Dual Differential Grouping: A More General Decomposition Method for Large-Scale Optimization

Jian-Yu Li, *Student Member, IEEE*, Zhi-Hui Zhan, *Senior Member, IEEE*, Kay Chen Tan, *Fellow, IEEE*, and Jun Zhang, *Fellow, IEEE*

***Abstract*—Cooperative coevolution (CC) algorithms based on variable decomposition methods are efficient in solving large-scale optimization problems (LSOPs). However, many decomposition methods, such as the differential grouping (DG) method and its variants, are based on the theorem of function additively separable, which may not work well on problems that are not additively separable and will result in a bottleneck for CC to solve various LSOPs. This deficiency motivates us to study how the decomposition method can decompose more kinds of separable functions, such as the multiplicatively separable function, to improve the general problem-solving ability of CC on LSOPs. With this concern, this article makes the first attempt to decompose multiplicatively separable functions and proposes a novel method called dual DG (DDG) for better LSOP decomposition and optimization. The novelty and advantage of DDG are that it can be suitable for not only additively separable functions but also multiplicatively separable functions, which can considerably expand the application scope of CC. In this article, we will first define the multiplicatively separable function, and then mathematically show its relationship to the additively separable function and how they can be transformed into each other. Based on this, the DDG can use two kinds of differences to detect the separable structure of both additively and multiplicatively separable functions. In addition, the time complexity of DDG is analyzed and a DDG-based CC algorithm framework is developed for solving LSOPs. To verify the superiority of DDG, experiments and comparisons with some state-of-the-art and champion algorithms are conducted not only on 30 LSOPs based on the test suite of the IEEE CEC large-scale global optimization competition, but also on a case study of the parameter optimization for a neural network-based application.**

***Index Terms*—Cooperative coevolution (CC), differential evolution, dual differential grouping (DDG), evolutionary computation (EC), large-scale optimization problem (LSOP), particle swarm optimization.**

Manuscript received 4 May 2021; revised 28 September 2021 and 14 December 2021; accepted 26 February 2022. Date of publication 25 March 2022; date of current version 17 May 2023. This work was supported in part by the National Key Research and Development Program of China under Grant 2019YFB2102102; in part by the National Natural Science Foundations of China under Grant 62176094 and Grant 61873097; in part by the Key-Area Research and Development of Guangdong Province under Grant 2020B010166002; in part by the Guangdong Natural Science Foundation Research Team under Grant 2018B030312003; and in part by the National Research Foundation of Korea under Grant NRF-2021H1D3A2A01082705. This article was recommended by Associate Editor H. Ishibuchi. *(Corresponding authors: Zhi-Hui Zhan; Jun Zhang.)*

Jian-Yu Li and Zhi-Hui Zhan are with the School of Computer Science and Engineering, South China University of Technology, Guangzhou 510006, China, also with the Pazhou Laboratory, Guangzhou 510330, China, and also with the Guangdong Provincial Key Laboratory of Computational Intelligence and Cyberspace Information, Guangzhou 510006, China (e-mail: zhanapollo@163.com).

Kay Chen Tan is with the Department of Computing, The Hong Kong Polytechnic University, Hong Kong, SAR (e-mail: kctan@polyu.edu.hk).

Jun Zhang is with Hanyang University, Ansan 15588, South Korea (e-mail: junzhang@ieee.org).

This article has supplementary material provided by the authors and color versions of one or more figures available at https://doi.org/10.1109/TCYB.2022.3158391.

Digital Object Identifier 10.1109/TCYB.2022.3158391

I. Introduction

LARGE-SCALE optimization problems (LSOPs), which are becoming increasingly ubiquitous in the research community and real-world applications, have attracted increasing attention in recent years [1]–[3]. Due to the "curse of dimensionality," a large number of decision variables make the landscape of LSOPs highly complex and very difficult to be optimized [4]–[7]. As evolutionary computation (EC) algorithms are efficient tools for solving various complex optimization problems [8]–[10], such as multimodal [11], [12]; multi-/many-objective [13], [14]; and expensive optimization problems [15], [16], many researchers have also studied powerful EC-based algorithms for solving LSOPs [17]–[19]. In this direction, the cooperative coevolution (CC) framework has achieved great success and, therefore, has been widely studied in recent years [20]–[22].

Inspired by the "divide-and-conquer" mechanism, the core idea of the CC framework is to decompose the LSOP into several nonoverlapped subproblems with lower dimensions and then optimize each subproblem using EC algorithms [23]–[26]. To better describe the idea of CC, Fig. 1 presents a general CC framework. As shown in Fig. 1, the CC framework mainly has three stages: 1) the decomposition stage, where the problem will be decomposed into several subproblems by the decomposition method; 2) the optimization stage, where the subproblems will be optimized by EC algorithms; and 3) the combination stage, where the subsolutions for corresponding subproblems will be combined to form the complete solution. As decomposition is the essential and critical stage in CC, the quality of decomposition can greatly influence the optimization results.

Therefore, to better achieve problem decomposition, many decomposition methods have been proposed and researched [27]–[30]. Generally speaking, existing decomposition methods can be roughly classified into two categories:

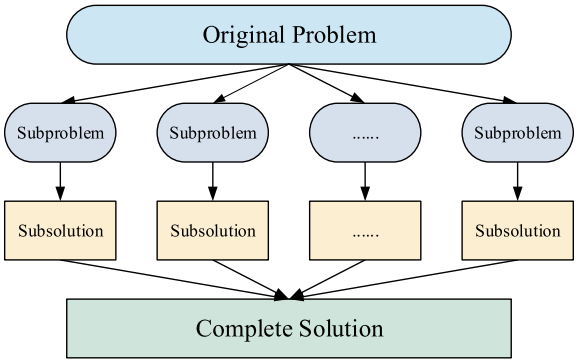


Fig. 1. General CC framework.

static and dynamic decomposition methods [31]–[33]. Intuitively, static decomposition methods attempt to obtain satisfactory decomposition results before the optimization stage and keep the decompositions fixed during the whole optimization process [34], [35], while dynamic methods decompose the problem dynamically according to the information obtained during the optimization process [36]–[38]. Among the existing methods, differential grouping (DG) [27] and its variants have obtained great success in decomposing LSOPs, especially additively separable LSOPs. Therefore, many enhanced DG methods have been proposed in the literature [28]–[30].

However, as many existing DG-based methods are based on the theorem of function additively separable, their decomposition abilities will deteriorate if the problem is not additively separable. For example, (1) presents an objective function of a minimization problem as

$$\begin{aligned} \min_{x_1,x_2} g_{\text{example}}\ (x_1, x_2) &= 2x_1x_2 + 5x_1 + 14x_2 + 35 \\ \text{s.t.}\ \ x_1 &\in [-5, 5], x_2 \in [-2, 2] \end{aligned} \tag{1}$$

where we know that (1) can be decomposed into two subproblems of x_1 and x_2 for independent optimization because the optimal value of x_1 is exactly a constant (i.e., –5), regardless of the value of x_2, and the optimal value of x_2 is also a constant (i.e., –2), regardless of the value of x_1. Therefore, instead of optimizing the whole (1), we can separate variables x_1 and x_2 and then optimize subproblems g_1 and g_2 independently as

$$\min_{x_1} g_1(x_1) = (2x_2 + 5) \cdot x_1 + 14x_2 + 35, x_1 \in [-5, 5] \tag{2}$$

$$\min_{x_2} g_2(x_2) = (2x_1 + 14) \cdot x_2 + 5x_1 + 35, x_2 \in [-2, 2] \tag{3}$$

where in (2), x_1 is the variable while x_2 is considered a function parameter with a fixed value belonging to $[-2, 2]$. Similarly, (3) is a function of x_2, while x_1 is considered a function parameter with a fixed value belonging to $[-5, 5]$. Therefore, after the decomposition, (2) is easy to obtain the optimal value of $x_1 = -5$, and (3) is also easy to optimize with $x_2 = -2$ as the optimum. However, the separable problem in (1) cannot be correctly decomposed by existing DG methods because the term "$2x_1x_2$" is not additively separable [27]. That is, existing DG-based methods will consider x_1 and x_2 as interacting and nonseparable and, therefore, group them in the same group, which is not suitable. The key issue behind this phenomenon is that the additively separable structure is not the only separable structure. In other words, even though a separable problem is not additively separable, it may be decomposed in other ways. Therefore, it is a potential and promising direction to explore a more general decomposition method for further improving the problem-solving ability of CC for LSOP. In fact, in many real-world applications, optimization problems can include variables with multiplicative interactions. For example, neural networks (NNs), including deep NNs and convolutional NNs [39], have become essential in various applications nowadays [40], where the NNs usually have a large number of parameters to be optimized to obtain better performance [41]. However, the large-scale and complex parameter optimization problem of NNs can contain many multiplicatively separable variables because the input of each layer will be multiplied by the parameters of the current layer to generate the input of the next layer. Therefore, besides the additively separable structure, it will be more promising to consider the decomposition of more kinds of separable structures including the multiplicatively separable structure.

With the above concerns, this article proposes a novel dual DG (DDG) method to achieve a more general and better problem decomposition. Compared to the existing DG-based methods, the advantage of DDG is that it can be suitable for not only additively separable problems but also multiplicatively separable problems, which can greatly expand the application scope of CC from single separable problems (i.e., additively) to dual separable problems (i.e., both additively and/or multiplicatively). For example, the problem given in (1) is multiplicatively separable and, therefore, it can be decomposed by the DDG, which will be described later as an example in Section III-A. Moreover, in Section III-A, we will first give the definition of a multiplicatively separable function and then mathematically show the relationship between the additively and multiplicatively separable functions and how they can be transformed into each other. Based on this, the details of DDG will be provided. After that, we will further develop a DDG-based CC algorithm for solving LSOPs and theoretically analyze the time complexity of DDG.

The major novelties and contributions of this article are summarized as follows.

1) The definition of the multiplicatively separable function and the relationship between it and the additively separable function are mathematically provided in this article. More importantly, two related theorems are then given and proved, which can guide the detection of the separable structure in multiplicatively separable problems.
2) Based on the given theorems, the DDG is proposed to achieve a more general decomposition ability for LSOP. By utilizing two different kinds of differences to detect separable structures, the DDG can decompose not only additively separable problems but also multiplicatively separable problems efficiently. Moreover, the time complexity of DDG is also analyzed and given in this article.
3) A DDG-based CC algorithm for solving LSOPs is further developed by combining the DDG with a classical and widely used CC framework.

To evaluate the proposed DDG and the DDG-based algorithms, experimental studies are conducted on 30 LSOPs, which are selected and generated from the widely used LSOP benchmarks in the latest IEEE CEC 2013 large-scale global optimization (LSGO) competitions test suite [42]. Furthermore, some state-of-the-art decomposition methods and algorithms, including the champion algorithm, are employed in the experimental comparisons to show the superiority of DDG. In addition, this article also conducts a case study on the large-scale parameter optimization of an NN-based three-category classification application, so as to further evaluate the real-world application potential of the proposed DDG.

The remainder of this article is organized as follows: Section II briefly introduces the background and related work, and Section III details the proposed methods and the time complexity of DDG. Experiments, including the settings, comparisons, and analyzes, are provided in Section VI. Finally, Section V presents the conclusion.

II. Background and Related Work

A. Separable Function and DG

The separable function is defined as follows.

Definition 1 [28]: A function $f(x)$ is partially separable for minimization with k independent components if and only if

$$\arg\min_{\mathbf{x}} f(\mathbf{x}) = \left(\arg\min_{\mathbf{x}_1} f(\mathbf{x}_1, ***) \; \arg\min_{\mathbf{x}_2} f(***, \mathbf{x}_2, ***), \ldots, \arg\min_{\mathbf{x}_k} f(***, \mathbf{x}_k)\right) \tag{4}$$

where $\mathbf{x}_1, \ldots,$ and $\mathbf{x}_k$ are k nonoverlapped subvectors of $\mathbf{x}$, and the "***" in the parentheses can be any value in the corresponding search space. Note that according to (4), the optimal value of $\mathbf{x}_i$ $(1 \leq i \leq k)$ should be the same no matter what the value of "***" is, and the combination of all optimal nonoverlapped subvectors should be the optimal solution to the original problem. Based on the above, if $\mathbf{x}_i$ and $\mathbf{x}_j$ $(i \neq j)$ are two nonoverlapped subvectors of $\mathbf{x}$ that satisfy (4), any variable in $\mathbf{x}_i$ and that in $\mathbf{x}_j$ are separable and do not interact with each other. In addition, note that the "argmin" should be "argmax" for maximization problems.

As a special type of partially separable function, the additively separable function is defined as follows.

Definition 2 [28]: A function f is partially additively separable if it has the following form:

$$f(\mathbf{x}) = \sum_{i=1}^{k} f_i(\mathbf{x}_i), 1 < k \leq D \tag{5}$$

where $\mathbf{x}_1, \mathbf{x}_2, \ldots,$ and $\mathbf{x}_k$ are k nonoverlapped subvectors of $\mathbf{x}$, functions $f_1, f_2, \ldots,$ and f_k are subfunctions of function f, and D is the total dimension of $\mathbf{x}$. Specifically, the function f is also called fully additively separable if k equals D, while it is regarded as fully nonseparable if $k = 1$. In (5), we can see that the change of a subvector $\mathbf{x}_i$ will only change the value of f_i and then change the original function f, but will not change the value on other subfunctions $f_j (i \neq j)$ because $\mathbf{x}_i$ is not the variable in f_j. Based on this, we have the following theorem:

Theorem 1 [27]: Given an additively separable function $f(x)$ with D-dimensional decision variable $\mathbf{x} = (x_1, x_2, \ldots, x_D)$, for any real value a, b, $c = b + \delta (\delta \neq 0)$, and $d (b \neq d)$, if the following condition holds for two variables x_i and x_j:

$$f(\mathbf{x})\big|_{x_i=a,x_j=b} - f(\mathbf{x})\big|_{x_i=c,x_j=b} \neq f(\mathbf{x})\big|_{x_i=a,x_j=d} - f(\mathbf{x})\big|_{x_i=c,x_j=d} \tag{6}$$

where

$$f(\mathbf{x})\big|_{x_i=a,x_j=b} = f(***, x_i = a, ***, x_j = b, ***) \tag{7}$$

then x_i and x_j are additively nonseparable.

Based on this theorem, the idea of DG is to check whether (6) holds for every two variables to determine the additively separable structure of an objective function. Specifically, if (6) is satisfied, then x_i and x_j are additively nonseparable, which will be regarded as interacting variables and grouped together. Due to the calculation error and the computational precision of the computer system, the DG method checks (8) instead of (6) in practical implementations.

$$\left|\left[f(\mathbf{x})\big|_{x_i=a,x_j=b} - f(\mathbf{x})\big|_{x_i=c,x_j=b}\right] - \left[f(\mathbf{x})\big|_{x_i=a,x_j=d} - f(\mathbf{x})\big|_{x_i=c,x_j=d}\right]\right| \leq \varepsilon_{\text{addi}} \tag{8}$$

where $\varepsilon_{\text{addi}}$ is the acceptance threshold for detecting additively separable variables.

B. Related Work

In this section, we briefly review the related work about variable interaction within the EC community. As briefly mentioned in Section I, decomposition methods can be roughly divided into two categories: 1) static decomposition methods [34] and 2) dynamic decomposition methods [36]–[41]. In general, both static and dynamic decomposition methods have advantages and disadvantages. Therefore, these researches are suitable for different situations [20]. As the proposed DDG in this article is a static decomposition method, the following contents primarily describe the related work on static decomposition methods.

Usually, a good decomposition requires the appropriate separable structure of the objective function. To analyze the separable structure, detecting and learning the interactions among decision variables are essential. Therefore, many variable interaction learning methods have been proposed. Chen *et al.* [43] proposed a variable interaction learning strategy for problem decompositions. In this strategy, each variable is initially considered as a separate group, and then the groups that affect each other are merged, which finally obtains the groups that are independent of each other. Omidvar *et al.* [27] proposed the classical DG method, which was for detecting the interactions among variables in additively separable functions.

To date, DG has shown great efficiency in problem decomposition because it can capture the interactions among variables in additively separable functions. However, DG cannot detect the indirect interactions between variables. For this problem, Mei *et al.* [30] proposed global DG (GDG) to detect the indirect interactions between variables. GDG uses a matrix

to denote the interactions among variables, where each element in the matrix represents the interaction degree between two variables. After obtaining the matrix by checking (8), GDG performs the breadth-first or depth-first technique to identify the direct and indirect interactions between variables. Similarly, Sun *et al.* [29] proposed extended DG (XDG), which iteratively detected the interactions between every two variables and accordingly divided the variables into several groups.

Although the above methods are powerful for detecting variable interactions, these methods require a large number of fitness evaluations (FEs), which can be an expensive cost in solving LSOPs. To address this issue, many studies have been proposed for detecting variable detections using fewer FEs [28], [31]. For example, Hu *et al.* [34] proposed a fast interdependency identification mechanism to save many FEs. Furthermore, Omidvar *et al.* [28] proposed DG2, which reused some sample points to reduce the need of FEs in the original DG. In addition, Sun *et al.* [31] were inspired by a binary search and proposed a recursive DG (RDG), where the detections of variable interactions were performed in a binary recursive manner. Moreover, the improved RDG method, called RDG3, has also been proposed and studied for overlapping functions [32].

In addition to the above DG-based methods, other decomposition methods have also been studied in recent years. For instance, Ge *et al.* [33] proposed a two-stage variable interaction method, where a learning model was first employed to explore some knowledge and then a marginalized denoising model was adopted to obtain the overall variable interactions based on the knowledge obtained in the first stage. Wang *et al.* [44] proposed a formula-based grouping that assumed the formula of the objective function was known before the optimization. Liu *et al.* [45] proposed a hybrid deep grouping method that not only considered variable interaction but also variable essentials, which was suitable for decomposing nonseparable problems.

In conclusion, a considerable number of the above methods are DG variants or are based on Theorem 1, which are only suitable for additively separable problems [46], [47]. Different from these methods, the DDG proposed in this article is useful for not only additively separable problems but also multiplicatively separable problems.

III. Proposed DDG

A. Multiplicatively Separable Function

The definition of a multiplicatively separable function is given in this article as follows.

Definition 3: A function g is partially multiplicatively separable if it has the following form:

$$g(\mathbf{x}) = \prod_{i=1}^{k} g_i(\mathbf{x}_i),\ 1 < k \le D \tag{9}$$

where $\mathbf{x}_1$, $\mathbf{x}_2$, …, and $\mathbf{x}_k$ are k nonoverlapped subvectors of $\mathbf{x}$, functions g_1, g_2, …, and g_k are subfunctions of function g, and D is the total dimension of $\mathbf{x}$. Specifically, function g is also considered as fully multiplicatively separable if k equals D, while it is regarded as multiplicatively nonseparable if $k = 1$.

Then, for the relationship between the additively separable function and the multiplicatively separable function, we can have two theorems, which are Theorem 2 and Theorem 3 as follows.

Theorem 2: Every additively separable function can be transformed into a multiplicatively separable function.

Proof: Given a partially additively separable function $f(\mathbf{x})$ as defined in Definition 2, letting $g(\mathbf{x}) = e^{f(\mathbf{x})}$, then $g(\mathbf{x})$ can be rewritten as

$$\begin{aligned} g(\mathbf{x}) &= e^{f(\mathbf{x})} \\ &= e^{\sum_{i=1}^{k} f_i(\mathbf{x}_i)} \\ &= \prod_{i=1}^{k} e^{f_i(\mathbf{x}_i)},\ 1 < k \le D. \end{aligned} \tag{10}$$

According to Definition 3 and (10), $g(\mathbf{x})$ is multiplicatively separable and, therefore, the proof is finished.

Theorem 3: Every multiplicatively separable function, where the minimum value is larger than 0, can be transformed into an additively separable function.

Proof: Given a multiplicatively separable function $g(\mathbf{x})$ as defined in Definition 3 and with the minimum value larger than 0, let $f(\mathbf{x}) = \ln g(\mathbf{x})$, then, $f(\mathbf{x})$ can be rewritten as

$$\begin{aligned} f(\mathbf{x}) &= \ln g(\mathbf{x}) \\ &= \ln \prod_{i=1}^{k} g_i(\mathbf{x}_i) \\ &= \sum_{i=1}^{k} \ln g_i(\mathbf{x}_i),\ 1 < k \le D. \end{aligned} \tag{11}$$

According to Definition 2, $f(\mathbf{x})$ is additively separable and, therefore, the proof is finished. That is, the logarithmic operation can transform a multiplicatively separable function into an additively separable function. It may be argued that some $g_i(\mathbf{x}_i)$ may be negative in (11). However, as $g(\mathbf{x})$ is positive, the number of negative subfunctions [e.g., $g_i(\mathbf{x}_i)$], must be even, and we can always find the same number of positive subfunctions to substitute the negative subfunctions. For example, for any subfunction $g_i(\mathbf{x}_i)$, where $g_i(\mathbf{x}_i) \neq 0$ because the minimum of $g(\mathbf{x})$ is larger than 0, we can use a new function $\text{newg}_i(\mathbf{x}_i)$ instead of $g_i(\mathbf{x}_i)$, as

$$\text{newg}_i(\mathbf{x}_i) = \begin{cases} g_i(\mathbf{x}_i), & \text{if } g_i(\mathbf{x}_i) > 0 \\ -g_i(\mathbf{x}_i), & \text{otherwise} \end{cases} \tag{12}$$

where $\text{newg}_i(\mathbf{x}_i)$ is guaranteed to be positive.

In addition, to make Theorem 3 easier to understand and to illustrate what it can do, we take the problem in (1) again as an example here. By using the logarithmic operation like (11), we can have

$$\begin{aligned} f_{\text{example}}(x_1, x_2) &= \ln g_{\text{example}}(x_1, x_2) \\ &= \ln(2x_1x_2 + 5x_1 + 14x_2 + 35) \\ &= \ln[(x_1 + 7)(2x_2 + 5)] \\ &= \ln(x_1 + 7) + \ln(2x_2 + 5) \end{aligned} \tag{13}$$

where x_1 and x_2 belong to $[-5, 5]$ and $[-2, 2]$, respectively, as defined in (1).

According to Definition 2, f_{example} is actually an additively separable function, which can be decomposed by DG-based methods. In other words, Theorem 3 shows that we can use a novel method modified/enhanced from the additive difference to detect the separable structure in multiplicatively separable functions. To be more specific, with Theorem 1 and Theorem 3, we can determine the multiplicatively separable variables by checking whether the following equation holds:

$$\begin{aligned}&\ln(f(\mathbf{x}))\big|_{x_i=a,x_j=b} - \ln(f(\mathbf{x}))\big|_{x_i=c,x_j=b} \\ &\quad\neq \ln(f(\mathbf{x}))\big|_{x_i=a,x_j=d} - \ln(f(\mathbf{x}))\big|_{x_i=c,x_j=d} \end{aligned} \tag{14}$$

where $f(\mathbf{x})$ should be positive. In practical implementations, we can use the (15) instead of (14)

$$\begin{aligned}&\left|\left[\ln(f(\mathbf{x}))\big|_{x_i=a,x_j=b} - \ln(f(\mathbf{x}))\big|_{x_i=c,x_j=b}\right]\right. \\ &\quad\left. - \left[\ln(f(\mathbf{x}))\big|_{x_i=a,x_j=d} - \ln(f(\mathbf{x}))\big|_{x_i=c,x_j=d}\right]\right| \leq \varepsilon_{\text{multi}} \end{aligned} \tag{15}$$

where $\varepsilon_{\text{multi}}$ is the acceptance threshold for detecting multiplicatively separable variables.

Based on the above, we can propose the DDG method to obtain better decomposition for both additively and multiplicatively separable problems, which is described in the following contents.

B. DDG

As mentioned before, the idea of DDG is to detect whether variables are additively or multiplicatively separable and then select the best way to partition them into different groups accordingly. The pseudocode of DDG is presented as Algorithm 1. Note that the only difference between DDG and DG lies in lines 20–27, which aim to detect not only additively but also multiplicatively separable variables. That is, DG only uses $\Delta_{\text{addi}} > \epsilon_{\text{addi}}$ while DDG uses both $\Delta_{\text{addi}} > \epsilon_{\text{addi}}$ and $\Delta_{\text{multi}} > \epsilon_{\text{multi}}$ as conditions in the If-statement in line 25 of Algorithm 1. Therefore, the DDG is also as easy to use as DG because it does not consume more FEs than DG and only adds a slight computational burden as line 21 of Algorithm 1.

In general, Algorithm 1 adopts a sequential fashion to detect the possible separable structure between each variable and the other variables. To be specific, Algorithm 1 will use a nested loop, that is, lines 7–35, to check whether each dimension variable is separable from other variables and then group the nonseparable variables together. The novel and key operations of Algorithm 1 lie in lines 20–27, which compute the dual differences for detecting additively or multiplicatively separable structures of variables.

For the additively separable structure, we can calculate the additive difference, that is, Δ_{addi}, to detect interactions as

$$\begin{aligned}\Delta_{\text{addi}} = &\left|\left[f(\mathbf{x})\big|_{x_i=a,x_j=b} - f(\mathbf{x})\big|_{x_i=c,x_j=b}\right]\right. \\ &\left. - \left[f(\mathbf{x})\big|_{x_i=a,x_j=d} - f(\mathbf{x})\big|_{x_i=c,x_j=d}\right]\right| \end{aligned} \tag{16}$$

where a, b, c, and d are real numbers within the search domain, and $\mathbf{x} = (x_1, x_2, \ldots, x_D)$ is a D-dimensional solution. In Algorithm 1, as suggested in the literature [27], a, b, c, and d are set as the lower bound of x_i, the lower bound of x_j, the upper bound of x_i, and the center of the search domain of x_j, respectively. It should be noted that these four variables can also be set with other values, and the settings adopted in this article are just conventional choices in the literature. For simplicity, $\mathbf{x}$ with $x_i = a$ and $x_j = b$, $\mathbf{x}$ with $x_i = c$ and $x_j = b$, $\mathbf{x}$ with $x_i = a$ and $x_j = d$, and $\mathbf{x}$ with $x_i = c$ and $x_j = d$ are denoted as $\mathbf{x}_1$, $\mathbf{x}_2$, $\mathbf{x}_3$, and $\mathbf{x}_4$ respectively, in Algorithm 1.

For the multiplicatively separable problem, as seen in line 21, the algorithm first uses the logarithmic function $\ln(x)$ to transform the original function into an additively separable function (similar to how (11) works) and then computes the

Algorithm 1: Dual Differential Grouping

Input: *func*-the objective function;
D-the problem dimension;
lb-the *D*-dimensional array of lower bounds;
ub-the *D*-dimensional array of upper bounds.
ϵ_{addi}-the threshold for additively separable detection.
ϵ_{multi}-the threshold for multiplicatively separable detection.
Output: *allgroups*-the identified groups containing the index of variables.

1 **Begin**
2 $dims \leftarrow \{1, 2, \ldots, D\}$; // the index set of undetected variables
3 $seps \leftarrow \{\}$; // initialize the index set for separable variables
4 $allgroups \leftarrow \{\}$; // initialize the set for all groups
5 $\mathbf{x}_1 \leftarrow lb$; //all variables in $\mathbf{x}_1$ is set as their lower bounds
6 $fit_1 \leftarrow func\ (\mathbf{x}_1)$;
7 **For** $i \in dims$ **Do**
8 $tempgroup \leftarrow \{i\}$; // the group includes variable index i
9 $\mathbf{x}_2 \leftarrow \mathbf{x}_1$;
10 $\mathbf{x}_{2,i} \leftarrow ub_i$; //variable i in $\mathbf{x}_2$ is set as its upper bound
11 $fit_2 \leftarrow func(\mathbf{x}_2)$;
12 **For** $j \in dims$ *and* $i \neq j$ **Do**
13 $\mathbf{x}_3 \leftarrow \mathbf{x}_1$; // variable i in $\mathbf{x}_3$ is its lower bound
14 $\mathbf{x}_{3,j} \leftarrow (lb_j + ub_j)/2$; //variable j in $\mathbf{x}_3$ is its domain center
15 $fit_3 \leftarrow func\ (\mathbf{x}_3)$;
16 $\mathbf{x}_4 \leftarrow \mathbf{x}_2$; // variable i in $\mathbf{x}_4$ is its upper bound
17 $\mathbf{x}_{4,j} \leftarrow (lb_j + ub_j)/2$; //variable j in $\mathbf{x}_4$ is its domain center
18 $fit_4 \leftarrow func\ (\mathbf{x}_4)$;
19 $\Delta_{addi} \leftarrow |(fit_1 - fit_2) - (fit_3 - fit_4)|$; // see Eq.(16)
20 **If** fit_1, fit_2, fit_3, and fit_4 are all positive **Then**
21 $\Delta_{multi} \leftarrow |(\ln(fit_1) - \ln(fit_2)) - (\ln(fit_3) - \ln(fit_4))|$;//Eq.(17)
22 **Else**
23 $\Delta_{multi} \leftarrow 1 \times 10^5$; //a large enough value to make $\Delta_{multi} > \epsilon_{multi}$
24 **End if**
25 **If** $\Delta_{addi} > \epsilon_{addi}$ *and* $\Delta_{multi} > \epsilon_{multi}$ **Then** // non-separable
26 $tempgroup \leftarrow tempgroup \cup j$; // j should be grouped with i
27 **End if**
28 **End for**
29 **If** length(*tempgroup*)==1 **Then** // variable i is separable
30 $seps \leftarrow seps \cup tempgroup$;
31 **Else** // variable i is non-separable with the variables in *tempgroup*
32 $allgroups \leftarrow allgroups \cup \{tempgroup\}$;
33 **End if**
34 $dims \leftarrow dims - tempgroup$; // remove the detected variable indexes
35 **End for**
36 $allgroups \leftarrow allgroups \cup \{seps\}$;
37 **End**

multiplicative difference Δ_{multi} for interaction detections as

$$\Delta_{\text{multi}} = \left|\left[\ln\left(f(\mathbf{x})\big|_{x_i=a,x_j=b}\right) - \ln\left(f(\mathbf{x})\big|_{x_i=c,x_j=b}\right)\right] - \left[\ln\left(f(\mathbf{x})\big|_{x_i=a,x_j=d}\right) - \ln\left(f(\mathbf{x})\big|_{x_i=c,x_j=d}\right)\right]\right|. \quad (17)$$

It should be noted that if $\ln(f(\mathbf{x}))$ encounters calculation errors due to the nonpositive value of $f(\mathbf{x})$, the algorithm will directly set Δ_{multi} with a value larger than $\varepsilon_{\text{multi}}$, as shown in line 23 of Algorithm 1. This can ensure that the following procedure will not mistake the corresponding variables as multiplicatively separable due to the calculation error.

After computing Δ_{addi} and Δ_{multi}, if they are both larger than their corresponding threshold ε_{addi} and $\varepsilon_{\text{multi}}$, respectively, then the variables x_i and x_j are neither additively separable nor multiplicatively separable. In this situation, x_i and x_j are considered as nonseparable, and x_j will be grouped into the same group of x_i, as shown in line 26. Otherwise, x_i and x_j are separable and will not be grouped into the same group.

The detection operations, that is, lines 12–28, will be repeated until the interactions between x_i and all the remaining variables are checked. After detecting the interaction between x_i and all the remaining variables, DDG checks whether the temporal group of x_i (i.e., *tempgroup*) has other variables. If no, then x_i is a fully separable variable, and its index will be stored in the fully separable group *seps*. Otherwise, the whole group *tempgroup* is stored as a new index set in *allgroups*. After this, DDG removes the index of detected variables (i.e., those in *tempgroup*) from the index set *dims*, and then selects an index of the rest of the variables in *dims* as the next variable for checking its interactions with the other rest variables in *dims*.

The above procedures will repeat until there is no variable index left in *dims*. Then, the *seps*, which includes the indices of all detected fully separable variables, will also be stored as an index set in *allgroups*. Finally, the DDG will output the *allgroups*, which contain the decomposition results that include both nonseparable and separable groups.

C. Complete DDG-Based CC Algorithm

As the decomposition method aims to divide the LSOP for better optimization, this part describes how the proposed DDG can be used in a CC framework for solving the LSOP.

Fig. 2 presents the flowchart of the complete algorithm framework, and Algorithm 2 shows the pseudocode. Note that the main novelty of Algorithm 2 lies in that the decomposition stage uses DDG to decompose the problems, as shown in line 3. Algorithm 2 is developed by adopting the proposed DDG method in the classical CC framework [26], [27]. Although many CC frameworks have been proposed [48]–[50], the focus of this article is on the decomposition method but not on the CC framework. Therefore, without loss of generality, the most classical and widely used CC framework (i.e., the one used in [26], [27]) is adopted in this article as an example to develop the DDG-based CC algorithm. Similar to other existing decomposition-based algorithms [27]–[29], Algorithm 2 mainly has three procedures: the decomposition stage, the optimization stage, and the combination stage. The

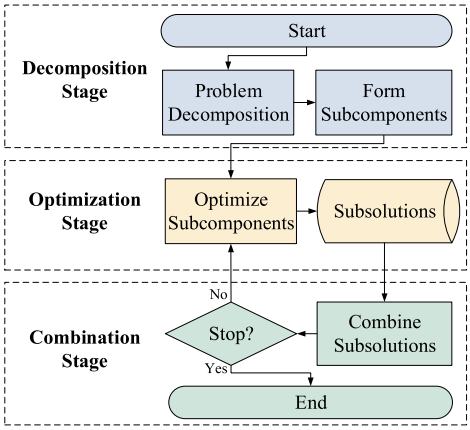


Fig. 2. Flowchart of the complete DDG-based algorithm.

Algorithm 2: Complete DDG-based Algorithm

Input: *func*-the objective function;
D-the problem dimension;
lbounds-the array of lower bounds;
ubounds-the array of upper bounds;
NP-the number of individuals in population;
MaxFEs-the maximum number of available FEs.
Output: sol_{best} -the best solution.

```
1  Begin
2      //decomposition stage using Algorithm 1
3      Obtain separable groups allgroups by DDG;
4      n_group ← the total number of groupings in allgroups;
5      FEs ← the consumed FEs in decomposition stage;
6      //optimization stage
7      Initialize the population pop;
8      fitness ← func(pop); // evaluate the fitness of individuals
9      sol_best ← the best solution in pop; //record the best solution so far
10     FEs ←FEs + NP; // update the consumed FEs
11     While FEs< MaxFEs Do // loop until the FEs are run out
12         For i ← 1 to n_group Do // optimize each decomposed group
13             indicies ← allgroups[i]; // the variable indices in ith group
14             subpop_i ← pop[:, indicies]; //form sub-populations
15             Update subpop_i, newfitness_i by optimizing subpop_i;
16             FEs ← FEs + NP; // update the consumed FEs
17             //combination stage to form updated complete solutions
18             pop[:, indicies] ← subpop; //update the population
19             fitness ← newfitness_i; //update the corresponding fitness
20             sol_best ← the best solution in pop; //update the best solution
21         End for
22     End while
23 End
```

decomposition stage employs the proposed DDG to partition the problem, where the details can be found in Algorithm 1. In the optimization stage, different subpopulation are formed according to the results from the decomposition stage. Then, each subpopulation will be iteratively optimized for the corresponding subproblem in a round-robin fashion by an optimizer, where the optimizer can be any EC algorithm [50]–[53]. In this article, the self-adaptive differential evolution with neighborhood search (SaNSDE) [53], which is also the classical optimizer in decomposition-based studies [28], is adopted to work together with DDG to develop the complete CC algorithm. After the optimization stage, the combination stage forms the complete solutions with updated subsolutions, as

shown in line 18 of Algorithm 2. The above optimization and combination stages will repeat until the *MaxFEs* are totally consumed. Finally, Algorithm 2 outputs the complete solution and terminates.

D. Analysis of Time Complexity

As the FE is the most time-consuming operation in decomposition methods, this part discusses the time complexity of DDG by analyzing the upper bound for the total number of FEs required by DDG. Without loss of generality, we assume that the problem is a D-dimensional problem with k separable subcomponents, where the subcomponents are nonoverlapped and each of them contains $m = D/k$ variables. In addition, without loss of generality, we assume that variables $x_{(i-1)\cdot m+1}$ to $x_{i\cdot m}$ belong to the ith subcomponent (i.e., group). As seen in Algorithm 1, only four lines will consume the FEs, which are lines 15 and 18 in the innermost For loop, line 11 in the outer For loop, and line 6 outside the loop. Therefore, the total time for executing these lines will be analyzed one by one in the following contents.

First, we analyze the total time for executing lines 15 and 18 in Algorithm 1, where both of them consume one FE. Instead of directly obtaining the total time for executing them, we can count how many times the algorithm will compute the differences (i.e., lines 19–24 in Algorithm 1) because each difference calculation exactly corresponds to one execution of both lines 15 and 18. For the sake of simplicity, we denote the upper bound as S for how many times the algorithm will compute the differences. According to the above assumptions andAlgorithm 1, after each loop for calculating the differences between a variable x_i and all the remaining variables in *dims*, m variables (including x_i) will be removed from *dims*. For example, for finding the first subcomponents of interacted variables, the algorithm will calculate the difference for D–1 times (between x_1 and the remaining D–1 variables) and find that variables x_2 to x_m interact with x_1. Then, the m variables $x_1, x_2, \ldots,$ and x_m, (i.e., the first one of the k nonoverlapped separable subcomponents), will be removed from *dims*. Similarly, for finding the second subcomponents, the algorithm will only calculate the difference for D–m–1 times (between x_{m+1} and the remaining D–m–1 variables); and for the ith subcomponent, only calculate the difference for $D-(i-1)\cdot m-1$ times (between $x_{(i-1)\cdot m+1}$ and the remaining $D-(i-1)\cdot m-1$ variables). As there are k subcomponents in total, the upper bound S can be calculated as

$$\begin{aligned} S &= (D-1)+(D-m-1)+\cdots+(D-(k-1)\times m-1) \\ &= (D-1)+(D-m-1)+\cdots+(m-1) \\ &= \frac{k}{2}(D+m-2) \\ &= \frac{D}{2m}(D+m-2) \end{aligned} \tag{18}$$

where D will not be less than m. Then, the total number for executing lines 15 and 18 is $2S$.

Second, we analyze the total time for executing line 11 in Algorithm 1, which also requires one corresponding FE. Every time line 11 is executed, Algorithm 1 will determine a subcomponent (refer to lines 12–34 in Algorithm 1). As there are k different subcomponents to be identified, the total time for executing line 11 is k, and the total needed number of FEs is also k.

Third, as line 6 of Algorithm 1 costs one FE and is outside the loop, the total time for executing line 6 is 1, which corresponds to one FE.

Therefore, based on the above, the total number of FEs required by DDG is $2S+k+1$. That is, the time complexity of DDG with respect to the needed number of FEs is

$$O(\text{FEs}) = O(2S+k+1) = O\left(\frac{D^2}{m}\right) \tag{19}$$

where m and k are not larger than D. Note that the time complexity of DDG, that is, (19), is the same as that of DG [27]. Therefore, the DDG is not time expensive when compared to other DG-based methods.

IV. Experimental Studies

A. Benchmark Functions and Comparison Methods

To evaluate the proposed DDG, 30 LSOPs are adopted as the test suite in this article, which are all minimization problems. In these LSOPs, 15 of them are the original test functions from the widely used IEEE CEC 2013 benchmark set for the LSGO competition [42], including additively separable functions and fully nonseparable functions. In addition, since there are no multiplicatively separable functions in the original IEEE CEC 2013 benchmark set, 15 multiplicatively separable functions are construed herein based on the functions in this benchmark set.

The adopted 30 test functions are shown in Table I. In Table I, $f_1, f_2, \ldots,$ and f_{15} are the corresponding 15 original functions in the CEC 2013 benchmark, while the remaining 15 multiplicatively separable functions are generated based on additively separable functions f_1–f_3 and fully nonseparable functions f_{13}-f_{15}. Specifically, T16–T30 are generated by multiplying two of the 6 functions, that is, f_1–f_3 and f_{13}–f_{15}. For simplicity, we use the notation "$f_a \otimes f_b$" to denote how T16–T30 are constructed. For example, if both f_a and f_b are 1000-D functions, $f_a \otimes f_b$ will result in a 2000-D problem as

$$f_a \otimes f_b(\mathbf{x}_{1:2000}) = f_a(\mathbf{x}_{1:1000}) \times f_b(\mathbf{x}_{1001:2000}) \tag{20}$$

where $\mathbf{x}_{1:2000}$ means the 2000-D variable vector, and $\mathbf{x}_{1:1000}$ and $\mathbf{x}_{1001:2000}$ are the first 1000 dimension values and the last 1000 dimension values of $\mathbf{x}$, respectively. Furthermore, as the optimal minimum value of each function in the IEEE CEC 2013 benchmark is 0, the test function T generated by (20) can obtain the optimum value (i.e., 0) as long as one of the f_a and f_b (or both) have been optimized to their optimal value 0. Therefore, if the test functions generated by (20) can be decomposed correctly, the optimization difficulties will greatly decrease. This is very suitable for evaluating the effectiveness of different decomposition methods on multiplicatively separable functions.

To further clarify the function characteristics in Table I, the symbols "A," "O," "N," and "M" are used to represent

TABLE I
CHARACTERISTICS OF THE 30 TEST FUNCTIONS

Test Functions	Construction	Function Type	Dimension
T01	f_1	A	1000
T02	f_2	A	1000
T03	f_3	A	1000
T04	f_4	A	1000
T05	f_5	A	1000
T06	f_6	A	1000
T07	f_7	A	1000
T08	f_8	A	1000
T09	f_9	A	1000
T10	f_{10}	A	1000
T11	f_{11}	A	1000
T12	f_{12}	O	1000
T13	f_{13}	O	905
T14	f_{14}	O	905
T15	f_{15}	N	1000
T16	$f_1 \otimes f_2$	M, A⊗A	2000
T17	$f_1 \otimes f_3$	M, A⊗A	2000
T18	$f_2 \otimes f_3$	M, A⊗A	2000
T19	$f_1 \otimes f_{13}$	M, A⊗O	1000+905
T20	$f_1 \otimes f_{14}$	M, A⊗O	1000+905
T21	$f_1 \otimes f_{15}$	M, A⊗N	1000+1000
T22	$f_2 \otimes f_{13}$	M, A⊗O	1000+905
T23	$f_2 \otimes f_{14}$	M, A⊗O	1000+905
T24	$f_2 \otimes f_{15}$	M, A⊗N	1000+1000
T25	$f_3 \otimes f_{13}$	M, A⊗O	1000+905
T26	$f_3 \otimes f_{14}$	M, A⊗O	1000+905
T27	$f_3 \otimes f_{15}$	M, A⊗N	1000+1000
T28	$f_{13} \otimes f_{14}$	M, O⊗O	905+905
T29	$f_{13} \otimes f_{15}$	M, O⊗N	905+1000
T30	$f_{14} \otimes f_{15}$	M, O⊗N	905+1000

the function type as "additively separable," "overlap," "non-separable," and "multiplicatively separable," respectively. As shown in Table I, the 30 test functions have various characteristics and, therefore, they can provide in-depth observations about how the proposed DDG may behave on different kinds of problems.

To compare the decomposition ability of DDG, some popular and state-of-the-art decomposition methods are adopted for comparisons of decomposition accuracies. These methods are DG [27], DG2 [28], XDG [29], and GDG [30], as briefly described in the related work in Section II-B. In addition, these decomposition methods are implemented based on their open available source code and are adopted in Algorithm 2 to replace the DDG to develop their corresponding versions of the CC algorithm. That is, all different decomposition methods work with the same optimizer, so as to achieve a fair comparison.

B. Experimental Settings and Evaluation Metrics

In the experiment, the parameters of all decomposition methods are configured according to their original papers. In DDG, the value of ϵ_{addi} for detecting additively separable variables is configured as 10^{-3}, which is recommended in the literature for detecting additively separable problems [28]. For the ϵ_{multi}, the value is set as $\epsilon_{multi} = 10^{-8}$, where the corresponding parameter study will be given later in Section IV-H. Moreover, to obtain a fair optimization comparison, the CC algorithms using different decomposition methods will adopt the same optimizer SaNSDE [28] as described in Section III-C. The population size of SaNSDE is set as 50, as suggested in [27], [28]. Note that the population size in the original SaNSDE [53] is 100 and the population size may influence the algorithm performance. However, the influence of population size is not the focus of this article. Therefore, the population size is set as the frequently used value (i.e., 50) in the LSOP literature [27], [28]. In addition, the maximum number of available FEs, that is, *MaxFEs*, is 3×10^6 for all algorithms on each problem of T01–T15 according to the literature [42], and is 6×10^6 for all algorithms on each problem of T16–T30 because each of them is construed by two problems in T01–T15.

To compare the decomposition ability of different decomposition methods, three evaluation metrics proposed in [30] are adopted in this article. The definitions of these metrics are as follows:

$$\rho_{\text{overall}} = \frac{\sum_{i=1}^{D}\sum_{j=1,j\neq i}^{D}(\mathbf{1}_{D\times D} - |\Theta - \Theta_{\text{ideal}}|)_{i,j}}{D(D-1)} \times 100\% \tag{21}$$

$$\rho_{\text{sep}} = \frac{\sum_{i=1}^{D}\sum_{j=1,j\neq i}^{D}((\mathbf{1}_{D\times D} - \Theta) \circ (\mathbf{1}_{D\times D} - \Theta_{\text{ideal}}))_{i,j}}{\sum_{i=1}^{D}\sum_{j=1,j\neq i}^{D}(\mathbf{1}_{D\times D} - \Theta_{\text{ideal}})_{i,j}} \times 100\% \tag{22}$$

$$\rho_{\text{inter}} = \frac{\sum_{i=1}^{D}\sum_{j=1,j\neq i}^{D}(\Theta \circ \Theta_{\text{ideal}})_{i,j}}{\sum_{i=1}^{D}\sum_{j=1,j\neq i}^{D}(\Theta_{\text{ideal}})_{i,j}} \times 100\% \tag{23}$$

where D is the problem dimension, Θ is the interaction matrix obtained by the decomposition method, $(\Theta)_{i,j}$ equals 1 if variable i and variable j have interaction and 0 otherwise, Θ_{ideal} is the ideal interaction matrix for a problem, and operator "$\circ$" is the entrywise product of two matrices. Based on their definitions, ρ_{overall} measures the overall accuracy of the decomposition method, ρ_{sep} only measures the accuracy of separable variable detection, and ρ_{inter} only measures the accuracy of interaction detection [30]. Note that T01–T15 are from existing benchmark problems and have their corresponding ideal interaction matrix [28], denoted as $\Theta_{\text{ideal1}}, \ldots,$ and Θ_{ideal15}, respectively. Based on this, the interaction matrix of T16–T30 can be constituted by $\Theta_{\text{ideal1}}, \ldots,$ and Θ_{ideal15}. For example, T16 is the multiplication of T01 and T02 and, therefore, its interaction matrix Θ_{ideal16} is

$$\Theta_{\text{ideal16}} = \begin{bmatrix} \Theta_{\text{ideal1}} & \mathbf{0} \\ \mathbf{0} & \Theta_{\text{ideal2}} \end{bmatrix}. \tag{24}$$

To reduce the statistical error, 25 independent runs of each CC algorithm are carried out on each problem, and the average results are used for comparison. In addition, the Wilcoxon rank-sum test with a significance level $\alpha = 0.05$ is adopted to statistically compare the optimization results, where the symbols "+," "≈"and "−" are used to show that the proposed algorithm is significantly better than, similar to, or significantly worse than the compared algorithm, respectively.

C. Comparisons on Decomposition Accuracy

The decomposition accuracy of DDG and other decomposition methods are compared in terms of the three evaluation metrics. The results are provided in Table II for the first metric and Table S.I in the supplementary material for the second and third metrics. The best results are marked in **boldface**.

TABLE II
OVERALL DECOMPOSITION ACCURACY OF DDG, DG, DG2, XDG, AND GDG ON 30 LARGE-SCALE OPTIMIZATION PROBLEMS

Function	DDG	DG	DG2	XDG	GDG
T01	**100.00%**	**100.00%**	**100.00%**	**100.00%**	**100.00%**
T02	**100.00%**	**100.00%**	**100.00%**	**100.00%**	**100.00%**
T03	**100.00%**	**100.00%**	0.00%	**100.00%**	**100.00%**
T04	98.00%	79.11%	**100.00%**	98.26%	98.26%
T05	98.04%	98.04%	**100.00%**	98.05%	98.05%
T06	97.32%	97.32%	51.30%	**97.48%**	**97.48%**
T07	96.04%	71.14%	**100.00%**	97.18%	97.18%
T08	93.15%	92.37%	**98.01%**	93.17%	93.17%
T09	92.43%	92.36%	**100.00%**	93.03%	93.03%
T10	93.07%	92.37%	**99.99%**	93.17%	93.17%
T11	89.57%	77.61%	**99.99%**	92.93%	92.93%
T12	85.15%	84.77%	**100.00%**	99.80%	99.80%
T13	78.23%	74.77%	**100.00%**	91.04%	91.04%
T14	90.31%	90.41%	**99.99%**	91.00%	91.00%
T15	**100.00%**	**100.00%**	**100.00%**	**100.00%**	**100.00%**
T16	**100.00%**	54.76%	0.00%	49.97%	50.03%
T17	**100.00%**	60.11%	25.10%	49.97%	50.03%
T18	**100.00%**	**100.00%**	24.99%	49.98%	50.03%
T19	**79.68%**	74.02%	26.04%	29.40%	72.68%
T20	71.93%	**73.99%**	26.41%	29.40%	73.68%
T21	**75.01%**	56.04%	24.99%	49.97%	50.04%
T22	**78.01%**	68.98%	22.70%	29.40%	72.22%
T23	**78.48%**	68.42%	24.13%	29.40%	73.63%
T24	**74.96%**	53.83%	24.99%	49.97%	50.06%
T25	**75.54%**	29.49%	51.39%	29.40%	72.53%
T26	**75.34%**	31.61%	52.19%	29.40%	73.63%
T27	**74.96%**	49.97%	49.97%	49.97%	50.04%
T28	87.97%	28.69%	54.58%	4.12%	**93.10%**
T29	47.26%	29.24%	50.16%	29.40%	**70.56%**
T30	47.26%	39.32%	51.36%	29.40%	**70.58%**
# of best	**15**	6	13	5	8

Moreover, to give a clearer understanding of DDG, the detailed grouping results of DDG on T04, T13, T19, and T29 are shown in Tables S.II–S.V in the supplementary material, which are representative additively separable function, overlapping function, multiplicatively separable function generated by fully separable function and overlapping function, and multiplicatively separable function generated by overlapping function and nonseparable function, respectively.

First, the decomposition results show that the DDG is competitive with other decomposition methods on the additively separable problem. As shown in Table II, the DDG, in term of the overall decomposition accuracy, can generate results that are competitive with DG on additively separable functions and correctly decompose the 3 fully additively separable functions (i.e., T01–T03). This verifies the ability of DDG to decompose the additively separable problem.

Second, the results in Table II show that the DDG can perform significantly better than other decomposition methods on the multiplicatively separable problem. It can be seen that on multiplicatively separable test functions (e.g., T16–T27), the decomposition accuracy of DDG is more promising than those obtained by other decomposition methods. Moreover, DDG can achieve significantly better decomposition accuracy than both DG2 and XDG on all the 15 generated multiplicatively separable functions, that is, T16–T30. In addition, on T16–T27, the DDG can perform significantly better than DG and GDG on 11 and 11 problems, respectively. For T28–T30, the decomposition accuracy of DDG is not as good as that of GDG. This may be due to that T28–T30 are all constructed by two nonseparable problems or overlapping problems. In such situations, although the problem can be decomposed correctly into two nonseparable or overlapping problems by checking the multiplicative difference, the additive difference cannot work well to decompose the nonseparable or overlapping problems correctly, for example, the original DG also works poorly on T28–T30. However, when compared to DG, the DDG actually works better on T28–T30, which suggests the effectiveness of multiplicative difference in DDG.

Third, Table S.I in the supplementary material shows that the main advantage of DDG is the detection of multiplicatively separable variables. As shown on the left side of Table S.I in the supplementary material, the DDG obtains 100% accuracy on detecting the separable variables in T16–T27, while DG, DG2, and XDG only have much lower accuracy. For the GDG, although it can also obtain high accuracy on detecting the separable variables of T16–T30, similar to DDG, it can only have low decomposition accuracy for the interacted variables (as shown on the right side of Table S.I in the supplementary material), and its overall detection accuracy therefore decreases. In addition, Table S.I in the supplementary material shows that on the multiplicatively separable problems T16–T30, the XDG has high accuracy on interacting variables but very poor accuracy on separable variables. This may be because the XDG fails to detect the separable structures in T16–T30 and mistakes many variables as interacting, resulting in high accuracy on the interacting variables but nearly zero accuracy on the separable variables. However, the DDG proposed in this article does not have this problem because it can detect the separable structure in not only additively separable problems but also multiplicatively separable problems.

In conclusion, the comparisons on decomposition results have shown the great effectiveness of DDG on both the additively and multiplicatively separable problem.

D. Comparisons on Optimization Results

To investigate the advantage of the DDG-based CC algorithm in optimizing LSOPs, this section compares the optimization results. The comparisons are made among the CC algorithms that use different decomposition methods, including DG, DG2, XDG, and GDG. Also, the random decomposition (RD) method is adopted to evaluate the effectiveness of DDG, where the RD groups variables randomly into five groups at the beginning of each run. For simplicity, the CC algorithm with the decomposition method X is denoted as "CC-X" in the following contents.

The comparison results are provided in Table III and the detailed results are given in Table S.VI of the supplementary material. According to the Wilcoxon rank-sum test, CC-DDG significantly outperforms CC-DG, CC-DG2, CC-XDG, CC-GDG, and CC-RD on 15, 16, 16, 16, and 15 problems, respectively. Furthermore, CC-DDG can obtain the best results on most multiplicatively separable problems (i.e., T21–T26 and T28–T30), showing its strong effectiveness in solving large-scale multiplicatively separable problems.

TABLE III
COMPARISON RESULTS OF CC ALGORITHM WITH DIFFERENT DECOMPOSITION METHODS ON THE 30 LARGE-SCALE OPTIMIZATION PROBLEMS

Statistical term	CC-DDG	CC-DG	CC-DG2	CC-XDG	CC-GDG	CC-RD
+/≈/-	NA	**15**/15/0	**16**/8/6	**16**/7/7	**16**/6/8	**15**/2/13

The symbols "+", "≈"and "-" are used to show that the CC-DDG is significantly better than, similar to, or significantly worse than the compared algorithm.

TABLE IV
COMPARISONS OF OVERALL DECOMPOSITION ACCURACY AMONG RDDG3, DDG, AND RDG3 AND OPTIMIZATION RESULTS AMONG CBCC-RDDG3, CBCC-DDG, AND CBCC-RDG3 ON 30 LARGE-SCALE OPTIMIZATION PROBLEMS

Statistical term	Overall Decomposition Accuracy			Statistical term	Optimization result with CBCC and CMAES		
	RDDG3	DDG	RDG3		CBCC-RDDG3	CBCC-DDG	CBCC-RDG3
Number of best results on T01-T15	8	11	6	+/≈/- on T01-T15	NA	**12**/0/3	**6**/7/2
Number of best results on T16-T30	9	10	0	+/≈/- on T16-T30	NA	**10**/1/4	**8**/3/4

In addition, on additively separable functions (T01–T11) and nonseparable functions (T12–T15), CC-DDG can also perform similarly to CC-DG and obtain the best results on T03, T06, and T10 among the six CC algorithms with different decomposition methods, which suggests that CC-DDG can also have competitive performance on additively separable LSOPs. In conclusion, CC-DDG is effective for optimizing both additively and multiplicatively separable LSOPs.

E. Comparisons With the Champion Algorithm

This part compares the DDG-based algorithm with the champion algorithm on IEEE CEC 2019 LSGO competitions, that is, the RDG3-based algorithm [32]. As the champion algorithm adopts the contribution-based CC framework (CBCC) as its optimization framework [32] and the covariance matrix adaptation with evolutionary strategy (CMA-ES) [54] as its optimizer, the DDG-based algorithm is also integrated with CCBC and CMA-ES for a fair comparison. Moreover, as the RDG3 is an enhanced extension of DG (i.e., the 3rd version of *recursive* DG) for solving overlapping problems, we should also extend the DDG with the 3rd version of *recursive* strategy to obtain the RDDG3 for a fair comparison. The extension is very easy by replacing DG with DDG, where the modification of RDDG3 over RDG3 is given in Algorithm S.1 (lines 9–15) of the supplementary material. This way, the RDDG3 is expected to be suitable for additively separable, multiplicatively separable, and overlapping problems. The three algorithms are called CBCC-RDDG3, CCBC-DDG, and CCBC-RDG3, where the settings of CBCC and CMA-ES in the three algorithms are configured the same according to the original paper of CCBC-RDG3 [32].

The comparison results of grouping accuracy among RDDG3, DDG, and RDG3 are provided on the left side of Table IV, where the corresponding detailed results can be seen in Table S.VII of the supplementary material. As shown in Table IV, the RDDG3 and DDG can obtain better grouping accuracy than the RDG3, especially on the 15 multiplicatively separable problems (i.e., the T16 to T30). To be specific, on the problems T01–T15, the RDDG3, DDG, and RDG3 obtain the best grouping results on 8, 11, and 6 problems, respectively. On the 15 multiplicatively separable problems (i.e., the T16 to T30), the RDDG3 and DDG can obtain the best grouping results on nine and ten problems, respectively, while the RDG3 has the best results on none of these problems. Therefore, the grouping results have shown the effectiveness of the RDDG3 and DDG on problem decomposition.

As for the optimization results, the comparison results of CBCC-RDDG3, CBCC-DDG, and CBCC-RDG3 are given on the right side of Table IV, where the corresponding detailed results can be seen in Table S.VII of the supplementary material. The results show that, when combining the advantages of RDG3 and DDG together, the CBCC-RDDG3 can outperform the CBCC-RDG3 not only on the 15 test functions T01–T15 in the original IEEE CEC 2013 LSGO benchmark, but also on the 15 generated multiplicatively separable problems T16–T30. Specifically, according to the Wilcoxon rank-sum test, CBCC-RDDG3 performs significantly better than CBCC-RDG3 on 6 and 8 problems of T01–T15 and T16–T30, respectively. This means that the CBCC-RDDG3 has better overall performance than CBCC-RDG3 on both the additively and multiplicatively separable problems. Although CBCC-DDG may be not superior to CBCC-RDG3 on some overlapping problems, it performs better than CBCC-RDG3 on 8 out of the 15 multiplicatively separable problems (i.e., T16–T21, T24, and T29, more than a half). Moreover, on the 12 overlapping problems as indicated in Table I, CBCC-RDDG3 significantly outperforms CBCC-RDG3 on seven of them (i.e., T12–T14, T20, T22, T25, and T28, more than a half). These suggest that the DDG variants (especially the resulted RDDG3) can be potential for more kinds of problems including the additively separable, multiplicatively separable, and overlapping problem. Moreover, as the RDDG3 is the recursive version of the DDG method and the RDG3 is the recursive version of the DG method, the superiority of RDDG3 over RDG3 further indicate that the DDG (i.e., considering not only additively separable problems but also multiplicatively separable problems) is more promising than the DG (i.e., considering only additively separable problems), which is a major motivation and contribution of the proposed DDG.

F. Comparisons With the State-of-the-Art Nondecomposition Algorithms

Besides the above comparisons with the decomposition-based algorithms, this part further compares the DDG-based CC algorithm with some state-of-the-art nondecomposition algorithms. In particular, the three well-known algorithms in the literature, that is: 1) SHADE-ILS [55]; 2) MLSHADE-SPA [56]; and 3) MOS [57], are adopted in the comparison.

TABLE V
COMPARISONS OF STATISTICAL RESULTS BETWEEN DIFFERENT CC ALGORITHMS AND STATE-OF-THE-ART NONDECOMPOSITION ALGORITHMS

Statistical term	CC-DDG (SHADE-ILS as optimizer)	SHADE-ILS	MLSHADE-SPA	MOS	CC-DG (SHADE-ILS as optimizer)	CC-DG2 (SHADE-ILS as optimizer)	CC-XDG (SHADE-ILS as optimizer)	CC-GDG (SHADE-ILS as optimizer)
+/≈/-	NA	**15**/7/8	**19**/4/7	**20**/3/7	**13**/15/2	**13**/8/9	**15**/3/12	**16**/4/10

The SHADE-ILS and MLSHADE-SPA are the winner and runner-up in the IEEE 2018 LSGO competition, and MOS is the winner in the IEEE LSGO competitions from 2013 to 2018. In the experiment, the SHADE-ILS is also adopted as the optimizer in CC-DDG, so that we can see the advantage of DDG more clearly from the comparisons. Moreover, other decomposition-based CC algorithms with SHADE-ILS as the optimizer are also adopted in the comparison to investigate the effectiveness of CC-DDG.

The comparisons of statistical results are given in Table V, where the detailed results are provided in Table S.VIII of the supplementary material. As shown in Table V, the CC-DDG can have a better overall performance than the compared algorithms including decomposition-based and nondecomposition algorithms. Specifically, when compared to the state-of-the-art nondecomposition algorithms, CC-DDG performs significantly better than SHADE-ILS, MLSHADE-SPA, and MOS on 15, 19, 20 problems, similar on 7, 4, and 3 problems, while significantly worse only on 8, 7, and 7 problems, respectively. That is, the comparisons have shown the effectiveness of CC-DDG and its potential to integrate with state-of-the-art nondecomposition algorithms.

G. Component Analysis of DDG

To investigate the component contribution, the DDG is compared with its variants that do not use the additive difference Δ_{addi} or the multiplicative difference Δ_{multi} for detecting separable structures. For simplicity, these two variants are simply denoted as DDG-w/o-addi and DDG-w/o-multi, respectively. Note that the DDG-w/o-multi is the same as DG as it does not detect the multiplicatively separable structure and only detects the additively separable structure. The comparison results of the variants are provided in Table S.IX of the supplementary material.

First, in term of the overall accuracy, as shown in the left three columns of Table S.IX in the supplementary material, the DDG obtains the best results on more problems than both the DDG-w/o-addi and DDG-w/o-multi. Specifically, among the 30 tested problems, DDG produces the best results on 22 problems, while DDG-w/o-addi and DDG-w/o-multi only obtain the best results on 16 and 9 problems, respectively. More importantly, among the three methods, the DDG-w/o-addi performs worst on additively separable problems (e.g., T01–T11), while DDG-w/o-multi performs worst on multiplicatively separable problems (e.g., T16–T30), which suggests the significance of both the additive difference and multiplicative difference for detecting separable structures.

Second, considering the accuracy of separable variable detection, as shown in the middle three columns of Table S.IX in the supplementary material, the contributions of additive difference and multiplicative difference are more obvious. As seen, the DDG-w/o-addi method has very poor accuracy in finding the separable variables in the additively separable problem, such as the fully separable problems T02 and T03. In addition, the DDG-w/o-multi method works very badly on detecting the separable variables in the multiplicatively separable problem, such as T25 and T27. Therefore, the decomposition ability of DDG will deteriorate if the additive difference or multiplicative difference is not used.

Third, the accuracy of interaction detection, together with the overall accuracy, further reveals the poor decomposition ability of DDG-w/o-multi, showing the effectiveness of Δ_{multi}. For example, although DDG-w/o-multi can detect the interaction structure in T27, it fails to detect the separable variables in T27, especially multiplicatively separable variables, and mistakenly characterizes them as interacting. As a result, most elements in the Θ obtained by DDG-w/o-multi for T27 are 1. This leads to the high value of ρ_{inter} (i.e., only on interacted variables) but a very small value on ρ_{sep} (i.e., on separable variables) and ρ_{overall} (i.e., on overall performance). In other words, in terms of the detection accuracy on separable variables and on all variables, DDG-w/o-multi performs very poorly because it cannot detect the multiplicatively separable variables correctly. Similar results can be also seen on other multiplicatively separable problems, e.g., T20–T26, T28, and T29. These results further verify the great effectiveness and contribution of Δ_{multi} for decomposing multiplicatively separable problems.

From the above, it can be concluded that both the additive and multiplicative differences have their own contributions to the effectiveness of DDG, and removing any of them will decrease the decomposition performance of DDG.

H. Influences of the Threshold for Multiplicatively Separable Detection

This part studies the influence of the threshold value for detecting multiplicatively separable variables. For this aim, we compared the DDG that uses the original setting (i.e., $\epsilon_{\text{multi}} = 10^{-8}$) with its variants using $\epsilon_{\text{multi}} = 10^{-4}$, $\epsilon_{\text{multi}} = 10^{-12}$, and $\epsilon_{\text{multi}} = 10^{-16}$, which are denoted as DDG-8 (the original DDG), DDG-4, DDG-12, and DDG-16, respectively. The decomposition results are provided in Table S.X of the supplementary material.

First, Table S.X in the supplementary material shows that the threshold setting ϵ_{multi} for detecting multiplicatively separable variables does not have a significant influence on the decomposition accuracy for additively separable problems. It can be seen that on additively separable functions, for example, T01–T11, the DDG with different threshold settings obtains

similar decomposition accuracy. In other words, the settings of ϵ_{multi} will not affect the decomposition ability of DDG on additively separable problems and, therefore, the DDG with a proper ϵ_{multi} can be suitable for both additively separable problems and multiplicatively separable problems.

Second, Table S.X in the supplementary material shows that different multiplicatively separable problems favor different ϵ_{multi}. As shown in Table S.X of the supplementary material, $\epsilon_{\text{multi}} = 10^{-8}$ achieves higher accuracy on T21 and T24 than other settings, $\epsilon_{\text{multi}} = 10^{-12}$ obtains the best results on T25–T27, $\epsilon_{\text{multi}} = 10^{-16}$ outperforms other settings on T19, T20, T22, and T23, and the best results in T28–T30 are produced by $\epsilon_{\text{multi}} = 10^{-4}$. Generally, when the test functions are construed by functions $f_1 - f_3$, smaller threshold settings (i.e., $\epsilon_{\text{multi}} = 10^{-8}$, $\epsilon_{\text{multi}} = 10^{-12}$, and $\epsilon_{\text{multi}} = 10^{-16}$) can have higher decomposition accuracy than the larger setting $\epsilon_{\text{multi}} = 10^{-4}$, while for problems constructed by T13–T15, larger threshold settings (e.g., $\epsilon_{\text{multi}} = 10^{-4}$) can produce better results than other settings. This may be because that different problems require different threshold values to determine whether the variables are multiplicatively separable. In addition, although $\epsilon_{\text{multi}} = 10^{-4}$ obtains most of the best results among the four settings, it has a much worse result on T15 than other settings, which means that its performance is sensitive to the problem characteristics and may have a poor generalization ability. Therefore, it should not be considered a good setting in this article. Instead, $\epsilon_{\text{multi}} = 10^{-8}$ balances the decomposition accuracy on different problems and is recommended in this article.

In conclusion, the threshold value ϵ_{multi} does not have a significant influence on the decomposition ability of DDG on the additively separable problem, but a good setting of ϵ_{multi} can further improve the ability of DDG to decompose multiplicatively separable problems.

I. Case Study on the Parameter Optimization for Neural Network-Based Application

To further study the effectiveness of the proposed DDG, this part conducts a case study on the NN parameter optimization application. NN is efficient for classification problems and has been widely used in many real-world applications [40], [41]. However, the parameters (e.g., weights and bias) of the NN are essential to the performance, which requires efficient optimization. Moreover, the NN often contains a great number of parameters (always more than 1000) to be optimized. Therefore, the parameters optimization of NN is a typical LSOP, which is suitable to evaluate the application ability of DDG.

In this article, we consider an NN-based classification system for the wine classification task, where the public wine dataset is collected from the UCI Machiness Learning Repository (http://archive.ics.uci.edu/ml/index.php). The dataset contains 178 samples, and each sample has 13 features and belongs to one of the three categories. For such a classification problem, a widely used three-layer NN model is adopted in this article, which includes one input, one hidden, and one output layer, as shown in Fig. 3. The input layer has 13 neurons to receive the 13 features. The hidden layer has 60 neurons, with the activation function of each hidden neuron being the widely used sigmoid function. Moreover, three-dimension one-hot encoding is used to encode the three categories and, therefore, there are three neurons in the output layer. For each input sample X_i, the target output Y_i is a 3-D vector with $Y_{i,j} == 1$ if X_i belongs to the jth category and $Y_{i,j} == 0$ otherwise ($j = 1, 2, 3$). Therefore, given NS samples and their corresponding targeted outputs, the NN parameter optimization problem can be defined as a minimization optimization problem with the objective function $F(\theta)$ as

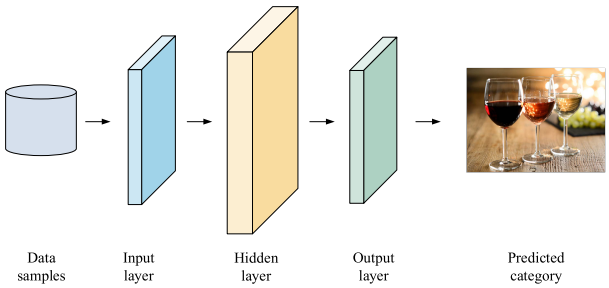


Fig. 3. Neural network-based classification system.

$$\min F(\theta) = \frac{1}{\text{NS}} \sum_{i=1}^{NS} \|\text{NN}(\theta, X_i) - Y_i\|^2 \tag{25}$$

where θ represents the parameters of the NN, that is, the variables to be optimized, $\text{NN}(\theta, X_i)$ is the output of the NN, $||\text{NN}(\theta, X_i) - -Y_i||^2$ computes the square sum of the 3-D differential vector between $\text{NN}(\theta, X_i)$ and Y_i, and smaller $F(\theta)$ is better.

Based on the above, the number of parameters between the input and hidden layers is $60 \times (13 + 1) = 840$ (each of the 60 hidden neurons requires 13 weights for the 13 features inputs and one bias), and the number of parameters between the hidden and output layers is $3 \times (60+1) = 183$ (each of the 3 output neurons requires 60 weights for the 60 hidden neurons and one bias). Therefore, the total number of parameters for optimization is $840 + 183 = 1023$, that is, the parameter optimization problem is with 1023 variables. In addition, for a fair comparison, all decompositions methods are integrated with the CC framework and have 3×10^6 FEs in total for every independent run (including the FEs consumed for decompositions). Besides, the first 80% of samples data are used as training data while the last 20% of samples data are treated as test data, and the search range of each parameter is $[-1, 1]$.

Table VI gives the comparison result of different decomposition-based CC algorithms over 25 runs. As can be seen, the CC-DDG obtains the best training loss and accuracy among the seven algorithms. Moreover, it is interesting that only DDG can decompose the problem into different groups, while all other methods (except the RD) fail to decompose the problem but still consume considerable unnecessary FEs, which suggest that this problem is not additively separable but multiplicatively separable. Therefore, our proposed DDG method is promising. For better visualization, Fig. 4 provides the grouping map of the 1023 variables based on the DDG

TABLE VI
COMPARISONS OF DIFFERENT CC ALGORITHMS FOR PARAMETER OPTIMIZATION OF THE NEURAL NETWORK-BASED SYSTEM

Methods	Training loss (Mean±Std.)	Best accuracy	FEs for grouping	Grouping results
CC-DDG	**1.70E-02 ±5.36E-04**	**97.14%**	62226	1 group with 20 variables, 59 groups with 17 variables
CC-DG	1.73E-02(+) ±6.57E-04	94.29%	2046	only 1 group with 1023 variables
CC-DG2	1.73E-02(+) ±5.84E-04	88.57%	523777	only 1 group with 1023 variables
CC-XDG	1.72E-02(+) ±5.28E-04	94.29%	4090	only 1 group with 1023 variables
CC-GDG	1.73E-02(+) ±9.47E-04	88.57%	524810	only 1 group with 1023 variables
CC-RDG3	1.72E-02(+) ±5.96E-04	94.29%	6130	only 1 group with 1023 variables
CC-RD	1.73E-02(+) ±3.82E-04	91.43%	0	5 random groups

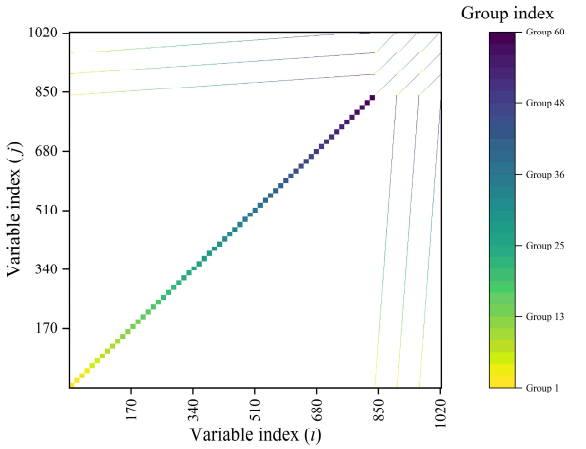


Fig. 4. Grouping map of 1023 variables based on the DDG, where a pixel (i, j) with color means the ith and the jth variables are in the same group, while with white color means not in the same group.

results, where the color of each pixel (i, j) means the ith and jth variables are in the same group of the corresponding color except that white color means the ith and jth variables are not in the same group. As shown in Fig. 4, the groupings have a regular pattern and similar size after the decomposition by DDG. Therefore, the superiority of CC-DDG may be due to that the DDG can decompose the problem appropriately with a small amount of FEs and then the problem can be optimized more efficiently and with more remainder FEs. Based on the above, the case study has verified the effectiveness of the proposed DDG.

V. CONCLUSION

In this article, we attempted to obtain a more general decomposition method to decompose and optimize more kinds of LSOPs correctly and efficiently. First, we mathematically defined the multiplicatively separable function and showed its relationship with the additively separable function. Then, we proposed and proved two related theorems that can guide the detection of the separable structure in multiplicatively separable problems. Based on the above, the novel DDG method was proposed to achieve a more general decomposition ability for LSOP. The DDG can utilize two kinds of differences to detect the separable structure of the additively and multiplicatively separable problem. Moreover, the DDG-based CC algorithm framework is developed for solving LSOP by combining the DDG with a classical and widely used CC framework. In addition, the time complexity of DDG is also analyzed with respect to the number of FEs. Extensive experiments were conducted on 30 LSOPs and a case study on parameter optimization for the NN-based application, where state-of-the-art methods are adopted for comparisons. The experimental results have verified the effectiveness and efficiency of the DDG and the DDG-based algorithm.

For future work, the idea of DDG will be further studied to reduce the FEs cost, improve the detection ability, and address the disadvantage in handling negative objective functions. Furthermore, the DDG will be further studied and extended to solving more kinds of separable problems, such as multiplicatively separable functions generated by different types of functions (e.g., T04–T11). Also, the combination of different decomposition methods and optimizers is worthy of research to develop more powerful algorithms. Besides, the DDG-based algorithms with the distributed computation method [58]–[60] are worthy to be studied and applied to challenging real-world LSOPs and big data applications.

REFERENCES

[1] P. Yang, K. Tang, and X. Yao, "Turning high-dimensional optimization computationally expensive optimization," *IEEE Trans. Evol. Comput.*, vol. 22, no. 1, pp. 143–156, Feb. 2018.
[2] Y.-F. Ge *et al.*, "Distributed differential evolution based on adaptive mergence and split for large-scale optimization," *IEEE Trans. Cybern.*, vol. 48, no. 7, pp. 2166–2180, Jul. 2018.
[3] J. Y. Li, Z.-H. Zhan, J. Xu, S. Kwong, and J. Zhang, "Surrogate-assisted hybrid-model estimation of distribution algorithm for mixed-variable hyperparameters optimization in convolutional neural networks," *IEEE Trans. Neural Netw. Learn. Syst.*, early access, Sep. 20, 2021, doi: 10.1109/TNNLS.2021.3106399.
[4] X. He, Y. Zhou, Z. Chen, J. Zhang, and W. N. Chen, "Large-scale evolution strategy based on search direction adaptation," *IEEE Trans. Cybern.*, vol. 51, no. 3, pp. 1651–1665, Mar. 2021, doi: 10.1109/TCYB.2019.2928563.
[5] Z.-J. Wang *et al.*, "Dynamic group learning distributed particle swarm optimization for large-scale optimization and its application in cloud workflow scheduling," *IEEE Trans. Cybern.*, vol. 50, no. 6, pp. 2715–2729, Jun. 2020.
[6] J.-Y. Li, Z.-H. Zhan, R.-D. Liu, C. Wang, S. Kwong, and J. Zhang, "Generation-level parallelism for evolutionary computation: A pipeline-based parallel particle swarm optimization," *IEEE Trans. Cybern.*, vol. 51, no. 10, pp. 4848–4859, Oct. 2021.
[7] X. Zhang, K.-J. Du, Z.-H. Zhan, S. Kwong, T. Gu, and J. Zhang, "Cooperative coevolutionary bare-bones particle swarm optimization with function independent decomposition for large-scale supply chain network design with uncertainties," *IEEE Trans. Cybern.*, vol. 50, no. 10, pp. 4454–4468, Oct. 2020.
[8] Z.-H. Zhan, L. Shi, K. C. Tan, and J. Zhang, "A survey on evolutionary computation for complex continuous optimization," *Artif. Intell. Rev.*, vol. 55, no. 1, pp. 59–110, 2022.
[9] J.-Y. Li, Z.-H. Zhan, and J. Zhang, "Evolutionary computation for expensive optimization: A survey," *Mach. Intell. Res.*, vol. 19, no. 1, pp. 3–23, 2022.
[10] J.-Y. Li, Z.-H. Zhan, K.-C. Tan, and J. Zhang, "A meta-knowledge transfer-based differential evolution for multitask optimization," *IEEE Trans. Evol. Comput.*, early access, Nov. 29, 2021, doi: 10.1109/TEVC.2021.3131236.

[11] Z.-J. Wang *et al.*, "Automatic niching differential evolution with contour prediction approach for multimodal optimization problems," *IEEE Trans. Evol. Comput.*, vol. 24, no. 1, pp. 114–128, Feb. 2020.

[12] Z.-G. Chen, Z.-H. Zhan, H. Wang, and J. Zhang, "Distributed individuals for multiple peaks: A novel differential evolution for multimodal optimization problems," *IEEE Trans. Evol. Comput.*, vol. 24, no. 4, pp. 708–719, Aug. 2020.

[13] S.-Z. Zhou, Z.-H. Zhan, Z.-G. Chen, S. Kwong, and J. Zhang, "A multi-objective ant colony system algorithm for airline crew rostering problem with fairness and satisfaction," *IEEE Trans. Intell. Transp. Syst.*, vol. 22, no. 11, pp. 6784–6798, Nov. 2021.

[14] X. F. Liu, Z.-H. Zhan, Y. Gao, J. Zhang, S. Kwong, and J. Zhang, "Coevolutionary particle swarm optimization with bottleneck objective learning strategy for many-objective optimization," *IEEE Trans. Evol. Comput.*, vol. 23, no. 4, pp. 587–602, Aug. 2019.

[15] J. Y. Li, Z.-H. Zhan, C. Wang, H. Jin, and J. Zhang, "Boosting data-driven evolutionary algorithm with localized data generation," *IEEE Trans. Evol. Comput.*, vol. 24, no. 5, pp. 923–937, Oct. 2020.

[16] J. Y. Li, Z.-H. Zhan, H. Wang, and J. Zhang, "Data-driven evolutionary algorithm with perturbation-based ensemble surrogates," *IEEE Trans. Cybern.*, vol. 51, no. 8, pp. 3925–3937, Aug. 2021.

[17] Z.-H. Zhan, J.-Y. Li, and J. Zhang, "Evolutionary deep learning: A survey," *Neurocomputing*, vol. 483, pp. 42–58, Apr. 2022.

[18] A. LaTorre, S. Muelas, and J.-M. Peña, "A comprehensive comparison of large scale global optimizers," *Inf. Sci.*, vol. 316, pp. 517–549, Sep. 2015.

[19] S. Mahdavi, M. E. Shiri, and S. Rahnamayan, "Metaheuristics in large-scale global continues optimization: A survey," *Inf. Sci.*, vol. 295, pp. 407–428, Feb. 2015.

[20] X. Ma *et al.*, "A survey on cooperative co-evolutionary algorithms," *IEEE Trans. Evol. Comput.*, vol. 23, no. 3, pp. 421–441, Jun. 2019.

[21] E. Sayed, D. Essam, and R. Sarker, "Dependency identification technique for large scale optimization problems," in *Proc. IEEE Congr. Evol. Comput.*, Jun. 2012, pp. 1–8.

[22] S. Mahdavi, S. Rahnamayan, and M. E. Shiri, "Incremental cooperative coevolution for large-scale global optimization," *Soft Comput.*, vol. 22, no. 6, pp. 2045–2064, 2018.

[23] M. A. Potter and K. A. De Jong, "A cooperative coevolutionary approach to function optimization," in *Proc. Int. Conf. Parallel Problem Solving Nature*, 1994, pp. 249–257.

[24] Y.-H. Jia, Y. Mei, and M. Zhang, "Contribution-based cooperative co-evolution for nonseparable large-scale problems with overlapping subcomponents," *IEEE Trans. Cybern.*, early access, Oct. 29, 2020, doi: 10.1109/TCYB.2020.3025577.

[25] W.-N. Chen, Y.-H. Jia, F. Zhao, X. Luo, X. Jia, and J. Zhang, "A cooperative co-evolutionary approach to large-scale multisource water distribution network optimization," *IEEE Trans. Evol. Comput.*, vol. 23, no. 5, pp. 842–857, Oct. 2019.

[26] Z. Yang, K. Tang, and X. Yao, "Large scale evolutionary optimization using cooperative coevolution," *Inf. Sci.*, vol. 178, no. 15, pp. 2985–2999, 2008.

[27] M. N. Omidvar, X. Li, Y. Mei, and X. Yao, "Cooperative co-evolution with differential grouping for large scale optimization," *IEEE Trans. Evol. Comput.*, vol. 18, no. 3, pp. 378–393, Jun. 2014.

[28] M. N. Omidvar, M. Yang, Y. Mei, X. Li, and X. Yao, "DG2: A faster and more accurate differential grouping for large-scale black-box optimization," *IEEE Trans. Evol. Comput.*, vol. 21, no. 6, pp. 929–942, Dec. 2017.

[29] Y. Sun, M. Kirley, and S. K. Halgamuge, "Extended differential grouping for large scale global optimization with direct and indirect variable interactions," in *Proc. Annu. Conf. Genet. Evol. Comput.*, 2015, pp. 313–320.

[30] Y. Mei, M. N. Omidvar, X. Li, and X. Yao, "A competitive divide-and-conquer algorithm for unconstrained large-scale black-box optimization," *ACM Trans. Math. Softw.*, vol. 42, no. 2, pp. 13–24, 2016.

[31] Y. Sun, M. Kirley, and S. K. Halgamuge, "A recursive decomposition method for large scale continuous optimization," *IEEE Trans. Evol. Comput.*, vol. 22, no. 5, pp. 647–661, Oct. 2018.

[32] Y. Sun, X. Li, A. Ernst, and M. N. Omidvar, "Decomposition for large-scale optimization problems with overlapping components," in *Proc. IEEE Congr. Evol. Comput.*, 2019, pp. 326–333.

[33] H. Ge, L. Sun, G. Tan, Z. Chen, and C. L. P. Chen, "Cooperative hierarchical PSO with two stage variable interaction reconstruction for large scale optimization," *IEEE Trans. Cybern.*, vol. 47, no. 9, pp. 2809–2823, Sep. 2017.

[34] X.-M. Hu, F.-L. He, W.-N. Chen, and J. Zhang, "Cooperation coevolution with fast interdependency identification for large scale optimization," *Inf. Sci.*, vol. 381, pp. 142–160, Mar. 2017.

[35] M. Yang, A. Zhou, C. Li, and X. Yao, "An efficient recursive differential grouping for large-scale continuous problems," *IEEE Trans. Evol. Comput.*, vol. 25, no. 1, pp. 159–171, Feb. 2021.

[36] T. Ray and X. Yao, "A cooperative coevolutionary algorithm with correlation based adaptive variable partitioning," in *Proc. IEEE Congr. Evol. Comput.*, 2009, pp. 983–989.

[37] M. N. Omidvar, X. Li, and X. Yao, "Cooperative co-evolution with delta grouping for large scale non-separable function optimization," in *Proc. IEEE Congr. Evol. Comput.*, Jul. 2010, pp. 1–8.

[38] X. Zhang, Y. Gong, Y. Lin, J. Zhang, S. Kwong, and J. Zhang, "Dynamic cooperative coevolution for large scale optimization," *IEEE Trans. Evol. Comput.*, vol. 23, no. 6, pp. 935–948, Dec. 2019.

[39] J.-Y. Li, Z.-H. Zhan, J. Xu, S. Kwong, and J. Zhang, "Surrogate-assisted hybrid-model estimation of distribution algorithm for mixed-variable hyperparameters optimization in convolution neural networks," *IEEE Trans. Neural Netw. Learn. Syst.*, early access, Sep. 20, 2021, doi: 10.1109/TNNLS.2021.3106399.

[40] J. Jumper *et al.*, "Highly accurate protein structure prediction with AlphaFold," *Nature*, vol. 596, pp. 583–592, Jul. 2021.

[41] X.-F. Liu, Z.-H. Zhan, and J. Zhang, "Resource-aware distributed differential evolution for training expensive neural-network-based controller in power electronic circuit," *IEEE Trans. Neural Netw. Learn. Syst.*, early access, May 7, 2021, doi: 10.1109/TNNLS.2021.3075205.

[42] X. Li, K. Tang, M. N. Omidvar, Z. Yang, and K. Qin, "Benchmark functions for the CEC'2013 special session and competition on large-scale global optimization," Evol. Comput. Mach. Learn. Group, RMIT Univ., Melbourne, VIC, Australia, Rep., 2013. [Online]. Available: http://www.tflsgo.org/special_sessions/cec2019.html#benchmark-competition

[43] W. Chen, T. Weise, Z. Yang, and K. Tang, "Large-scale global optimization using cooperative coevolution with variable interaction learning," in *Proc. Int. Conf. Parallel Problem Solving Nature*, 2010, pp. 300–309.

[44] Y. Wang, H. Liu, F. Wei, T. Zong, and X. Li, "Cooperative coevolution with formula-based variable grouping for large-scale global optimization," *Evol. Comput.*, vol. 26, no. 4, pp. 569–596, Dec. 2018.

[45] H. Liu, Y. Wang, and N. Fan, "A hybrid deep grouping algorithm for large scale global optimization," *IEEE Trans. Evol. Comput.*, vol. 24, no. 6, pp. 1112–1124, Dec. 2020.

[46] W. Chen and K. Tang, "Impact of problem decomposition on cooperative coevolution," in *Proc. IEEE Congr. Evol. Comput.*, Jun. 2013, pp. 733–740.

[47] H. Liu, Y. Wang, X. Liu, and S. Guan, "Empirical study of effect of grouping strategies for large scale optimization," in *Proc. Int. Joint Conf. Neural Netw.*, Jul. 2016, pp. 3433–3439.

[48] M. Yang *et al.*, "Efficient resource allocation in cooperative co-evolution for large-scale global optimization," *IEEE Trans. Evol. Comput.*, vol. 21, no. 4, pp. 493–505, Aug. 2017.

[49] Y.-H. Jia *et al.*, "Distributed cooperative co-evolution with adaptive computing resource allocation for large scale optimization," *IEEE Trans. Evol. Comput.*, vol. 23, no. 2, pp. 188–202, Apr. 2019.

[50] M. N. Omidvar, B. Kazimipour, X. D. Li, and X. Yao, "CBCC3–A contribution-based cooperative co-evolutionary algorithm with improved exploration/exploitation balance," in *Proc. IEEE Congr. Evol. Comput.*, 2016, pp. 3541–3548.

[51] Z. Li and Q. Zhang, "A simple yet efficient evolution strategy for large-scale black-box optimization," *IEEE Trans. Evol. Comput.*, vol. 22, no. 5, pp. 637–646, Oct. 2018.

[52] Y. Guo, J.-Y. Li, and Z. H. Zhan, "Efficient hyperparameter optimization for convolution neural networks in deep learning: A distributed particle swarm optimization approach," *Cybern. Syst.*, vol. 52, no. 1, pp. 36–57, 2020.

[53] Z. Yang, K. Tang, and X. Yao, "Self-adaptive differential evolution with neighborhood search," in *Proc. IEEE Congr. Evol. Comput.*, 2008, pp. 1110–1116.

[54] N. Hansen and A. Ostermeier, "Completely derandomized self-adaptation in evolution strategies," *Evol. Comput.*, vol. 9, no. 2, pp. 159–195, 2001.

[55] D. Molina, A. LaTorre, and F. Herrera. "SHADE with iterative local search for large-scale global optimization," in *Proc. IEEE Congr. Evol. Comput.*, 2018, pp. 1–8.

[56] A. A. Hadi, A. W. Mohamed, and K. M. Jambi, "LSHADE-SPA memetic framework for solving large-scale optimization problems," *Complex Intell. Syst.*, vol. 5, pp. 25–40, Mar. 2019.

[57] A. LaTorre, S. Muelas, and J.-M. Pe∼na, "Large scale global optimization: Experimental results with MOS-based hybrid algorithms," in *Proc. IEEE Congr. Evol. Comput.*, 2013, pp. 2742–2749.
[58] Z.-H. Zhan *et al.*, "Cloudde: A heterogeneous differential evolution algorithm and its distributed cloud version," *IEEE Trans. Parallel Distrib. Syst.*, vol. 28, no. 3, pp. 704–716, Mar. 2017.
[59] J.-Y. Li, K.-J. Du, Z.-H. Zhan, H. Wang, and J. Zhang, "Distributed differential evolution with adaptive resource allocation," *IEEE Trans. Cybern.*, early access, Jan. 1, 2022, doi: 10.1109/TCYB.2022.3153964.
[60] Z.-H. Zhan, Z.-J. Wang, H. Jin, and J. Zhang, "Adaptive distributed differential evolution," *IEEE Trans. Cybern.*, vol. 50, no. 11, pp. 4633–4647, Nov. 2020.

Jian-Yu Li (Student Member, IEEE) received the B.S. degree in computer science and technology from the South China University of Technology, Guangzhou, China, in 2018, where he is currently pursuing the Ph.D. degree in computer science and technology with the School of Computer Science and Engineering.

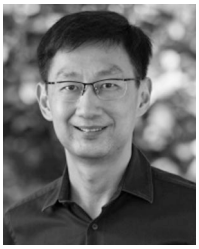
His research interests mainly include computational intelligence, data-driven optimization, machine learning, including deep learning, and their applications in real-world problems, and in environments of distributed computing and big data.

Dr. Li has been invited as a Reviewer of the IEEE TRANSACTIONS ON EVOLUTIONARY COMPUTATION and *Neurocomputing*.

Zhi-Hui Zhan (Senior Member, IEEE) received the bachelor's and Ph.D. degrees in computer science from the Sun Yat-sen University, Guangzhou, China, in 2007 and 2013, respectively.

He is currently the Changjiang Scholar Young Professor with the School of Computer Science and Engineering, South China University of Technology, Guangzhou. His current research interests include evolutionary computation, swarm intelligence, and their applications in real-world problems and in environments of cloud computing and big data.

Dr. Zhan was a recipient of the IEEE Computational Intelligence Society (CIS) Outstanding Early Career Award in 2021, the Outstanding Youth Science Foundation from National Natural Science Foundations of China in 2018, and the Wu Wen-Jun Artificial Intelligence Excellent Youth from the Chinese Association for Artificial Intelligence in 2017. His doctoral dissertation was awarded the IEEE CIS Outstanding Ph.D. Dissertation and the China Computer Federation Outstanding Ph.D. Dissertation. He is one of the World's Top 2% Scientists for both Career-Long Impact and Year Impact in Artificial Intelligence and one of the Highly Cited Chinese Researchers in Computer Science. He is currently the Chair of the Membership Development Committee in IEEE Guangzhou Section and the Vice-Chair of IEEE CIS Guangzhou Chapter. He is currently an Associate Editor of the IEEE TRANSACTIONS ON EVOLUTIONARY COMPUTATION, *Neurocomputing*, and *Memetic Computing*.



Kay Chen Tan (Fellow, IEEE) received the B.Eng. (First Class Hons.) and Ph.D. degrees from the University of Glasgow, Glasgow, U.K., in 1994 and 1997, respectively.

He is currently a Chair Professor of Computational Intelligence with the Department of Computing, The Hong Kong Polytechnic University, Hong Kong. He has published over 300 refereed articles and seven books.

Prof. Tan is currently the Vice-President (Publications) of IEEE Computational Intelligence Society, USA. He served as the Editor-in-Chief for the *IEEE Computational Intelligence Magazine* from 2010 to 2013 and the IEEE TRANSACTIONS ON EVOLUTIONARY COMPUTATION from 2015 to 2020. He currently serves as an editorial board member for more than ten journals. He is an IEEE Distinguished Lecturer Program Speaker and the Chief Co-Editor of Springer Book Series on *Machine Learning: Foundations, Methodologies, and Applications*.

Jun Zhang (Fellow, IEEE) received the Ph.D. degree from the City University of Hong Kong, Hong Kong, SAR, in 2002.

He is currently a Korea Brain Pool Fellow Professor with Hanyang University, Ansan, South Korea. His current research interests include computational intelligence, cloud computing, operations research, and power electronic circuits. He has published over more than 150 IEEE Transactions papers in his research areas.

Dr. Zhang was a recipient of the First-Grade Award in Natural Science Research from the Ministry of Education, China, in 2009, the National Science Fund for Distinguished Young Scholars of China in 2011, and the Changjiang Chair Professor from the Ministry of Education, China, in 2013. He is currently an Associate Editor of the IEEE TRANSACTIONS ON CYBERNETICS and the IEEE TRANSACTIONS ON EVOLUTIONARY COMPUTATION.