization algorithm. The study concentrates on the aforementioned two issues. To solve the task of problem decomposition, we propose a fast variable interdependence searching operator, which operates by recursively partitioning decision variables into blocks and identifying the interdependences among different blocks. Then we decompose a large scale problem into small scale subproblems based on the obtained interdependencies. To solve the task of subproblem optimization, we propose a cross-cluster mutation strategy, in which each operator is identified as exploration-biased or exploitation-biased. The population is divided into several clusters. For the individuals among different clusters, the exploration-biased operators are applied, and for the individuals within each cluster, the exploitation-biased operators are applied. By favoring search in the vicinity of each cluster and in the regions among different clusters, this strategy promotes efficient exploration as well as efficient exploitation. We further incorporate the proposed strategies into the original differential algorithm to perform optimization. The reason that we adopt differential evolution as base optimizer is that it has been frequently adopted and the resulting variants have been achieving top ranks in various competitions [30].

The reminder of this paper is organized as follows. Section 2 reviews the related works with regard to cooperative optimization and differential evolution. Section 3 gives the description of the proposed algorithm, which includes the fast variable interdependence learning method, and the cross-cluster mutation strategy. Section 4 presents the experimental results, followed by concluding remarks in Section 5.

2. Related works

This work is closely related to cooperative optimization and differential evolution. In this section, we firstly review the prevailing algorithms for cooperative optimization. And then review the state of the art algorithms for differential evolution.

2.1. Cooperative optimization

The cooperative optimization algorithm addresses the “curse of dimensionality” by applying the divide-and-conquer strategy. One of the most important issue with regards to divide-and-conquer is the task of problem decomposition, the process of partitioning a large scale problem into subproblems. The decomposition decision regarding variable interdependencies plays a significant role in the algorithm’s performance. Based on whether the variable interdependencies are considered during the execution process, the existing algorithms can be classified into two categories. The algorithms belonging to the first category are performed without considering the variable interdependencies. For example, the algorithms reported in [10–15] operate by decomposing problems arbitrarily, with each subproblem being optimized by a separate optimizer. When a subproblem is being optimized, the variables belonging to other subproblems are kept unchanged. The solution to the problem is obtained by combining the subsolutions found by each of the separate optimizer. During the optimization process, the decomposition strategy is kept unchanged. The advanced algorithms belonging to this category operates by decomposing problem dynamically [8,16]. During the optimization process, the algorithms decompose problem based on a measured stagnation, and eliminate previous decomposition strategies which are no longer making contributions. In this way, the algorithm has the opportunity to optimize interacting variables. Generally, algorithms that do not consider variable interdependencies are effective for separable problems, but have difficulty solving nonseparable problems, especially the problems with tightly interacted variables.

The algorithms belonging to the second category are performed by considering the variable interdependencies implicitly or explicitly. For example, the algorithms reported in [17–19] use a random grouping scheme to decompose a large scale problem into subproblems, and then optimize each of the subproblem by a separate optimizer. For co-adaptation, the algorithms apply weights to the subproblems, and optimize the weight vectors with a certain optimizer. In [20–22], the variable interdependencies are captured during the optimization process. Initially, the system does not have any knowledge with regards to the variable interdependencies. With the optimization process going on, the system captures the variable interdependencies by investigating the relationship between the objective functions and the candidate solutions. While optimizing, the decomposition and the optimization strategies are adapted according to the obtained interdependencies. In our previous work [24], we propose a statistical model to explore the interdependencies among variables. At this point, the degree of interdependence between each pair of variables is quantified. With these interdependencies, the problem is decomposed and optimized using a cooperative particle swarm optimization (CPSO) framework. Generally, the algorithms that consider variable interdependencies provide opportunities to solve nonseparable problems. However, many of them require extensive computational burden to capture the variable interdependencies. For example, in our previous work [24], the computational efforts incurred by the variable interdependence learning take about 30% of the total computational efforts.

A critical issue in the application of cooperative optimization is to determine the learning strategy allowing efficient exploration of the interdependencies among variables, as well as avoiding intensive computational efforts consumption. Motivated by these findings, we developed a fast variable interdependence search operator to mitigate the problems encountered in the aforementioned algorithms. The proposed operator works as follows. The subsets of variables are referred as variable blocks. The interdependencies among variable blocks are further defined. We advocate a fast search operator which operates by recursively partitioning decision variables into blocks and then identify the interdependencies among the blocks. By favoring search the interdependencies among variable blocks recursively, this operator promotes efficient exploration of the interdependencies, without substantially adding computational burden.

2.2. Differential evolution

A typical minimal optimization problem can be described as:

$$\begin{aligned} &\text{Find}: && \vec{x}^* \in S \\ &\text{Such that}: && \forall \vec{x} \in S, \quad f(\vec{x}^*) \le f(\vec{x}), \end{aligned} \tag{1}$$

where $S=[b_{1l},\ b_{1u}]\times[b_{2l},\ b_{2u}]\times\cdots\times[b_{nl},\ b_{nu}]\subseteq R^n$ is the solution space, $\vec{x}\in S$ is the solution vector, and $f: S\to R$ is the objective function. Since $\max\{f(\vec{x})\} = -\min\{-f(\vec{x})\}$, the restriction to minimization is without loss of generality. The differential evolution algorithm has emerged as a competitive form of evolutionary computing technique for optimization problems. Since its first introduction by Storn and Price [42–44], it has attracted a high level of interest and has shown superior performance on benchmark functions as well as real-world problems [3,5,14,15,17,18,6,30,45–49]. Some examples of empirical studies on differential evolution are reported in [42,46,47]. From these studies, the main effort has been concentrated on understanding and tuning of the control parameters such as the mutation scale factor *F*, the crossover rate *CR*, and the population size *NP*. These studies provide an empirical way to improve performance.

As can be perceived from the empirical studies, the control parameters involved are problem dependent [46,48]. It is time-consuming to tune parameters manually for different problems. Therefore, researchers have started to investigate self-adaptive techniques to tune parameters automatically during the evolution process [3,5,15,17,49]. For example, in [3,15], the chromosome in the population is extended with control parameter *F* and *CR* encoded. The better values of the encoded control parameters lead to better individuals which are more likely to produce better parameter values in the next generation. In [5,15,49], both trial vector generation strategies and their associated control parameter values can be gradually self-adapted according to their previous records of generating promising solutions.

Mutation and crossover are important operators in a differential evolution algorithm. The mutation generates new individuals by adding random changes to existing individuals. The objective is to keep the diversity of the population, and thus avoid local optima. The crossover generates new individuals by combining two or more existing individuals. The objective is to guide the search moving toward the promising areas of the solution space. Appropriate design of mutation and crossover operators can help to increase the efficiency of optimization, so there has been a surge of interest in developing new mutation and crossover operators for the basic DE algorithm and a wealth of DE-variants have been proposed. Examples of the prominent and competitive algorithms are DE with trigonometric mutation [50], DE with neighborhood based mutation [51], DE using arithmetic recombination [52,53], and DE with adaptive selection of mutation strategies [54].

The main objective of many aforementioned algorithms is to balance the exploration and exploitation dilemma. Exploration enables the algorithm to identify the regions of search space where optimum solutions are located, while exploitation accelerates the algorithm converge to the optimum solution. Even though the existing algorithms facilitated research into global numerical optimization, many of them are still not free from insufficient exploration or insufficient exploitation. In fact, insufficient exploration and insufficient exploitation will not only cause the algorithm loses promising solutions, but also cause the algorithm fails to converge. Generally, the operators of DE algorithm can exhibit different properties during the execution process. Some operators favor exploration, while some operators favor exploitation. For a given optimization problem, inappropriate choice of the operators may make the algorithm be excessive exploitative or converge to a pseudo-optimum. Motivated by these findings, we propose a DE variant which integrates the exploration-biased operator and the exploitation-biased operator in a unified model. The proposed algorithm adopts a multi-cluster strategy. For the individuals among different clusters, the exploration-biased operators are applied, and for the individuals within each cluster, the exploitation-biased operators are applied. The algorithm is designed in such a way so that the exploration-biased operators favor search in the regions among different clusters, and the exploitation-biased operators favor search in the vicinity of each cluster.

3. Problem decomposition

In this section, we aim at solving the task of problem decomposition. The details with regards to the variable interdependence, the fast variable interdependence learning algorithm, and the problem decomposition method are provided.

3.1. Variable interdependence

One important issue for the optimization problem is the variable separability. The definition of separable and nonseparable variables can be described as follows.

Definition 1. Given an optimization problem $f(\vec{x})$ with variable set $S=\{x_1, x_2, \ldots, x_n\}$, variable x_i is said to be separable from variable x_j, if for all the context vectors

$$\forall \vec{c} = (\ldots, x_{i-1}, -, x_{i+1}, \ldots, x_{j-1}, -, x_{j+1}\ldots),$$

the following constraint is satisfied:

$$\forall x_i, x_i' \in S, \quad \text{and} \quad \forall x_j \in S,$$

$$\text{if } f(\vec{\alpha}) \le f(\vec{\beta}), \quad \text{then} \quad \forall x_j' \in S, f(\vec{\alpha}') \le f(\vec{\beta}'),$$

where

$$\vec{\alpha} = (\ldots, x_i, \ldots, x_j, \ldots), \quad \vec{\beta} = (\ldots, x_i', \ldots, x_j, \ldots),$$
$$\vec{\alpha}' = (\ldots, x_i, \ldots, x_j', \ldots), \quad \vec{\beta}' = (\ldots, x_i', \ldots, x_j', \ldots),$$
$$x_i \ne x_i', x_j \ne x_j';$$

if the above constraint is not satisfied, variable x_i is said to be nonseparable from variable x_j.

Variable x_i is separable from variable x_j means that x_i is independent of x_j. Variable x_i is nonseparable from variable x_j means that there exists interdependence between x_i and x_j. For many real world problems, interdependencies often occur among subsets. The set of variables in a subproblem can be termed as variable block. We define the interdependence between each pair of variable blocks as follows.

Definition 2. Given an optimization problem $f(\vec{x})$ with variable set $S=\{x_1, x_2, \ldots, x_n\}$, and B_1 and B_2 are two subset of S which satisfy $B_1\cap B_2=\emptyset$, B_1 is said to be separable from B_2 if for all the context vectors

$$\forall \vec{c} = (\ldots, x_{i-1}, -, x_{i+1}, \ldots, x_{j-1}, -, x_{j+1}, \ldots),$$

the following constraint is satisfied:

$$\forall x_{i_1}, x_{i_2}, \ldots, x_{i_m}, x_{i_1}', x_{i_2}', \ldots, x_{i_m}' \in B_1, \quad \text{and} \quad \forall x_{j_1}, x_{j_2}, \ldots, x_{j_k} \in B_2,$$

$$\text{if } f(\vec{\alpha}) \le f(\vec{\beta}), \quad \text{then} \quad \forall x_{j_1}', x_{j_2}', \ldots, x_{j_k}' \in B_2, \quad f(\vec{\alpha}') < f(\vec{\beta}'),$$

where

$$\vec{\alpha} = (\ldots, x_{i_1}, \ldots, x_{i_m}, \ldots, x_{j_1}, \ldots, x_{j_k}, \ldots),$$

$$\vec{\beta} = (\ldots, x'_{i_1} \ldots, x'_{i_m}, \ldots, x_{j_1}, \ldots, x_{j_k}, \ldots),$$

$$\vec{\alpha}' = (\ldots, x_{i_1}, \ldots, x_{i_m}, \ldots, x'_{j_1}, \ldots, x'_{j_k}, \ldots),$$

$$\vec{\beta}' = (\ldots, x'_{i_1}, \ldots, x'_{i_m}, \ldots, x'_{j_1}, \ldots, x'_{j_k}, \ldots),$$

$$(x_{i_1} - x'_{i_1})^2 + (x_{i_2} - x'_{i_2})^2 + \cdots + (x_{i_m} - x'_{i_m})^2 \neq 0,$$

$$(x_{j_1} - x'_{j_1})^2 + (x_{j_2} - x'_{j_2})^2 + \cdots + (x_{j_k} - x'_{j_k})^2 \neq 0;$$

if the above constraint is not satisfied, variable block B_1 is said to be nonseparable from variable block B_2.

B_1 is separable from B_2 means that the variables belonging to B_1 are independent of the variables belonging to B_2. B_1 is nonseparable from B_2 means that some variables belonging to B_1 are interacted with some variables belonging to B_2.

3.2. Fast interdependence searching

The ideal decomposition method is to partition the variable set into blocks so that the variables within each block are nonseparable, and the variables among different blocks are separable. At this point, the issue lies in identifying the interdependence among the variables. However, for a high dimensional problem, it is computationally intensive to identify the interdependence between each pair of variables. To identify the variables that are interacted with a given variable x_i, the referred variables can be grouped into blocks. For a given block B, if it is separable from x_i, then all the variables in B are separable from x_i. If block B is interacted with x_i, then at least one variable in B is interacted with x_i. This phenomenon indicates that it will save computational efforts if the variables that are separable from x_i are omitted. On the other hand, if a block is interacted with x_i, the block can be further partitioned into smaller blocks and the interacted variables can be identified in the smaller blocks. Motivated by these findings, a fast interdependence searching algorithm is proposed, which works as follows:

- Given an optimization problem $f(\vec{x})$ with variable set $S = \{x_1, x_2, \ldots, x_n\}$, for a variable set $P \subseteq S$ and a variable $x_i \in S - P$, if P contains one variable x_j, then Definition 1 can be used to identify whether x_j is interacted with x_i. If P contains more than one variables, the relationship can be further identified between block $\{x_i\}$ and block P.
- For $\{x_i\}$ and P, if the constraint in Definition 2 is satisfied, then x_i is separable from P, i.e., $\forall x_j \in P$, x_i and x_j are separable from each other.
- For $\{x_i\}$ and P, if the constraint in Definition 2 is violated, i.e., there exist some variables in P, which are interacted with x_i. To identify the interacted variables more specifically, P can be partitioned into subsets and then the interacted variables can be identified in the subsets.

Fig. 1 shows the pseudo code for the fast interdependence searching algorithm. In the algorithm, the interdependence between $\{x_i\}$ and P is tested for at most N times (line 2 to line 10). The reason that we set the upper limit is to avoid infinite loop when $\{x_i\}$ and P are separable. For $\{x_i\}$ and P, suppose q is the probability of constraints deviated by Definition 2, then the probability for the algorithm identifying the interdependence in the kth loops is $1-(1-q)^k$. The analysis indicates that the probability of the algorithm failing to identify the interdependence decreases exponentially as the number of loops increases. After testing the relationship between $\{x_i\}$ and P. The algorithm executes the following steps. If $\{x_i\}$ is separable from P, then all the variables in

```
1.  Function: Interacted_Set = Fast_Search(x_i, P)
    //Search the variables that are interacted with x_i in P.
2.  separable=true;
3.  cnt=0;
4.  while separable==true and cnt<N
5.      Create candidate solution vectors α⃗, β⃗, α⃗′ and β⃗′ randomly
        based on x_i, P;
6.      if (f(α⃗) − f(β⃗)) · (f(α⃗′) − f(β⃗′)) < 0
7.          separable=false;
8.      endif
9.      cnt=cnt+1;
10. endwhile
11. if separable==true
12.     return Interacted_Set=emptyset;
13. elseif P contains one variable x_j
14.     return Interacted_Set={x_j};
15. else
16.     Divide P into two subsets P_1 and P_2;
17.     return
            Interacted_Set=Fast_Search(x_i, P_1)∪Fast_Search(x_i,P_2);
18. endif
```

Fig. 1. Pseudo code for the fast interdependence searching algorithm.

P are separable from x_i. If $\{x_i\}$ is interacted with P, and P contains one variable x_j, then x_j is interacted with x_i (line 12, 13). If $\{x_i\}$ is interacted with P, and P contains more than one variables, then the algorithm partitions P into two subsets P_1 and P_2, and then identifies the interacted variables in P_1 and P_2, recursively. The variables interacted with x_i is obtained by merging the interacted variables in P_1 and P_2.

The number of function evaluations (FEs) is used to evaluate the computational efforts of the proposed algorithm. For an n dimensional problem $f(\vec{x})$, the FEs involved in the interdependence searching are as follows:

1. If $f(\vec{x})$ is completely separable, i.e., all of its variables are separable, the number of FEs is $4n \times N$.
2. If $f(\vec{x})$ is completely nonseparable, i.e., all of its variables are interacted, the number of FEs varies from $4n \log n$ to $4n \log n \times N$.
3. If $f(\vec{x})$ is partially nonseparable, i.e. some of its variables are interacted, the number of FEs varies from $4n \times N$ to $4n \log n \times N$.

As can be seen from the above items, in the best case, the computational effort involved is $4n \times N$, which increases linearly as the dimension of the problem increases. On the other hand, in the worst case, the computational effort involved is $4n \log n \times N$, which increases under order $n \log n$ as the dimension of the problem increase. This observation shows that the proposed algorithm requires relatively low computational efforts to capture the interdependence among variables. In other words, this strategy is effective in capturing variable interdependencies with little domain knowledge.

3.3. Decomposition

The next issue lies with decomposing large scale problems into small scale subproblems. With the variable interdependence, we want to obtain a partition of variable set S, which satisfies that $S_1 \cup S_2 \cup \cdots \cup S_k = S$ and $S_i \cap S_j = \emptyset (i \neq j)$. For each pair of variables x_i and x_j, if x_i and x_j are separable from each other, then they are allocated to different subsets, else, they are allocated to the same subset. In order to obtain this partition, the following method is used, which works as follows. Firstly, a variable set S' is created, initially, $S' = \{x_1, x_2, \ldots, x_n\}$. Secondly, an variable x_i is selected from S' randomly, and then an empty set S_i is created. For S_i, all the variables

that are interacted with x_i are selected and recruited. Meanwhile, the selected variables are removed from S'. This process is repeated until all the variables in S' are exhausted.

4. Differential evolution with cross-cluster mutation

In this section, we aim at solving the task of the optimization of subproblems. The exploration and exploitation tendencies of DE mutation operators are studied, followed by the cross-cluster mutation strategies.

4.1. Exploration and exploitation tendencies of DE mutation operators

The exploration and exploitation are important criteria for evaluating the performance of a DE algorithm. Exploration promotes the search in wider regions of the search space, thus increases the probability of locating the global optimum solution, while exploitation promotes the search in the vicinity of a local optimum solution, thus accelerates convergence. The mutation operators play a crucial role in a DE algorithm. Different mutation operators exhibit different exploration and exploitation tendencies. For example, the operators that incorporate randomly selected individuals or randomly selected parameters (e.g., DE/rand/1, DE/rand/2) often exhibit exploration tendency, since the random introduced elements add variability to the mutant individual. On the other hand, the operators that incorporate best individuals (e.g., DE/best/1, DE/current-to-best/1, DE/best/2, DE/current-to-best/2) often exhibit exploitation tendency, since the best individual usually attracts the mutant individual to gather around its current position.

In order to determine appropriate operators that guide the search moving towards and converges to the global optimum, the first issue lies in quantifying the exploration and exploitation abilities of the operators. Generally, different operators gather individuals into clusters with different speed. A faster speed indicates a stronger exploitation ability, and a slower speed indicates a stronger exploration ability. Thus the clustering tendency of the population is used to evaluate the exploration and exploitation abilities of the operators. The Hopkins test value (h-value) is used to compare the distance between a set of vectors which are scattered in the search space and a set of vectors which are randomly selected in the given data set. More specifically, suppose there is a search space $S \in D^n$, and a data set $X = \{x_1, x_2, \ldots, x_{NP}\} \subset S$. Let $Y = \{y_1, y_2, \ldots, y_M\}$ be vectors which are uniformly scattered in S, and let $X_1 = \{x'_1, x'_2, \ldots, x'_M\} \subset X$ be vectors which are randomly selected from X. The lth power of h-value of X can be defined as:

$$h = \frac{\sum_{j=1}^{M} d_j^l}{\sum_{j=1}^{M} d_j^l + \sum_{j=1}^{M} \delta_j^l}, \tag{2}$$

where d_j is the distance of $y_i \in Y$ to its closest vector in X_1, and δ_j is the distance of $x'_i \in X_1$ to its closest vector in $X_1 - \{x'_i\}$. As can be perceived from Eq. (2), when data set X contains clusters, the sum $\sum_{j=1}^{m} \delta_j^l$ is expected to be smaller compared with $\sum_{j=1}^{m} d_j^l$, thus h-value is expected to be larger. On the other hand, when data set X does not contain clusters, h-value is expected to be smaller.

To illustrate the exploration and exploitation abilities of different mutation operators, the experiment on the 2-D Shekel's Foxholes benchmark function is conducted as an example. Fig. 2 shows the two dimensional surface landscapes of this function. The function has a global optimum at $f_{min} = f(-32, 32) = 0.998004$, and the search space is $S = [-65.536, 65.536]^2$. The differential evolution framework is adopted. For the experiments, the mutation operators DE/best/1, DE/rand/1, DE/current-to-best/1, DE/best/2, DE/rand/2, and DE/current-to-best/2 are utilized. Due to the random nature of the h-value, the experiment is repeated for 25 times, and the average results are reported. Fig. 3 shows the average h-value of the six mutation operators, where the X-axis is the generation number, and the Y-axis is the h-value. As can be seen from Fig. 3, all the operators generate large h-values within 100 generations. However, different operators gather the individuals into clusters with different speed. DE/best/1 exhibits the fastest speed, which indicates its strong exploitative ability. On the other hand, DE/rand/2 exhibits the slowest speed, which indicates its strong explorative ability.

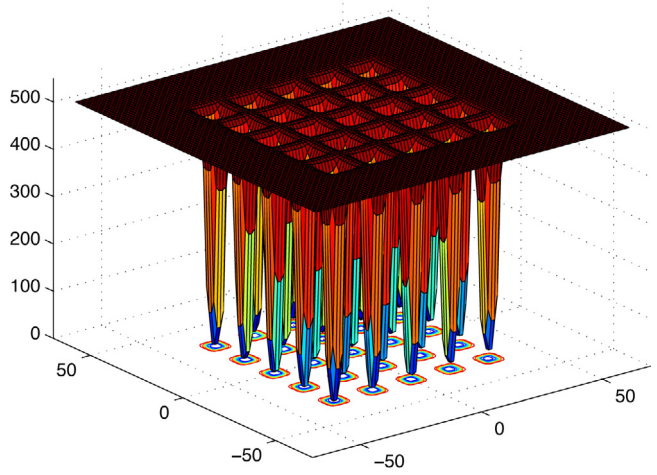


Fig. 2. Surface landscapes of the Shekel's Foxholes benchmark function in the search space S.

4.2. Cross-cluster mutation

As can be seen from the previous section, different mutation operators exhibit different exploration and exploitation abilities. Generally, a high performance DE algorithm requires effective exploration as well as its effective exploitation. Lets observe the details of the six mutation operators. The DE/best/1, DE/best/2, DE/current-to-best/1, and DE/current-to-best/2 are exploitative biased, since x_g^{best} is involved in the equation and it tends to attract the individuals around its current position. The DE/rand/1 and DE/rand/2 are explorative biased, since all the components are

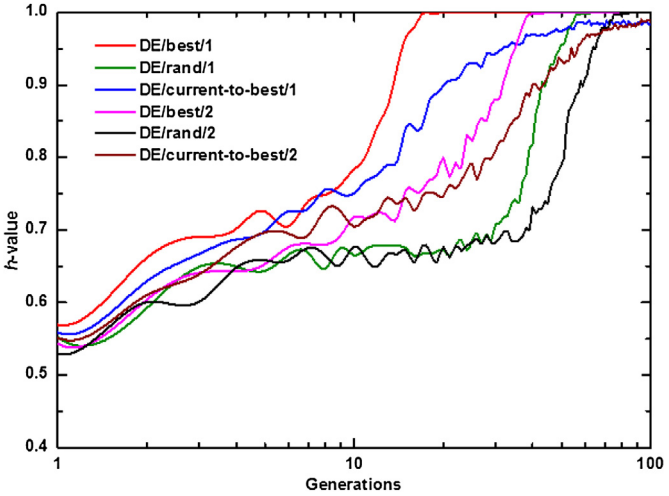


Fig. 3. Average h-value of the six classic DE mutation operators on the Shekel's Foxholes benchmark function.

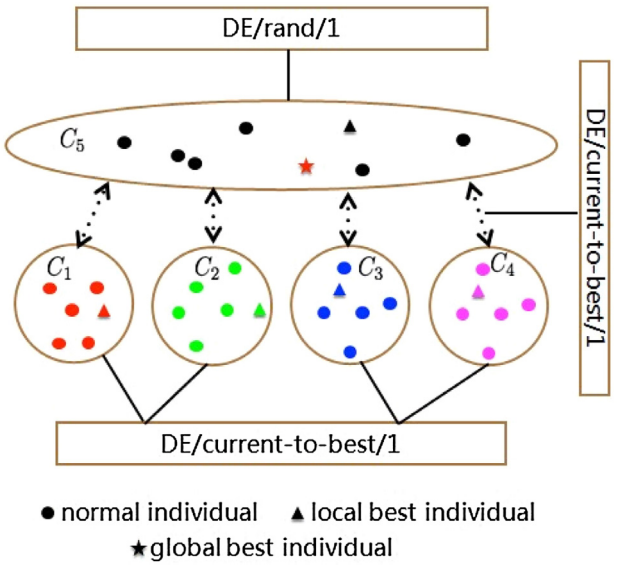


Fig. 4. Schematic diagram for the differential evolution with multi-cluster mutation.

selected randomly and they tend to add variety to the population. As can be seen from the mutation formulations, the individuals involved in mutation are randomly selected from the population. However, selecting them randomly will not fully bring the advantages of the mutation operators into practice. For the exploitative biased operators, to further enhance exploitation, a better way is to select the individuals involved in mutation around the vicinity of the current to best individual in the population. For the explorative biased operators, to further enhance exploration, a better way is to select the individuals involved in mutation far away from the mutant individual.

Motivated by these observations, we propose a cross-cluster mutation strategy. The proposed strategy is based on a multi-subpopulation framework, which works as follows:

1. The population is divided into several sub-populations.
2. Within each sub-population, the individuals are evolved by DE algorithm with a specific mutation operator, where the individuals involved in mutation are selected in the sub-population.
3. The evolution within each sub-population stops when clusters can be found, i.e., the evolution stops when the *h*-value reach a predefined threshold.
4. For the individuals in different sub-population, the algorithm applies explorative mutation, where the individuals involved in mutation are selected from other sub-populations.
5. Steps 2–4 are repeated until the stopping criteria is satisfied.

Fig. 4 shows a schematic diagram for the proposed evolution framework. The population is divided into sub-populations, which are labeled as C_1, C_2, C_3, C_4 and C_5, respectively. Among the sub-populations, C_1, C_2, C_3, C_4 are evolved by DE algorithm with DE/current-to-best/1, and C_5 is evolved by DE algorithm with DE/rand/1. C_1, C_2, C_3, C_4 tend to gather around their current to best individual, and C_5 tends to scatter in a wider region of the search space. Moreover, due to the random nature of the mutation operator, C_1, C_2, C_3, C_4 generally gather to different regions of the search space. When the evolutions within C_1, C_2, C_3, C_4 stop, the algorithm applies DE/current-to-best/1 to each individual, with some of the individuals involved in mutation taken from C_5. The framework is organized in such way so that the exploitive and explorative abilities of the mutation operators are fully utilized.

5. Experimental studies

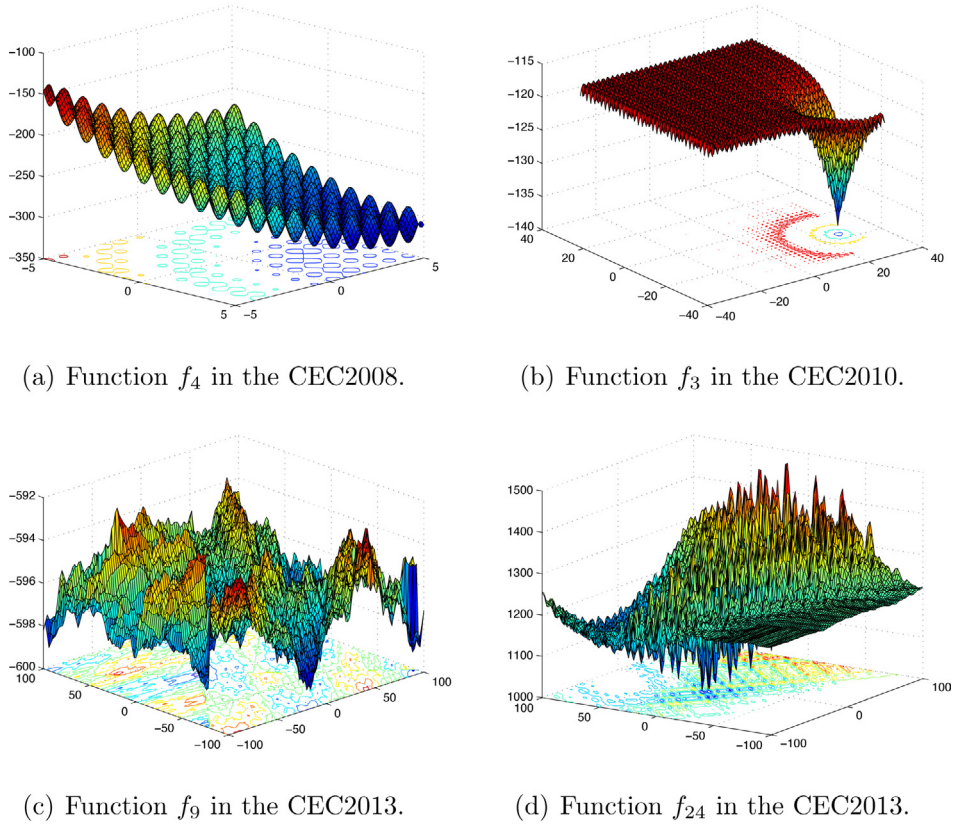
5.1. Test functions

In this section, experiments are conducted to test the performance of the proposed algorithm. In the experiments, the CEC2008, CEC2010, and CEC2013 are selected as benchmarks, which are shifted, rotated, expanded, and combined variants of the basic functions. The CEC2008 has seven functions. Among the functions, two of them are unimodal, and the others are multimodal. The CEC2010 has 20 functions. Among the functions, three of them are separable, 15 of them are partially separable, and two of them are completely nonseparable. The CEC2013 has 28 functions. Among the functions, five of them are unimodal, 15 of them are multimodal, and eight of them are composition functions. The CEC2008 and CEC2010 are designed to test the performance of algorithms on large scale optimization problems. The CEC2013 is designed to test the performance of algorithms on moderate scale optimization problems. While testing, the explicit equations of the problems are not allowed to be used by the algorithms. The properties and the formulas of them are reported in [50–52], respectively. As an example, Fig. 5 shows the surface landscapes of CEC2008 f_4, CEC2010 f_3, CEC2013 f_9, and CEC2013 f_{24} on the first and the second dimensions.

5.2. Experimental settings

To test the performance of the algorithm, four sets of experiments are carried out. The first set of experiments are carried out by applying the problem decomposition operator. The experiments are conducted on the CEC2010 benchmark, and the dimension of the functions is taken as 1000. The objective of the experiments is to illustrate the performance of the decomposition mechanism. The second set of experiments are carried out by applying the differential evolution operator. The experiments are conducted on the CEC2013 benchmark, and the dimensions of the functions are taken as 30 and 50, respectively. The objective of the experiments is to illustrate the performance of the cross-cluster optimization mechanism. The third and the fourth set of experiments are carried out by applying the entire algorithm, i.e., the problem decomposition operator and the differential evolution operator. The experiments are conducted on the CEC2008 and CEC2010 benchmarks, and the dimension of the functions is taken as 1000. The objective of the experiments is to test the overall performance of the proposed algorithm. For all of the experiments, the number of function evaluations (FEs) is used to evaluate the computational efforts.

In the experiment, the parameters are selected empirically so that the results having high quality solution can be obtained. In the variable interdependence searching stage, concerning the efficiency of the algorithm, the total number of test N is taken as 10 (Line 4 of Fig. 1). The next decision is on the parameters used in the differential evolution operator. The parameters F and CR are selected through studies reported in [5], which are taken as 0.5 and 0.8, respectively. The population size and the number of subpopulations are selected through experimental study. The experiments are conduct on CEC2008 f_1, CEC2008 f_5, CEC2010 f_1, and CEC2010 f_5. The results are presented in Table 1. In Table 1, the population size is taken as 20, 50, 80, 100, and 200, respectively, and the number of subpopulations is taken as 3, 4, 5, 6, and 7, respectively. For each run, the experiment is repeated for 25 times, and the mean value and the standard derivation of the 25 runs are presented. It can be observed from Table 1 that the parameters *population size* = 100, *number of subpopulations* = 5 yield the best overall performance. Thus we adopt this configuration in the rest of the experiments.



(a) Function f_4 in the CEC2008. (b) Function f_3 in the CEC2010.

(c) Function f_9 in the CEC2013. (d) Function f_{24} in the CEC2013.

Fig. 5. Surface landscapes of CEC2008 f_4, CEC2010 f_3, CEC2013 f_9, and CEC2013 f_{24}. The properties and the formulas are reported in [48–50].

5.3. Existing algorithms for comparison

For the purpose of comparison, we select the following algorithms which have been applied for all or some of the test functions.

1. Multiple trajectory search (MTS) [53]: the MTS uses multiple agents to search the solution space. Each agent searches locally using one of the three candidate local search methods. By choosing a local search method that best fits the landscape of a solution's neighborhood, an agent may find its way to a local optimum or the global optimum.
2. Estimation of distribution algorithm for large scale global optimization (LSEDA-gl) [54]: in LSEDA-gl, three strategies, i.e., sampling under mixed Gaussian & Lévy probability distribution, standard deviation control, and restart strategy are adopted to improve the performance of univariate EDA on large scale global optimization problems.
3. Self-adaptive differential evolution algorithm (jDEdynNP-F) [55]: in jDEdynNP-F, the control parameters F and CR are self-adapted and a population size reduction method is used. Additionally, it uses a mechanism for sign changing of control parameter F with a probability based on the fitness of randomly chosen vectors.
4. Memetic algorithm based on local search chains (MA-SW-Chains) [4]: the MA-SW-Chains assigns each individual a local search intensity. And several local search chains are applied to perform optimization task based on the intensity.
5. Two-stage based ensemble optimization evolutionary algorithm (EOEA) [56]: the EOEA has two stages. The objective of the first stage is to shrink the searching scope to the promising area, and the objective of the second stage is to explore the limited area extensively.
6. Dynamic multi-swarm particle swarm optimizer with sub-regional harmony search (DMS-PSO-SHS) [35]: in DMS-PSO-SHS, the PSO population is divided into sub-swarms. The sub-swarms are regrouped by regrouping schedules and information is exchanged among the particles in the whole swarm.

Among the above algorithms, the MTS, the LSEDA-gl, the jDEdynNP-F are top ranked algorithms of CEC2008 competition, and the MA-SW-Chains, the EOEA, the DMS-PSO-SHS are top ranked algorithms of CEC2010 competition. They have been shown to be successful and can find suitable solutions. A comparison with them will demonstrate the performance of our algorithm, and will show whether it is better or worse than other algorithms. In the rest of this paper, we term the proposed algorithm with problem decomposition as CDECC (cooperative differential evolution using cross-cluster mutation).

5.4. Simulated results and discussions

5.4.1. Results for the problem decomposition mechanism

This subsection aims to show the performance of the problem decomposition mechanism. The experiments are conducted on CEC2010 benchmark functions. Table 2 presents the grouping results after applying the problem decomposition operator. In the table, *Separability* is the separability of each function, *Found_G* is the number of groups partitioned using the proposed decomposition mechanism, *Actual_G* is the actual number of groups should be partitioned according to the report in the literature associated to the CEC2010 [49], and FEs is the computational efforts incurred while searching the interdependence among variables. As can be seen from Table 2, the proposed algorithm can obtain the correct decomposition strategy for most of the functions. According to the instructions associated to the CEC2010, the maximum number of

Table 1
Results of some benchmarks with different population size and number of subpopulations.

Func	Population size		Number of subpopulations				
			3	4	5	6	7
CEC2008 f_1	20	Mean	0.00E−00	0.00E−00	0.00E−00	0.00E−00	0.00E−00
		Std	0.00E−00	0.00E−00	0.00E−00	0.00E−00	0.00E−00
	50	Mean	0.00E−00	0.00E−00	0.00E−00	0.00E−00	0.00E−00
		Std	0.00E−00	0.00E−00	0.00E−00	0.00E−00	0.00E−00
	80	Mean	0.00E−00	0.00E−00	0.00E−00	0.00E−00	0.00E−00
		Std	0.00E−00	0.00E−00	0.00E−00	0.00E−00	0.00E−00
	100	Mean	0.00E−00	0.00E−00	0.00E−00	0.00E−00	0.00E−00
		Std	0.00E−00	0.00E−00	0.00E−00	0.00E−00	0.00E−00
	200	Mean	2.65E−25	2.59E−26	2.40E−26	2.05E−26	1.36E−24
		Std	7.66E−26	1.36E−26	5.53E−27	3.19E−27	2.56E−25
CEC2008 f_5	20	Mean	1.38E−13	1.17E−14	2.42E−14	5.62E−14	7.94E−14
		Std	7.86E−14	9.58E−15	2.88E−14	8.60E−14	6.96E−15
	50	Mean	7.87E−14	4.83E−14	3.14E−15	1.71E−15	6.96E−15
		Std	2.68E−14	4.54E−14	5.26E−15	1.37E−15	8.49E−16
	80	Mean	2.08E−15	1.37E−15	1.32E−15	7.03E−15	1.98E−14
		Std	1.71E−15	9.50E−16	1.76E−15	6.04E−15	1.64E−14
	100	Mean	3.34E−15	1.42E−15	3.85E−16	3.95E−16	7.52E−16
		Std	1.79E−15	9.71E−16	5.15E−17	4.09E−16	3.28E−16
	200	Mean	1.40E−12	5.48E−13	9.25E−13	2.42E−12	1.57E−12
		Std	6.87E−13	3.29E−13	4.09E−14	1.89E−13	3.17E−13
CEC2010 f_1	20	Mean	0.00E−00	0.00E−00	0.00E−00	0.00E−00	3.20E−30
		Std	0.00E−00	0.00E−00	0.00E−00	0.00E−00	2.39E−30
	50	Mean	0.00E−00	0.00E−00	0.00E−00	0.00E−00	9.67E−28
		Std	0.00E−00	0.00E−00	0.00E−00	0.00E−00	2.36E−28
	80	Mean	1.09E−26	1.21E−27	2.01E−28	1.65E−25	3.38E−27
		Std	6.69E−27	7.54E−27	7.51E−29	6.52E−26	6.75E−27
	100	Mean	0.00E−00	8.68E−28	2.47E−28	1.12E−27	5.27E−27
		Std	0.00E−00	3.57E−28	3.34E−29	9.72E−28	7.78E−27
	200	Mean	3.56E−24	1.16E−24	4.75E−26	1.25E−25	2.00E−26
		Std	3.17E−24	1.32E−25	1.47E−26	2.07E−26	3.56E−27
CEC2010 f_5	20	Mean	1.73E+08	1.77E+08	1.76E+08	1.94E+08	2.07E+08
		Std	4.09E+07	3.23E+07	4.59E+07	4.68E+07	3.27E+07
	50	Mean	1.59E+08	1.66E+08	1.53E+08	1.59E+08	1.84E+08
		Std	2.95E+07	2.67E+07	4.47E+07	6.97E+07	3.33E+07
	80	Mean	1.42E+08	1.61E+08	1.47E+08	1.33E+08	1.52E+08
		Std	3.72E+07	3.28E+07	3.90E+07	5.99E+07	5.13E+07
	100	Mean	1.49E+08	1.58E+08	1.35E+08	1.46E+08	1.39E+08
		Std	2.29E+07	2.67E+07	3.58E+07	2.92E+07	6.75E+07
	200	Mean	1.59E+08	1.57E+08	1.50E+08	9.25E+07	1.56E+08
		Std	1.14E+07	5.25E+07	4.60E+07	1.91E+07	2.27E+07

Table 2
Grouping results for the problem decomposition mechanism on the CEC2010 benchmark functions.

Func	*Separability*	*Found_G*	*Actual_G*	FEs
f_1	Separable	1000	1000	4000
f_2	Separable	1000	1000	4000
f_3	Separable	1000	1000	4000
f_4	Nonseparable	951	951	5796
f_5	Nonseparable	951	951	5796
f_6	Nonseparable	951	951	5796
f_7	Nonseparable	951	951	5796
f_8	Nonseparable	951	951	5052
f_9	Nonseparable	510	510	21,412
f_{10}	Nonseparable	510	510	21,412
f_{11}	Nonseparable	521	510	21,356
f_{12}	Nonseparable	510	510	21,412
f_{13}	Nonseparable	521	510	14,732
f_{14}	Nonseparable	20	20	35,652
f_{15}	Nonseparable	20	20	35,652
f_{16}	Nonseparable	20	20	35,652
f_{17}	Nonseparable	20	20	35,652
f_{18}	Nonseparable	27	20	34,056
f_{19}	Nonseparable	1	1	39,864
f_{20}	Nonseparable	1	1	39,864

FEs is 5.0×10^6. It can be seen that the total number of FEs incurred in the decomposition process is less than 2% of the total FEs.

5.4.2. Results for the cross-cluster optimization mechanism

This subsection aims to show the performance of the cross-cluster optimization mechanism. The experiments are conducted on CEC2013 benchmark function, and the dimensions of the functions are taken as 30 and 50, respectively. For each function, the results after 5.0×10^5 FEs are recorded. The best, median, worst values, the mean value, and the standard derivation of the 25 runs are presented. The results are presented in Table 3. It can be seen from Table 3 that the cross-cluster optimizer obtains results more accurate than 10^{-1} for 12 out of the 28 functions, including four unimodal functions (f_1, f_2, f_4, and f_5), seven basic multimodal functions ($f_6, f_7, f_{10}, f_{11}, f_{14}, f_{16}, f_{19}$), and one composition function (f_{22}). It also can be seen from Table 3 that cross-cluster optimizer obtains results more accurate than 10^3 for all of the remaining 16 functions. Among the 16 functions, one of them is unimodal function (f_3), eight of them are multimodal functions ($f_8, f_9, f_{12}, f_{13}, f_{15}, f_{17}0, f_{18}, f_{20}$), and seven of them are composition functions ($f_{21}, f_{23} - f_{28}$).

5.4.3. Results for the CEC2008 benchmark functions

This subsection aims to show the results of the CDECC for the CEC2008 benchmarks. The results are presented following the instructions reported in the literature associated to the CEC2008

Table 3
Results for the cross-cluster optimization mechanism on the CEC2013 benchmark functions.

		f_1	f_2	f_3	f_4	f_5	f_6	f_7
30D	Best	1.00E−08	1.00E−08	2.36E+03	1.00E−08	1.00E−08	1.00E−08	1.47E−01
	Median	1.00E−08	1.00E−08	3.22E+03	1.00E−08	1.00E−08	1.00E−08	1.69E−01
	Worst	1.00E−08	1.00E−08	1.83E+04	1.00E−08	1.00E−08	1.00E−08	3.35E−01
	Mean	1.00E−08	1.00E−08	2.50E+03	1.00E−08	1.00E−08	1.00E−08	1.47E−01
	Std	0.00E−00	0.00E−00	5.18E+03	0.00E−00	0.00E−00	0.00E−00	8.07E−02
50D	Best	1.00E−08	1.00E−08	9.72E+03	1.00E−08	1.00E−08	6.23E−02	2.76E−01
	Median	1.00E−08	1.00E−08	1.18E+05	1.00E−08	1.00E−08	4.34E−01	4.15E−01
	Worst	1.00E−08	1.00E−08	1.55E+06	1.00E−08	1.00E−08	4.91E−01	7.00E−01
	Mean	1.00E−08	1.00E−08	2.25E+05	1.00E−08	1.00E−08	4.07E−01	4.19E−01
	Std	0.00E−00	0.00E−00	3.48E+05	0.00E−00	0.00E−00	1.12E−01	5.88E−01

		f_8	f_9	f_{10}	f_{11}	f_{12}	f_{13}	f_{14}
30D	Best	2.07E+01	1.64E+01	7.39E−03	1.00E−08	1.62E+01	2.86E+01	1.00E−08
	Median	2.09E+01	2.38E+01	3.68E−02	1.00E−08	3.28E+01	7.10E+01	6.24E−02
	Worst	2.10E+01	2.98E+01	9.35E−02	1.00E−08	4.57E+01	1.29E+02	1.45E−01
	Mean	2.09E+01	2.39E+01	4.07E−02	1.00E−08	3.14E+01	7.31E+01	6.07E−02
	Std	6.04E−02	3.37E−00	2.50E−02	0.00E−00	7.35E−00	2.58E+01	3.39E−02
50D	Best	2.08E+01	3.03E+01	9.85E−03	1.00E−08	5.36E+01	1.45E+02	4.99E−02
	Median	2.11E+01	5.11E+01	2.95E−02	1.00E−08	9.65E+01	2.03E+02	9.99E−02
	Worst	2.11E+01	5.72E+01	1.03E−01	3.97E−00	1.41E+02	2.76E+02	1.62E−01
	Mean	2.11E+01	4.92E+01	3.66E−02	4.37E−01	9.85E+01	2.07E+02	1.03E−01
	Std	9.14E−02	6.07E−00	2.38E−02	1.11E−00	2.20E+01	4.26E+01	3.25E−02

		f_{15}	f_{16}	f_{17}	f_{18}	f_{19}	f_{20}	f_{21}
30D	Best	2.56E+03	2.59E−02	3.04E+01	5.41E+01	7.59E−02	8.45E−00	2.00E+02
	Median	3.36E+03	2.24E−01	3.04E+01	7.27E+01	1.23E−01	1.06E+01	3.00E+02
	Worst	3.84E+03	2.91E?01	3.04E+01	9.01E+01	1.71E−01	1.18E+01	4.43E+02
	Mean	3.29E+03	1.98E−01	3.04E+01	7.27E+01	1.23E−01	1.06E+01	3.00E+02
	Std	3.70E+02	8.73E−02	0.00E−00	9.09E−00	6.37E−02	4.82E−01	6.51E+01
50D	Best	6.29E+03	3.05E−02	5.07E+01	1.05E+02	1.83E−01	91.74E+01	2.00E+02
	Median	7.10E+03	2.58E−01	5.07E+01	1.47E+02	3.04E−01	1.90E+01	7.58E+02
	Worst	8.51E+03	3.61E−01	5.07E+01	2.01E+02	4.41E−01	2.08E+01	1.12E+03
	Mean	7.22E+03	1.98E−01	5.07E+01	1.47E+02	3.07E−01	1.91E+01	7.58E+02
	Std	5.09E+02	1.32E−01	0.00E−00	2.25E+01	6.15E−02	8.12E−01	4.07E+02

		f_{22}	f_{23}	f_{24}	f_{25}	f_{26}	f_{27}	f_{28}
30D	Best	1.15E−01	2.91E+03	2.04E+02	2.44E+02	2.00E+02	3.38E+02	1.00E+02
	Median	1.06E−00	3.71E+03	2.18E+02	2.60E+02	2.00E+02	5.09E+02	3.00E+02
	Worst	1.27E−00	4.76E+03	2.36E+02	2.92E+02	3.35E+02	7.96E+02	3.00E+02
	Mean	9.92E−01	3.73E+03	2.18E+02	2.61E+02	2.44E+02	5.14E+02	2.92E+02
	Std	6.71E−01	5.35E+02	7.23E−00	9.98E−00	6.02E+01	1.08E+02	4.00E+01
50D	Best	9.56E−01	5.33E+03	2.43E+02	3.09E+02	2.00E+02	7.57E+02	4.00E+02
	Median	1.20E−00	7.34E+03	2.65E+02	3.32E+02	3.75E+02	1.15E+03	4.00E+02
	Worst	1.68E−00	9.13E+03	2.90E+02	3.67E+02	4.39E+02	1.44E+03	4.00E+02
	Mean	1.24E−01	7.37E+03	2.65E+02	3.35E+02	3.70E+02	1.13E+03	4.00E+02
	Std	7.06E−01	7.63E+02	1.21E+01	1.51E+01	3.97E+01	1.35E+02	0.00E−00

[50]. For each run, the results after 5.0×10^4, 5.0×10^5, 5.0×10^6 FEs are recorded. The 1st, 7th, 13th, 19th, 25th values, the mean value, and the standard derivation of the 25 runs are presented. For functions $f_1 - f_6$, the values of $f(x) - f(x^*)$ are presented. For function f_7, since the optimum value is unknown, the function values are presented directly. The results are presented in Table 4. It can be seen that the CDECC obtains results more accurate than 10^{-13} for four out of the seven functions, including one nonseparable function f_5, and three separable functions f_1, f_4, f_6. For the functions f_2 and f_3, the CDECC approximates the optimum with accuracy 1.0×10^1 and 1.0×10^3, respectively. For the function f_7, the CDECC obtains value -1.47×10^4. For all the seven functions, the results with FEs = 5.0×10^5 are better than the results with FEs = 5.0×10^4, and the results with FEs = 5.0×10^6 are better than the results with FEs = 5.0×10^5. The results indicate that the CDECC can achieve improvements until the maximum FEs is reached.

Fig. 6 is a plot of the convergence graphs of the CDECC for functions $f_1 - f_7$. For each function, the curve is plotted using the results over the 25 runs. For $f_1 - f_6$, the abscissa is the number of function evaluations and the vertical axis is the value of $f(x) - f(x^*)$ averaged over the 25 runs. For f_7, the abscissa is the number of function evaluations and the vertical axis is the value of $f(x)$ averaged over the 25 runs. It can be seen from Fig. 6 that the evolution curves decrease at first and become flat after a turning point for functions $f_1 - f_3$, $f_5 - f_6$. For functions f_4 and f_7, the algorithm can improve the results steadily until the maximum number of FEs is reached.

For the purpose of comparison, Table 5 presents the results from 25 independent runs of CDECC and the results of MTS, LSEDA-gl, and jDEdynNP-F. All the results of the CDECC and the compared algorithms are taken after 5.6×10^6 FEs. "Mean" represents the average value of the obtained results. "Std" represents the standard deviation of the results. To determine the statistical differences between the CDECC and the compared algorithms, the Friedman test and Holm procedure are conducted [57]. The results are presented in Table 6. It can be seen from the Friedman test results that the differences among the four algorithms are statistically relevant with 95% certainty. The CDECC and the MTS obtain the best overall rank. The Holm procedure shows that the CDECC

Table 4
Results of the 25 independent runs for CEC2008 benchmark functions (*Dim* = 1000).

FEs		Func						
		f_1	f_2	f_3	f_4	f_5	f_6	f_7
5.0E+04	1st	1.35E+01	1.24E+02	6.59E+05	3.10E+03	2.97E−01	1.49E−00	−1.25E+04
	7th	1.51E+01	1.26E+02	1.47E+06	3.20E+03	3.23E−01	1.58E−00	−1.24E+04
	13th	1.63E+01	1.28E+02	2.10E+06	3.25E+03	3.87E−01	1.61E−00	−1.23E+04
	19th	1.72E+01	1.29E+02	3.22E+06	3.32E+03	5.11E−01	1.64E−00	−1.23E+04
	25th	2.00E+01	1.30E+02	7.72E+06	3.38E+03	8.86E−01	1.76E−00	−1.22E+04
	Mean	1.63E+01	1.27E02	2.78E+06	3.25E+03	4.52E−01	1.61E−00	−1.23E+04
	Std	1.57E−00	1.60E−00	1.77E+06	7.76E+01	1.68E−01	5.95E−02	5.51E+01
5.0E+05	1st	1.30E−05	1.03E+02	3.79E+03	5.15E+02	1.18E−07	2.52E−03	−1.31E+04
	7th	9.58E−05	1.05E+02	4.19E+03	6.10E+02	6.73E−07	2.82E−03	−1.31E+04
	13th	1.02E−04	1.05E+02	4.41E+03	6.33E+02	9.99E−07	2.97E−03	−1.30E+04
	19th	1.11E−04	1.06E+02	4.58E+03	6.45E+02	1.42E−06	3.24E−03	−1.30E+04
	25th	1.35E−04	1.07E+02	5.39E+03	6.69E+02	9.86E−06	4.69E−03	−1.30E+04
	Mean	1.01E−04	1.05E+02	4.41E+03	6.25E+02	1.45E−06	3.10E−03	−1.30E+04
	Std	2.34E−05	1.05E−00	3.28E+02	3.25E+01	1.87E−06	4.83E−04	4.44E+01
5.0E+06	1st	0.00E−00	7.17E+01	1.07E+03	5.33E−14	1.12E−15	2.09E−13	−1.50E+04
	7th	0.00E−00	7.21E+01	1.13E+03	8.88E−14	1.56E−15	2.34E−13	−1.48E+04
	13th	0.00E−00	7.33E+01	1.23E+03	1.24E−13	2.64E−15	2.41E−13	−1.44E+04
	19th	0.00E−00	7.56E+01	1.39E+03	1.60E−13	3.57E−15	2.52E−13	−1.41E+04
	25th	0.00E−00	7.84E+01	1.41E+03	3.91E−13	1.14E−14	2.98E−13	−1.39E+04
	Mean	0.00E−00	7.53E+01	1.24E+03	1.40E−13	1.32E−15	2.45E−13	−1.47E+04
	Std	0.00E−00	1.17E−00	1.89E+02	7.63E−14	1.76E−15	2.42E−14	1.64E+02

Table 5
Comparison results of CDECC and other algorithms for CEC2008 benchmark functions (*Dim* = 1000).

Func	CDECC		MTS		LSEDA-gl		jDEdynNP-F	
	Mean	Std	Mean	Std	Mean	Std	Mean	Std
f_1	**0.00E−00**	**0.00E−00**	**0.00E−00**	**0.00E−00**	3.22E−13	2.84E−14	1.23E−13	2.13E−14
f_2	7.53E+01	1.17E−00	4.72E−02	8.58E−03	**1.04E−05**	**5.11E−07**	2.12E+01	2.53E−00
f_3	1.24E+03	1.89E+02	**3.41E−04**	**1.38E−04**	1.73E+03	1.40E+02	1.50E+03	1.48E+02
f_4	1.40E−13	7.63E−14	**0.00E−00**	**0.00E−00**	5.45E+02	1.80E+01	1.47E−02	9.03E−03
f_5	1.32E−15	1.76E−15	**0.00E−00**	**0.00E−00**	1.71E−13	0.00E−00	5.34E−14	1.33E−07
f_6	**2.45E−13**	**2.42E−14**	1.24E−11	4.78E−13	4.26E−13	0.00E−00	4.94E−08	1.33E−07
f_7	**−1.47E+04**	**1.64E+02**	−1.39E+04	2.94E+01	−1.35E+04	5.72E+01	−1.35E+04	3.74E+01

obtains the same results with MTS, which is scored as the first rank algorithm in the CEC2008 competition. The Holm procedure also shows that the CDECC obtains better results than LSEDA-gl and jDEdynNP-F, and the differences are statistically relevant with 95% certainty.

5.4.4. Results for the CEC2010 benchmark functions

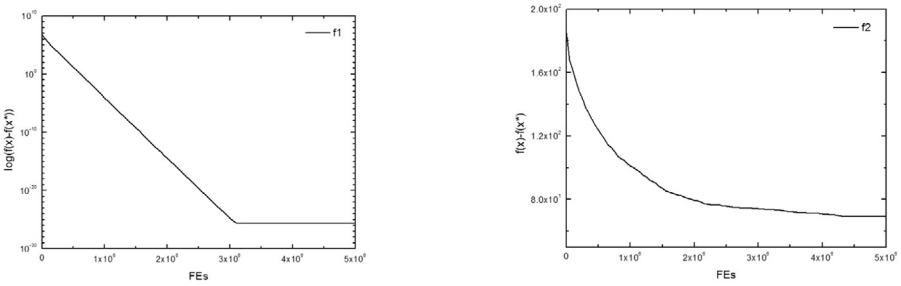
This subsection aims to show the results of the CDECC for the CEC2010 benchmarks. The results are presented following the instructions reported in the literature associated to the CEC2010 [51]. For each run, the results after 1.2×10^5, 6.0×10^5, 3.0×10^6 FEs are recorded. The best, median, worst values, the mean value, and the standard derivation of the 25 runs are presented. The results are presented in Table 7. It can be seen from Table 7 that the CDECC obtains results more accurate than 10^{-2} for five out of the 20 functions. Among the five functions, two of them are separable (f_1, f_3), and three of them are partially nonseparable (f_{11}, f_{16}, f_{17}). It also can be seen that the CDECC obtain results more accurate than 10^3 for eight out of the remaining 12 functions. Among the eight functions, seven of them ($f_2, f_6, f_{10}, f_{12}, f_{13}, f_{15}, f_{18}$) are partially nonseparable, and one of them (f_{20}) is completely nonseparable. For all the 20 functions, the results with FEs = 6.0×10^5 are better than the results with FEs = 1.2×10^5, and the results with 3.0×10^6 are better than the results with 6.0×10^5. The results indicate that the CDECC can achieve improvements until the maximum FEs is reached.

As an example, Fig. 7 is a plot of the convergence graphs of functions $f_2, f_5, f_8, f_{10}, f_{13}, f_{15}, f_{18}, f_{20}$. For each function, the curve is plotted using the results averaged over the 25 runs. As can be seen from Fig. 7, for function f_2, the evolution curve decreases linearly as the number of FEs increases at first, and then becomes flat after a turning point. For functions f_8, the evolution curve decreases steeply at first and then becomes flat after a turning point. For functions $f_5, f_{10}, f_{13}, f_{15}, f_{18}$ and f_{20}, the evolution curves decrease steadily at first, and then become flat steadily. It can also be seen from Fig. 7 that all the evolution curves are smooth, which indicates that the differences among different runs are not significant, thus indicates the stability of the CDECC.

For the purpose of comparison, Table 8 presents the results from 25 independent runs of CDECC and the results of studies using MA-SW-Chains, EOEA and DMS-PSO-SHS. All the results of the CDECC

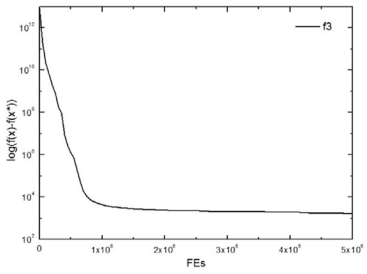
Table 6
Results of the Friedman test and Holm procedure ($\alpha = 0.05$).

Friedman test					Holm procedure			
Algorithm	Rank	χ^2	p value	Diff.?	Algorithm	z	p value	Diff.?
CDECC	1.79	9.0910	0.0280	Yes	–	–	–	–
MTS	1.79				CDECC *vs.* MTS	0.00	0.5000	No
LSEDA-gl	3.21				CDECC *vs.* LSEDA-gl	2.06	0.0197	Yes
jDEdynNP-F	3.21				CDECC *vs.* jDEdynNP-F	2.06	0.0197	Yes

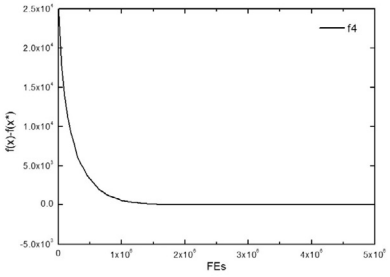


(a) f_1, dim=1000

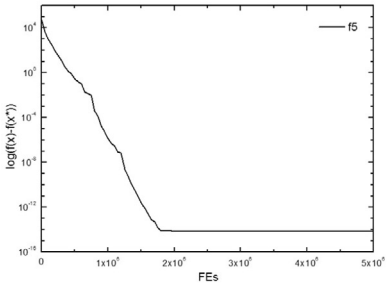
(b) f_2, dim=1000



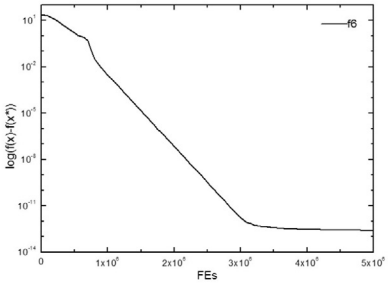
(c) f_3, dim=1000



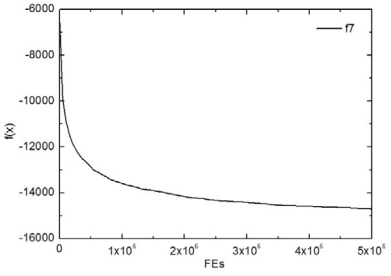
(d) f_4, dim=1000



(e) f_5, dim=1000



(f) f_6, dim=1000



(g) f_7, dim=1000

Fig. 6. Plots of evolution curves of the CDECC for CEC2008 functions. The results were obtained from 25 independent runs of the algorithms.

and the compared algorithms are taken after 3.0×10^6 FEs. It can be seen from Table 8 that CDECC obtains best results on eight out of the 20 functions, which include one separable function, and seven partially nonseparable functions. To determine the statistical differences between the CDECC and the compared algorithms, the Friedman test and Holm procedure are also conducted. The results are presented in Table 9. It can be seen that the differences among the four algorithms are not statistically relevant with 95% certainty, which indicates that all the CDECC and the compared algorithms are effective. Moreover, the CDECC obtains the best overall rank among the four algorithms. The Holm procedure shows that the CDECC obtains better results than MA-SW-Chains and EOEA, and the differences are statistically relevant with more than 95% certainty. When compared with DMS-PSO-SHS, though the differences between two algorithms are not statistically relevant with 95% certainty, the CDECC obtains better results, as indicated by $z = 0.54$ and p value = 0.2946.

5.5. Discussions

The results of the CDECC can be summarized as follows:

- The results for the problem decomposition mechanism indicate that the interdependence searching and the problem

Table 7
Results of the 25 independent runs For CEC2010 benchmark functions (*Dim* = 1000).

FEs		Func						
		f_1	f_2	f_3	f_4	f_5	f_6	f_7
1.2E+05	Best	1.36E+09	9.43E+03	8.90E+00	9.72E+12	4.65E+08	9.76E+06	1.37E+10
	Median	2.41E+09	1.07E+04	1.99E+01	2.18E+13	4.78E+08	1.56E+07	3.82E+10
	Worst	3.87E+09	1.43E+04	2.63E+01	3.26E+13	4.95E+08	1.98E+07	4.96E+10
	Mean	2.43E+09	1.13E+04	2.01E+01	3.18E+13	4.77E+08	1.60E+07	3.76E+10
	Std	2.54E+08	2.13E+03	8.32E+00	3.21E+12	1.83E+08	3.74E+06	1.91E+10
6.0E+05	Best	7.97E+02	1.90E+03	8.60E−01	8.62E+12	2.47E+08	2.96E+06	1.20E+09
	Median	8.18E+02	2.30E+03	1.32E+00	9.08E+12	4.15E+07	4.33E+06	1.89E+09
	Worst	9.32E+03	2.63E+03	2.47E−00	9.59E+12	4.35E+07	8.53E+06	2.30E+09
	Mean	8.17E+03	2.28E+03	1.42E−00	9.10E+12	4.16E+08	4.29E+06	1.86E+09
	Std	6.93E+02	7.59E+02	1.34E−00	8.79E+11	1.28E+08	3.00E+06	4.46E+08
3.0E+06	Best	0.00E−00	1.31E+02	2.18E−13	2.86E+12	7.89E+07	4.05E+03	4.58E+08
	Median	0.00E−00	1.84E+02	2.29E−13	6.50E+12	1.39E+08	1.49E+04	6.84E+08
	Worst	3.57E−27	2.11E+02	2.49E−13	8.16E+12	1.96E+08	8.45E+04	8.66E+08
	Mean	2.04E−28	1.74E+02	2.29E−13	6.10E+12	1.39E+08	1.94E+04	6.82E+08
	Std	7.51E−28	2.61E+01	7.02E−15	1.47E+12	3.58E+07	1.60E+04	1.26E+08

FEs		Func						
		f_8	f_9	f_{10}	f_{11}	f_{12}	f_{13}	f_{14}
1.2E+05	Best	1.47E+09	4.59E+09	1.07E+04	2.14E+02	8.34E+05	1.02E+09	4.74E+09
	Median	2.15E+09	4.59E+09	1.22E+04	2.00E+02	1.25E+06	1.92E+09	5.18E+09
	Worst	4.72E+09	4.59E+09	1.34E+04	2.23E+02	2.52E+06	2.58E+09	6.32E+09
	Mean	2.37E+09	4.59E+09	1.21E+04	2.01E+02	1.32E+06	1.90E+09	5.29E+09
	Std	3.27E+08	7.30E+07	4.23E+02	3.69E+01	6.14E+05	8.29E+08	1.14E+09
6.0E+05	Best	9.66E+07	4.37E+08	7.14E+04	1.18E−00	2.66E+05	1.91E+04	8.73E+08
	Median	1.24E+08	4.61E+08	7.33E+04	2.24E−00	3.80E+05	2.14E+04	9.47E+08
	Worst	1.87E+08	4.96E+08	7.57E+04	2.35E−00	4.67E+05	3.25E+04	1.92E+09
	Mean	1.28E+08	4.58E+08	7.34E+04	2.20E+00	3.75E+05	2.20E+04	9.52E+09
	Std	9.40E+07	2.06E+07	4.37E+02	4.17E−00	1.03E+05	5.79E+03	3.62E+09
3.0E+06	Best	3.76E+07	4.26E+07	1.08E+03	2.15E−03	2.38E+03	2.53E+02	1.41E+07
	Median	4.58E+07	5.00E+07	1.40E+03	1.56E−02	3.66E+03	6.63E+02	1.74E+07
	Worst	5.86E+07	5.63E+07	1.86E+03	1.91E−02	5.67E+03	7.98E+02	1.90E+07
	Mean	4.63E+07	4.92E+07	1.43E+03	1.44E−02	3.92E+03	6.55E+02	1.72E+07
	Std	5.09E+06	3.36E+06	2.09E+02	4.11E−03	1.03E+03	6.35E+01	1.40E+06

FEs		Func					
Func		f_{15}	f_{16}	f_{17}	f_{18}	f_{19}	f_{20}
1.2E+05	Best	1.45E+04	4.11E+02	3.08E+06	4.50E+10	6.53E+06	3.24E+11
	Median	1.57E+04	4.18E+02	3.27E+06	4.89E+10	6.74E+06	3.48E+11
	Worst	1.73E+04	4.32E+02	3.89E+06	5.02E+10	6.96E+06	3.76E+11
	Mean	1.58E+04	4.17E+02	3.24E+06	4.86E+10	6.73E+06	3.51E+11
	Std	5.30E+02	2.33E+01	6.06E+05	5.73E+09	4.05E+05	4.81E+10
6.0E+05	Best	1.02E+04	4.21E−00	8.62E+05	4.92E+07	2.05E+06	6.89E+07
	Median	1.20E+04	6.47E−00	8.88E+05	5.07E+07	2.14E+06	7.27E+07
	Worst	1.32E+04	3.02E+02	9.37E+05	5.38E+07	2.36E+06	7.65E+07
	Mean	1.21E+04	5.73E+01	8.90E+05	5.08E+07	2.12E+06	7.21E+07
	Std	2.08E+02	1.48E+02	1.99E+05	3.41E+07	1.27E+05	5.68E+07
3.0E+06	Best	1.48E+03	1.42E−13	1.05E−01	2.03E+03	1.10E+06	9.77E+02
	Median	1.95E+03	1.64E−13	1.49E−01	2.42E+03	1.27E+06	9.77E+02
	Worst	2.32E+03	8.79E−01	1.80E−01	2.94E+03	1.84E+06	9.78E+02
	Mean	1.93E+03	7.03E−02	1.48E−01	2.44E+03	1.31E+06	9.77E+02
	Std	2.12E+02	2.38E−01	2.26E−02	2.72E+02	1.76E+05	2.34E−01

decomposition mechanism are effective and the total number of FEs incurred in this process is less than 2% of the total FEs.

- The results for the cross-cluster optimization mechanism indicate that the proposed optimizer is effective for moderate scale problems.
- On the CEC2008 functions, the statistical tests show that the CDECC obtains the same overall results with MTS, which is scored as the first rank algorithm of CEC2008 competition. The statistical tests also show that the CDECC obtains better results than LSEDA-gl and JDEdynNP-F, which are scored as the second and third rank algorithms of CEC2008 competition, respectively.
- On the CEC2010 functions, the statistical tests show that the CDECC and the compared algorithms are not statistically different, which indicates that all the four algorithms are effective. However, the CDECC obtains the best overall rank. The statistical tests also show that the CDECC obtains better results than MA-SW-Chains and EOEA with statistical significance, and obtains better results than DMS-PSO-SHS.

The simulated results indicate that the CDECC can perform well on large scale numerical optimization problems. The reason that the CDECC obtains such results relies on the following issues. Firstly, the fast interdependence learning strategy allows partitioning a large scale problem into small scale subproblems so that the interdependencies among different subproblems are minimized. Secondly, the exploit biased mutations facilitate the algorithm converging to the global optimum. Thirdly, the exploration biased mutations enable the algorithm to escape when it trapped into

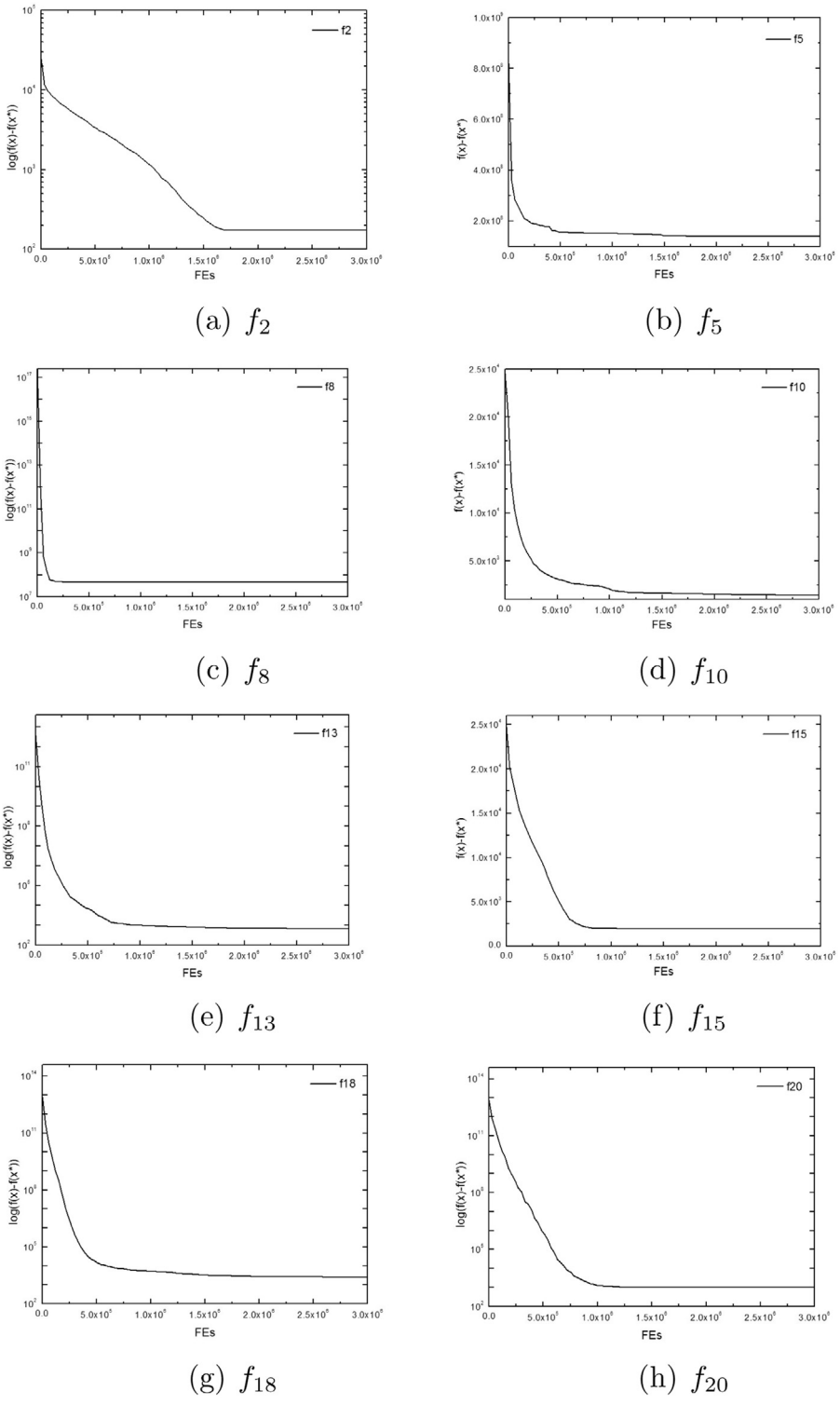


Fig. 7. Plots of evolution curves of the CDECC for CEC2010 functions. The results were obtained from 25 independent runs of the algorithms.

pseudo optimum. Due to the complexity of the real world optimizations, we cannot exhaust all of them to test the performance of the CDECC in our experiment. However, the experiment here serves as an illustration of the usefulness of the method and provides a guideline for researchers when designing related algorithms.

The application to practical problems is crucial for utilizing the CDECC. It can be inferred that the CDECC is independent of the practical problems and does not require to know about their exact structures. Before implementing CDECC, the key issues required include identifying and defining the decision parameters, the range of the parameters, and the evaluable objective function. Once the key issues are determined, an engineer is able to employ the CDECC to explore the parameters for yielding the best system performance.

Table 8
Comparison results of CDECC and other algorithms for CEC2010 benchmark functions ($Dim = 1000$).

Func	CDECC		MA-SW-Chains		EOEA		DMS-PSO-SHS	
	Mean	Std	Mean	Std	Mean	Std	Mean	Std
f_1	**2.04E−28**	**7.51E−28**	2.10E−14	1.99E−14	2.20E−23	2.87E−23	5.51E−15	4.00E−14
f_2	1.74E+02	2.61E+01	8.10E+02	5.88E+01	**3.62E−01**	**6.71E−01**	8.51E+01	2.06E+01
f_3	2.29E−13	7.02E−15	7.28E−13	3.40E−13	**1.67E−13**	**1.13E−14**	5.52E−11	3.25E−10
f_4	6.10E+12	1.47E+12	3.53E+11	3.12E+10	3.09E+12	1.61E+12	**2.46E+11**	**3.31E+10**
f_5	1.39E+08	3.58E+07	1.68E+08	1.04E+08	**2.24E+07**	**5.91E+06**	8.36E+07	6.10E+06
f_6	1.94E+04	1.60E+04	8.14E+04	2.84E+05	3.85E+06	4.97E+05	**8.28E−02**	**9.96E−01**
f_7	6.82E+08	1.26E+08	**1.03E+02**	**8.70E+01**	1.24E+02	1.55E+02	1.95E+03	1.56E+02
f_8	4.63E+07	5.09E+06	1.41E+07	3.68E+07	**1.01E+07**	**1.28E+07**	1.29E+07	1.91E+06
f_9	4.92E+07	3.36E+06	1.41E+07	1.15E+06	4.63E+07	4.78E+06	**8.72E+06**	**6.51E+05**
f_{10}	**1.43E+03**	**2.09E+02**	2.07E+03	1.44E+02	1.48E+03	6.91E+01	5.53E+03	5.18E+02
f_{11}	**1.44E−02**	**4.11E−03**	3.80E+01	7.35E−00	3.86E+01	1.65E+01	3.25E+01	3.00E−00
f_{12}	3.92E+03	1.03E+03	**3.62E−06**	**5.92E−07**	1.37E+04	2.90E+03	6.13E+02	6.00E+01
f_{13}	**6.55E+02**	**6.35E+01**	1.25E+03	5.72E+02	1.24E+03	4.59E+02	1.12E+03	1.05E+02
f_{14}	**1.72E+07**	**1.40E+06**	3.11E+07	1.93E+06	1.65E+08	8.95E+06	1.76E+07	1.55E+06
f_{15}	**1.93E+03**	**2.12E+02**	2.74E+03	1.22E+02	2.14E+03	1.22E+02	4.08E+03	2.17E+02
f_{16}	**7.03E−02**	**2.38E−01**	9.98E+01	1.40E+01	8.26E+01	1.68E+01	6.98E+01	4.24E−00
f_{17}	**1.48E−01**	**2.26E−02**	1.24E−00	1.25E−01	7.93E+04	8.80E+03	3.83E+03	4.15E+02
f_{18}	2.44E+03	2.72E+02	**1.30E+03**	**4.36E+02**	2.94E+03	6.92E+02	2.56E+03	1.16E+02
f_{19}	1.31E+06	1.76E+05	**2.85E+05**	**1.78E+04**	1.84E+06	9.97E+04	1.17E+06	1.06E+05
f_{20}	9.77E+02	2.34E−01	1.07E+03	7.29E+01	1.97E+03	2.35E+02	**3.52E+02**	**4.02E+01**

Table 9
Results of the Friedman test and Holm procedure ($\alpha = 0.05$).

Friedman test					Holm procedure			
Algorithm	Rank	χ^2	p value	Diff.?	Algorithm	z	p value	Diff.?
CDECC	2.03	6.785	0.079	No	–	–	–	–
MA-SW-Chains	2.85				CDECC *vs.* MA-SW-Chains	2.01	0.0222	Yes
EOEA	2.88				CDECC *vs.* EOEA	2.08	0.0188	Yes
DMS-PSO-SHS	2.25				CDECC *vs.* DMS-PSO-SHS	0.54	0.2946	No

6. Conclusions

To solve large scale optimization problems, one of the most common ways is to adopt the cooperative optimization strategies. The decomposition of the problem and the optimization of the subproblems are critical issues with regard to cooperative optimization. The main contributions of this paper included the following two aspects. Firstly, to perform the problem decomposition tasks, a fast learning strategy was proposed to capture the interdependencies among different variables, with which a large scale problem can be decomposed into small scale sub-problems without adding much computational efforts. Secondly, to optimize the subproblems more effectively and efficiently, a cross-cluster mutation strategy operator was proposed, with which the exploration and the exploitation abilities of the algorithm were further enhanced, thus accelerated the convergence to the global optimal solutions. The performance of the proposed algorithm was tested on benchmarks from three data sets. Simulated results showed that the proposed algorithm could find the optimal solutions for most of the test functions. From the study, we found that the proposed algorithm can perform well on nonseparable problems, especially when the problem possesses small scale interacted subcomponents.

In the future, this research will be extended to study the theoretical properties of the algorithm (in particular, the complexity and convergence analysis) and to apply the algorithm to solve real-life optimization problems such as manufacture scheduling and engineering design.

Acknowledgements

The authors would like to thank the support of the National Natural Science Foundation of China (61103146, 61272207, 61402076, 61202306), Startup Fund for the Doctoral Program of Liaoning Province (20141023), the Open Project of Shanghai Key Laboratory of Trustworthy Computing (07dz22304201301), the Fundamental Research Funds for the Central Universities (DUT12RC(3)72, DUT14QY06), the Open Project of the Key Laboratory of Ministry of Education (2014ACOCP02), and the Research Promotion Grant-in-Aid of Kochi University of Technology.

References

[1] A. Torn, A. Zilinskas, Global Optimization, Springer-Verlag, New York, 1989.
[2] R. Mallipeddia, P.N. Suganthana, Q.K. Panb, M.F. Tasgetirenc, Differential evolution algorithm with ensemble of parameters and mutation strategies, Appl. Soft Comput. 11 (2) (2011) 1679–1696.
[3] J. Brest, A. Zamuda, I. Fister, M.S. Maucec, Large scale global optimization using self-adaptive differential evolution algorithm, in: Proceedings of the 2010 Congress on Evolutionary Computation, Barcelona, Spain, 2010, pp. 1–8.
[4] D. Molina, M. Lozano, F. Herrera, MA-SW-chains: memetic algorithm based on local search chains for large scale continuous global optimization, in: Proceedings of the 2010 Congress on Evolutionary Computation, Barcelona, Spain, 2010, pp. 3153–3160.
[5] S.Z. Zhao, P.N. Suganthan, S. Das, Self-adaptive differential evolution with multi-trajectory search for large scale optimization, Soft Comput. 15 (1) (2011) 2175–2185.
[6] S. Das, P.N. Suganthan, Differential evolution: a survey of the state-of-the-art, IEEE Trans. Evol. Comput. 15 (1) (2011) 4–30.
[7] W. Jin, D. Kundur, T. Zourntos, K.L. Butler-Purry, A flocking-based paradigm for hierarchical cyber-physical smart grid modeling and control, IEEE Trans. Smart Grid 5 (6) (2014) 2687–2700.
[8] F. van den Bergh, A.P. Engelbrecht, A cooperative approach to particle swarm optimization, IEEE Trans. Evol. Comput. 8 (3) (2004) 225–239.
[9] A. Zimek, E. Schubert, H. Kriegel, A survey on unsupervised outlier detection in high-dimensional numerical data, Stat. Anal. Data Min. 5 (5) (2012) 363–387.
[10] M. Potter, K. de Jong, Cooperative coevolution: an architecture for evolving coadapted subcomponents, Evol. Comput. 8 (1) (2000) 1–29.
[11] M. Potter, K. de Jong, A cooperative coevolutionary approach to function optimization, in: Proceedings of the Third Conference on Parallel Problem Solving from Nature, Jerusalem, Israel, vol. 2, 1994, pp. 249–257.
[12] M. Potter, The Design and Analysis of a Computational Model of Cooperative Coevolution (PhD Thesis), George Mason University, 1997.

[13] Y. Liu, X. Yao, Q. Zhao, T. Higuchi, Scaling up fast evolutionary programming with cooperative coevolution, in: Proceedings of the 2001 Congress on Evolutionary Computation, Piscataway, NJ, USA, 2001, pp. 1101–1108.
[14] Y. Shi, H. Teng, Z. Li, Cooperative co-evolutionary differential evolution for function optimization, in: Proceedings of the First International Conference on Advances in Natural Computation, Changsha, China, 2005, pp. 1080–1088.
[15] A. Zamuda, J. Brest, B. Boskovic, V. Zumer, Large scale global optimization using differential evolution with self-adaptation and cooperative co-evolution, in: Proceedings of the 2008 Congress on Evolutionary Computation, Hong Kong, China, 2008, pp. 3718–3725.
[16] D. Sofge, K. de Jong, A. Schultz, A blended population approach to cooperative coevolution for decomposition of complex problems, in: Proceedings of the 2002 Congress on Evolutionary Computation, Honolulu, HI, USA, vol. 1, 2002, pp. 413–418.
[17] Z. Yang, K. Tang, X. Yao, Large scale evolutionary optimization using cooperative coevolution, Inf. Sci. 178 (15) (2008) 2985–2999.
[18] Z. Yang, K. Tang, X. Yao, Multilevel cooperative coevolution for large scale optimization, in: Proceedings of the 2002 Congress on Evolutionary Computation, Hong Kong, China, 2008, pp. 1663–1670.
[19] X. Li, X. Yao, Tackling high dimensional nonseparable optimization problems by cooperatively coevolving particle swarms, in: Proceedings of the 2009 Congress on Evolutionary Computation, Trondheim, Norway, 2009, pp. 1546–1553.
[20] K. Weicker, N. Weicker, On the improvement of coevolutionary optimizers by learning variable interdependencies, in: Proceedings of the 1999 Congress on Evolutionary Computation, Washington, DC, USA, 1999, pp. 1627–1632.
[21] T. Ray, X. Yao, A cooperative coevolutionary algorithm with correlation based adaptive variable partitioning, in: Proceedings of the 2009 Congress on Evolutionary Computation, Trondheim, Norway, 2009, pp. 983–989.
[22] C.K. Goh, D. Lim, L. Ma, Y.S. Ong, P.S. Dutta, A surrogate-assisted memetic coevolutionary algorithm for expensive constrained optimization problems, in: Proceedings of the 2011 Congress on Evolutionary Computation, New Orleans, Louisiana, USA, 2011, pp. 744–749.
[23] W. Chen, T. Weise, Z. Yang, Tang Ke, Large-scale global optimization using cooperative coevolution with variable interaction learning, in: Proceedings of the 11th International Conference on Parallel Problem Solving From Nature, Krakow, Poland, vol. 6239, 2010, pp. 300–309.
[24] L. Sun, S. Yoshida, X. Cheng, Y. Liang, A cooperative particle swarm optimizer with statistical variable interdependence learning, Inf. Sci. 186 (1) (2012) 20–39.
[25] D.E. Goldberg, Genetic algorithms in search, in: Optimization and Machine Learning, Addison-Wesley, Reading, MA, 1989.
[26] L.J. Fogel, Intelligence Through Simulated Evolution: Forty Years of Evolutionary Programming, John Wiley, 1999.
[27] X. Yao, Y. Liu, G. Lin, Evolutionary programming made faster, IEEE Trans. Evol. Comput. 3 (2) (1999) 82–102.
[28] A. Auger, Convergence results for the (1, λ)-SA-ES using the theory of φ-irreducible Markov chains, Theor. Comput. Sci. 334 (1–3) (2005) 35–69.
[29] J. Jagerskupper, How the (1 + 1) ES using isotropic mutations minimizes positive definite quadratic forms, Theor. Comput. Sci. 361 (1) (2006) 38–56.
[30] S.M. Islam, S. Das, S. Ghosh, S. Roy, P.N. Suganthan, An adaptive differential evolution algorithm with novel mutation and crossover strategies for global numerical optimization, IEEE Trans. Syst. Man Cybern. B: Cybern. 42 (2) (2012) 482–500.
[31] M. Dorigo, V. Maniezzo, A. Colorni, Ant system: optimization by a colony of cooperating agents, IEEE Trans. Syst. Man Cybern. B: Cybern. 26 (1) (1996) 29–41.
[32] J. Kennedy, R. Eberhart, Particle swarm optimization, in: Proceedings of the IEEE International Conference on Neural Networks, Perth, Australia, vol. 4, 1995, pp. 1942–1948.
[33] E. Eberhart, Y. Shi, Particle swarm optimization: developments, applications and resources, in: Proceedings of the 2000 Congress on Evolutionary Computation, San Diego, USA, 2000, pp. 84–88.
[34] S. Janson, M. Middendorf, A hierarchical particle swarm optimizer and its adaptive variant, IEEE Trans. Syst. Man Cybern. B: Cybern. 35 (6) (2005) 1272–1282.
[35] S. Zhao, P.N. Suganthan, S. Das, Dynamic multi-swarm particle swarm optimizer with subregional harmony search, in: Proceedings of the 2010 Congress on Evolutionary Computation, Barcelona, Spain, 2010, pp. 1983–1990.
[36] S.Z. Zhao, P.N. Suganthan, Q.K. Pan, M.F. Tasgetiren, Dynamic multi-swarm particle swarm optimizer with harmony search, Expert Syst. Appl. 38 (4) (2011) 3735–3742.
[37] M. Nasir, D. Maity, S. Das, S. Sengupta, U. Haldar, P.N. Suganthan, A dynamic neighborhood learning based particle swarm optimizer for global numerical optimization, Inf. Sci. 209 (20) (2012) 16–36.
[38] S. Das, A. Biswas, S. Dasgupta, A. Abraham, Bacterial foraging optimization algorithm: theoretical foundations analysis, and applications, Found. Comput. Intell. 3 (2009) 23–55.
[39] S. Kirkpatrick, C.D. Gelatt, M.P. Vecchi, Optimization by simulated annealing, Science 220 (4598) (1983) 671–680.
[40] F. Glover, M. Laguna, Tabu Search, Kluwer Academic Publishers, Boston, MA, 1997.
[40] Ali R. Yildiz, Hybrid Taguchi-harmony search algorithm for solving engineering optimization problems, Int. J. Ind. Eng. Theory Appl. Pract. 15 (3) (2008) 286–293.
[42] R. Storn, K.V. Price, Differential evolution: a simple and efficient adaptive scheme for global optimization over continuous spaces, J. Glob. Optim. 11 (4) (1997) 341–359.
[43] R. Storn, System design by constraint adaptation and differential evolution, IEEE Trans. Evol. Comput. 3 (1) (1999) 22–34.
[44] K.V. Price, R. Storn, J. Lampinen, Differential Evolution: A Practical Approach to Global Optimization, Springer, 2005, ISBN 3-540-20950-6.
[45] F. Neri, V. Tirronen, Recent advances in differential evolution: a review and experimental analysis, Artif. Intell. Rev. 33 (1) (2010) 61–106.
[46] M. Weber, F. Neri, V. Tirronen, A study on scale factor in distributed differential evolution, Inf. Sci. 181 (12) (2011) 2488–2511.
[47] J. Ronkkonen, S. Kukkonen, P.V. Price, Real parameter optimization with differential evolution, in: Proceedings of the 2005 Congress on Evolutionary Computation, Edinburgh, UK, vol. 1, 2005, pp. 506–513.
[48] J. Brest, S. Greiner, B. Boskovic, M. Mernik, V. Zumer, Self-adapting control parameters in differential evolution: a comparative study on numerical benchmark problems, IEEE Trans. Evol. Comput. 10 (6) (2006) 646–657.
[49] 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.
[50] K. Tang, X. Yao, P.N. Suganthan, C. MacNish, Y.P. Chen, C.M. Chen, Z. Yang, Benchmark Functions for the CEC'2008 Special Session and Competition on Large Scale Global Optimization, Technical Report, Nature Inspired Computation and Applications Laboratory, USTC, China, 2008.
[51] K. Tang, X. Li, P.N. Suganthan, Z. Yang, T. Weise, Benchmark Functions for the CEC 2010 Special Session and Competition on Large Scale Global Optimization, Technical Report, Nature Inspired Computation and Applications Laboratory, USTC, China, 2009.
[52] J. Liang, B. Qu, P.N. Suganthan, Alfredo G. Hernández-Díaz, Problem Definitions and Evaluation Criteria for the CEC 2013 Special Session on Real-Parameter Optimization, Technical Report, Computational Intelligence Laboratory, Zhengzhou University, Zhengzhou, China, 2012.
[53] Y. Tseng, C. Chen, Multiple trajectory search for large scale global optimization, in: Proceedings of the 2008 Congress on Evolutionary Computation, Hong Kong, 2008, pp. 3052–3059.
[54] Y. Wang, B. Li, A restart univariate estimation of distribution algorithm: sampling under mixed Gaussian and Levy probability distribution, in: Proceedings of the 2008 Congress on Evolutionary Computation, Hong Kong, 2008, pp. 3917–3924.
[55] J. Brest, A. Zamuda, B. Boskovic, M. Maucec, V. Zumer, High dimensional real-parameter optimization using self-adaptive differential evolution algorithm with population size reduction, in: Proceedings of the 2008 Congress on Evolutionary Computation, Hong Kong, 2008, pp. 2032–2039.
[56] Y. Wang, B. Li, Two-stage based ensemble optimization for large-scale global optimization, in: Proceedings of the 2010 Congress on Evolutionary Computation, Barcelona, Spain, 2010, pp. 1–8.
[57] S. Garcia, D. Molina, M. Lozano, F. Herrera, A study on the use of non-parametric tests for analyzing the evolutionary algorithms' behaviour: a case study on the CEC'2005 special session on real parameter optimization, J. Heuristics 15 (2009).