DG2: A Faster and More Accurate Differential Grouping for Large-Scale Black-Box Optimization

Mohammad Nabi Omidvar, *Member, IEEE*, Ming Yang, Yi Mei, *Member, IEEE*,
Xiaodong Li, *Senior Member, IEEE*, and Xin Yao, *Fellow, IEEE*

***Abstract*—Identification of variable interaction is essential for an efficient implementation of a divide-and-conquer algorithm for large-scale black-box optimization. In this paper, we propose an improved variant of the differential grouping (DG) algorithm, which has a better efficiency and grouping accuracy. The proposed algorithm, DG2, finds a reliable threshold value by estimating the magnitude of roundoff errors. With respect to efficiency, DG2 reuses the sample points that are generated for detecting interactions and saves up to half of the computational resources on fully separable functions. We mathematically show that the new sampling technique achieves the lower bound with respect to the number of function evaluations. Unlike its predecessor, DG2 checks all possible pairs of variables for interactions and has the capacity to identify overlapping components of an objective function. On the accuracy aspect, DG2 outperforms the state-of-the-art decomposition methods on the latest large-scale continuous optimization benchmark suites. DG2 also performs reliably in the presence of imbalance among contribution of components in an objective function. Another major advantage of DG2 is the automatic calculation of its threshold parameter (ϵ), which makes it parameter-free. Finally, the experimental results show that when DG2 is used within a cooperative co-evolutionary framework, it can generate competitive results as compared to several state-of-the-art algorithms.**

***Index Terms*—Cooperative co-evolution, differential grouping (DG), large-scale global optimization, problem decomposition.**

Manuscript received June 10, 2016; revised September 29, 2016 and February 10, 2017; accepted March 29, 2017. Date of publication April 25, 2017; date of current version November 28, 2017. This work was supported in part by the EPSRC under Grant EP/K001523/1 and Grant EP/J017515/1, in part by the ARC Discovery under Grant DP120102205, in part by the National Natural Science Foundation of China under Grant 61305086 and Grant 61329302, and in part by the Open Research Project of the Hubei Key Laboratory of Intelligent Geo-Information Processing under Grant KLIGIP201602. The work of X. Yao was supported by the Royal Society Wolfson Research Merit Award. *(Corresponding author: Xin Yao.)*

M. N. Omidvar is with the Center of Excellence for Research in Computational Intelligence and Applications, School of Computer Science, University of Birmingham, Birmingham B15 2TT, U.K. (e-mail: m.omidvar@cs.bham.ac).

M. Yang is with the School of Computer Science, China University of Geosciences, Wuhan 430074, China (e-mail: yangming0702@gmail.com).

Y. Mei is with the School of Engineering and Computer Science, Victoria University of Wellington, Wellington 6012, New Zealand (e-mail: yi.mei@ecs.vuw.ac.nz).

X. Li is with the School of Computer Science and Information Technology, RMIT University, Melbourne, VIC 3001, Australia (e-mail: xiaodong.li@rmit.edu.au).

X. Yao is with the Department of Computer Science and Engineering, Southern University of Science and Technology, Shenzhen 518055, China, and also with the Center of Excellence for Research in Computational Intelligence and Applications, School of Computer Science, University of Birmingham, Birmingham B15 2TT, U.K. (e-mail: x.yao@cs.bham.ac.uk).

This paper has supplementary downloadable multimedia material available at http://ieeexplore.ieee.org provided by the authors.

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TEVC.2017.2694221

I. Introduction

Large-scale global optimization has become an active field of research in the past decade due to the growing number of large-scale optimization problems in engineering and sciences [1], [2]. Most engineering problems have shown an exponential increase in the number of decision variables they entail [3]. Advances in machine learning and the rise of deep artificial neural networks has resulted in optimization problems with over a billion variables [4], [5]. Ubiquity of data has also caused the emergence of large-scale optimization problems at the heart of many data analytics and learning problems [6]. Target shape design optimization for aircraft wings and turbine blades [7], satellite layout design [8], parameter estimation in large-scale systems biology models [9], seismic waveform inversion [10], and parameter calibration of water distribution system [11] are just a few examples from a wide array of large-scale optimization problems.

A major challenge of large-scale optimization is the exponential growth in the size of the search space with respect to the number of decision variables. It is this *curse-of-dimensionality* that has made large-scale optimization an exceedingly difficult task. This motivated the development of a wide range of scalable algorithms in the classic mathematical programming domain [12], [13] as well as metaheuristics [14], [15]. Evolutionary algorithms, in particular, have shown superior performance as compared to other classic methods on problems with millions or even billions of variables [16], [17]. Other methods such as swarm intelligence [18]–[20], memetic algorithms [21]–[23], differential evolution [24], [25], evolution strategies [26], and estimation of distribution algorithms [27], [28] have also gained popularity for large-scale optimization because of their ability to deal with black-box problems. It should be noted that the notion of large-scale changes over time and varies from problem to problem. In a broad sense, a problem is considered large-scale if it causes scalability issues on the state-of-the-art algorithms. For the current study, which focuses on real-parameter optimization, the existing algorithms exhibit scalability issues on problems having more than about a hundred decision variables.

A number of approaches such as dimensionality reduction [29], surrogate modeling [30], local search [21], [22], and divide-and-conquer (also known as decomposition) methods [31] can be used for large-scale optimization, among which decomposition methods have gained popularity in recent years [18], [31]–[47]. Decomposition methods break a large-scale problem into a set of smaller and simpler subproblems each of which is optimized in an iterative manner. In the context of evolutionary algorithms, cooperative co-evolution [48] is a popular means of exploiting the modular nature of many complex large-scale problems and has been used in a wide range of areas beyond optimization, such as the study of evolutionary game theory [49]. A major challenge of using a cooperative co-evolutionary framework for large-scale optimization is the right choice of problem decomposition. Ideally, a given objective function should be decomposed such that the interaction between the resultant components is minimized. For a black-box optimization problem, the variable interaction information are not available. Therefore, specific algorithms are required to identify the underlying interaction structure of the decision variables.

Differential grouping (DG) [32] is a competitive decomposition algorithm that can identify the nonseparable components of a continuous objective function and has shown superior performance as compared to other decomposition algorithms, such as variable interaction learning [33] on the CEC'2010 [50] large-scale benchmark suite [32]. Despite its success on the CEC'2010 benchmark problems, it has been shown that DG has some difficulty with the CEC'2013 large-scale benchmark functions [51]. In particular, DG has the following major shortcomings.

1) High computational cost on fully separable functions.
2) Inability to detect objective functions with overlapping components, i.e., components that share decision variables [51].
3) Sensitivity to computational roundoff errors [37].
4) Requiring the user to specify a threshold parameter (ϵ).

In this paper, we propose an improved version of DG that addresses the above issues. In particular, this improved version, DG2, reduces the total number of objective function evaluations by half for fully separable functions which require the most function evaluations. This allows the algorithm to check all pairs of variables for interaction at a much lower cost as compared to its predecessor. Testing all pairs of variables for interaction is essential to identify functions with overlapping components. The reduction in the total number of objective function evaluations is achieved through systematic generation of sample points to maximize point reuse in the process of applying the DG theorem (see Section II). We mathematically show that this new method achieves the lower bound when the DG theorem is used to detect the interactions.

In addition to improving the efficiency, DG2 significantly improves the grouping accuracy of DG on the existing large-scale benchmark suites. A major advantage of DG2 is its parameter-free property. DG2 takes the computational rounding errors into account in estimating a proper threshold value (ϵ) which determines its sensitivity to weak interactions. In particular, DG2 has the following advantages over the static method used in DG.

1) Unlike DG that uses a single global ϵ value to detect all the interactions, DG2 dynamically calculates an ϵ value to detect the interaction between each pair of variables. For each interaction, DG2 approximates the magnitude of roundoff errors and calculates the threshold value accordingly. This is particularly useful when dealing with imbalanced functions, in which the magnitude of roundoff error may be different from component to component.
2) Unlike DG, the new method does not require the user to specify any external parameter. In other words, DG2 is parameter-free.

The organization of the rest of this paper is as follows. Section II contains the details of the DG theorem and algorithm. Section III gives an outline of the proposed improvements. Section III-A contains the details on how to reduce the total required objective function evaluations as well as a proof of the lower bound for the total required evaluations. Section III-B focuses on improving the accuracy of DG and making it parameter-free. The experimental results about the grouping accuracy of DG2 and its performance within a cooperative co-evolutionary framework are presented in Section IV. Finally, Section V concludes this paper.

II. Background and Related Work

Problem decomposition is an integral part of using cooperative co-evolution for function optimization. A good problem decomposition is one that has minimal dependence among its components. This is often characterized by separability structure of the objective function, which is defined as follows.

Definition 1 [51]: A function $f(\mathbf{x})$ is partially separable with m independent components iff

$$\arg\min_{\mathbf{x}} f(\mathbf{x}) = \left(\arg\min_{\mathbf{x}_1} f(\mathbf{x}_1, \ldots), \ldots, \arg\min_{\mathbf{x}_m} f(\ldots, \mathbf{x}_m)\right)$$

where $\mathbf{x} = (x_1, \ldots, x_n)^\top$ is a decision vector of n dimensions, $\mathbf{x}_1, \ldots, \mathbf{x}_m$ are disjoint subvectors of $\mathbf{x}$, and $2 \le m \le n$.

Additive separability is a special type of partial separability, which is defined as follows.

Definition 2 [51]: A function is partially additively separable if it has the following general form:

$$f(\mathbf{x}) = \sum_{i=1}^{m} f_i(\mathbf{x}_i),\ m > 1$$

where $f_i(\cdot)$ is a nonseparable subfunction, and m is the number of nonseparable components of f. The definition of $\mathbf{x}$ and $\mathbf{x}_i$ is identical to what was given in Definition 1.

Many decomposition algorithms have been proposed to decompose a black-box optimization problem into smaller subproblems. Static grouping is the simplest decomposition strategy in which the decision variables are grouped into arbitrary groups. In its simplest form, an n-dimensional problem is broken down into s k-dimensional problems. Examples of such methods are the divide-in-half method by Shi *et al.* [52], and the method employed by van den Bergh and Engelbrecht [53]. These methods are oblivious of variable interactions which may have a significant impact on the optimization performance [31]. Some other decomposition algorithms, such as random grouping [31], adaptive variable partitioning [39], delta grouping [54], and min/max variance decomposition [55] use various heuristics in order to form the groups based on variable interaction characteristics of the objective function. The drawback of these methods is their low grouping accuracy, and the fact that they presuppose the number and/or the size of components. These algorithms also divide the decision variables into s k-dimensional components. Improved versions of random grouping and delta grouping use a so-called multilevel strategy [54], [56] in which multiple fixed decompositions are used over the course of optimization. More sophisticated decomposition methods, such as variable interaction learning [33], meta-modeling decomposition [40], statistical learning decomposition [38], and DG [32] do not presuppose the number and/or size of components. Among these algorithms, DG has shown superior performance with respect to grouping accuracy [33], [40]. The following theorem is at the heart of interaction detection of DG.

Theorem 1 [32]: Let $f(\mathbf{x})$ be an additively separable function. $\forall a, b_1 \neq b_2, \delta \in \mathbb{R}$,[1] $\delta \neq 0$, variables x_p and x_q interact if the following condition holds:

$$\Delta_{\delta,x_p}[f](\mathbf{x})\big|_{x_p=a,x_q=b_1} \neq \Delta_{\delta,x_p}[f](\mathbf{x})\big|_{x_p=a,x_q=b_2} \qquad (1)$$

where

$$\Delta_{\delta,x_p}[f](\mathbf{x}) = f(\ldots, x_p+\delta, \ldots) - f(\ldots, x_p, \ldots) \qquad (2)$$

refers to the forward difference of f with respect to variable x_p with interval δ.

Theorem 1 states that two variables x_p and x_q interact if (2) evaluated with any two different values of x_q gives different results [32]. A proof of this theorem can be found in [32]. For the sake of brevity the left hand side of (1) is denoted by $\Delta^{(1)}$ and its right hand side by $\Delta^{(2)}$. It is clear that $\Delta^{(1)} \neq \Delta^{(2)} \iff |\Delta^{(1)} - \Delta^{(2)}| \neq 0$. It is also clear that this equality check is not practical on computing devices due to limited precision of floating-point numbers. For this reason, the equality check can be converted into an inequality check of the form $\lambda = |\Delta^{(1)} - \Delta^{(2)}| > \epsilon$ by introducing the control parameter ϵ that determines the sensitivity of DG to interactions.

Two major drawbacks of DG are its sensitivity to the parameter ϵ and its poor accuracy in detecting interacting variables on functions with overlapping components. As reported in [32], the grouping accuracy of DG is low on the Rosenbrock function [57] which has overlapping components with overlap size of one. Also, if DG is used to find the interaction structure of functions with overlapping variables, the shared decision variables between two components will be placed in one group and will be excluded from other groups. It is not yet clear what an optimal decomposition may be for an overlapping function; nevertheless, an accurate identification of the underlying structure is essential to propose a meaningful decomposition.

Global DG (GDG) [37] and extended DG (XDG) [58] are two variants of DG, which aim at addressing the above shortcomings. XDG focuses on identifying *indirect* interactions in order to deal with the Rosenbrock function. The issue with

[1]Values of a, b_1, b_2, and δ are chosen such that f is evaluated within its domain.

Algorithm 1: $(g, \mathbf{x}_1, \ldots, \mathbf{x}_g, \mathbf{x}_{\text{sep}}, \Gamma) = \texttt{DG2}(f, n, \overline{\mathbf{x}}, \underline{\mathbf{x}})$

```
1 $(\boldsymbol{\Lambda}, \mathbf{F}, \check{\mathbf{f}}, f_{\text{base}}, \Gamma) = \texttt{ISM}(f, n, \overline{\mathbf{x}}, \underline{\mathbf{x}})$;
2 $\boldsymbol{\Theta} = \texttt{DSM}(\boldsymbol{\Lambda}, \mathbf{F}, \check{\mathbf{f}}, f_{\text{base}}, n)$;
3 $(k, \mathbf{y}_1, \ldots, \mathbf{y}_k) = \texttt{ConnComp}(\boldsymbol{\Theta})$ ;
4 $\mathbf{x}_{sep} = \{\}$, $g = 0$;
5 for $i = 1 \to k$ do
6     if $|\mathbf{y}_i| = 1$ then
7         $\mathbf{x}_{sep} = \mathbf{x}_{sep} \cup \mathbf{y}_i$;
8     else
9         $g = g + 1$, $\mathbf{x}_g = \mathbf{y}_i$;
```

XDG is that it inherits the sensitivity issue of DG and also its method of inferring variable interaction may consider separable variables as nonseparable. This issue is discussed further in Section IV-A. GDG addresses the sensitivity issue of DG by taking computational errors into account. However, the use of a global parameter to detect all interactions makes it unsuitable for imbalanced functions. This issue is discussed further in Section IV-B. GDG also addresses the problem of identifying overlapping functions by examining all pairs of variables for interaction. However, we will show in Section IV that DG2 can achieve the same goal with fewer objective function evaluations.

III. Improved Differential Grouping

In this section, we describe the details of improving the grouping accuracy and efficiency of DG. Algorithm 1 shows the high-level structure of DG2 that incorporates these improvements. DG2 has three major parts. The first part is forming what we call a *raw interaction structure matrix* ($\boldsymbol{\Lambda}$) that contains the quantity $|\Delta^{(1)} - \Delta^{(2)}|$ for all pairs of variables. This is done by the `ISM` function. The second part of the algorithm is finding a suitable threshold parameter (ϵ) in order to convert the raw interaction structure matrix $\boldsymbol{\Lambda}$ to a *design structure matrix* $\boldsymbol{\Theta}$. The entry $\boldsymbol{\Theta}_{ij}$ takes 1 if $\boldsymbol{\Lambda}_{ij} > \epsilon$, and 0 otherwise. It should be noted that unlike DG and GDG, DG2 obtains a threshold based on information, such as magnitude of function values and the values of the raw interaction structure matrix that are calculated by the `ISM` function (Algorithm 1). Finally, the last part of the algorithm deals with the decomposition of the variables into nonseparable groups, which is performed by identifying the connected components of the graph with the node adjacency matrix $\boldsymbol{\Theta}$. This can be efficiently done in linear time in n [59].

It should be noted that a complete design structure matrix is necessary to detect overlapping functions, in which different components share common variables. This type of functions is more general in practice, and is more challenging. Given the design structure matrix, various decompositions can be devised in order to deal with overlapping components. However, the study of an optimal decomposition for overlapping functions is beyond the scope of this paper.

In the remainder of this paper, we focus on two major issues.

1) Finding an efficient implementation for the `ISM` function in order to form the interaction structure matrix using the minimum possible function evaluations (Section III-A).

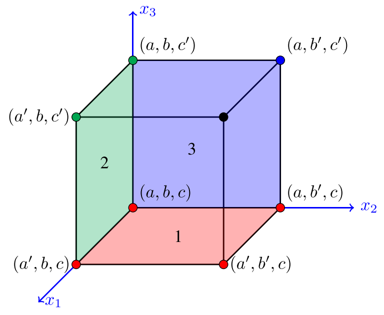

Fig. 1. Geometric representation of point generation in DG2 for a 3-D function.

2) Finding an effective thresholding method that results in an accurate decomposition of a function into its components that generalizes over a wide range of functions (Section III-B).

A. Improving the Efficiency of Differential Grouping

As mentioned earlier, in order to detect the overlapping functions, it is essential to examine all pairs of variables for interaction. It is clear that for an n-dimensional function, the total number interactions is $\binom{n}{2}$. According to Theorem 1 each comparison requires four fitness evaluations which results in a total of $4 \cdot \binom{n}{2} = 2n(n-1)$ evaluations.

In this section, we show that by systematic selection of sample points for calculating the difference equation (2), the total number of fitness evaluations can be significantly reduced. In order to show this, we assume a simple function with only three decision variables, i.e., $f(x_1, x_2, x_3)$. The total number of function evaluations according to Theorem 1 is as follows:

$$\begin{aligned}
x_1\leftrightarrow x_2 &: \Delta^{(1)} = f\big(a', b, c\big) - f(a,b,c), \Delta^{(2)} = f\big(a', b', c\big) - f\big(a, b', c\big)\\
x_1\leftrightarrow x_3 &: \Delta^{(1)} = f\big(a', b, c\big) - f(a,b,c), \Delta^{(2)} = f\big(a', b, c'\big) - f\big(a, b, c'\big)\\
x_2\leftrightarrow x_3 &: \Delta^{(1)} = f\big(a, b', c\big) - f(a,b,c), \Delta^{(2)} = f\big(a, b', c'\big) - f\big(a, b, c'\big)
\end{aligned}$$

where a, b, and c are the values taken by x_1, x_2, and x_3 respectively, and $a' = x_1 + \delta$, $b' = x_2 + \delta$, and $c' = x_3 + \delta$.

For a clearer illustration, the points that are evaluated with function f are color-coded and are shown geometrically in Fig. 1. From previous calculations we know that the total number of function evaluations for a 3-D function is $2n(n-1)|_{n=3} = 12$. However, it is clear from Fig. 1 that only seven unique points are required.

In order to calculate $\Delta^{(1)}$ and $\Delta^{(2)}$, four points are required. According to Theorem 1 these points are chosen such that they form a rectangle. To calculate $\Delta^{(1)}$ a base point is required which, in this example, is (a, b, c). Then, in order to find interactions with x_1, the first variable should be varied in order to calculate $\Delta^{(1)}$. Therefore, the second point will be (a', b, c). To find the interaction between x_1 and x_2, the same difference equation as $\Delta^{(1)}$ should be evaluated for a different value of x_2. Therefore, we get (a', b', c) and (a, b', c). If we follow this pattern to find all interactions, we can see that the base point (a, b, c) is repeated exactly three times, the cases where only one dimension is varied with respect to the base point, such as (a', b, c), (a, b', c), and (a, b, c') are repeated exactly two times, and the cases where two of the dimensions are varied with respect to the base point, such as (a', b', c), (a', b, c'), and (a, b', c') are evaluated only once.

This process can be generalized for an arbitrary number of decision variables. For a general case, we need the following evaluations in order to detect the interaction between the ith and the jth dimensions:

$$x_i - x_j \text{ interaction :}$$
$$\begin{cases}\Delta^{(1)} = f\big(\ldots, x_i', \ldots\big) - f(x_1, \ldots, x_n)\\ \Delta^{(2)} = f\Big(\ldots, x_i', \ldots, x_j', \ldots\Big) - f\Big(\ldots, x_j', \ldots\Big).\end{cases}$$

Based on this pattern we see that the total number of evaluations is $2n(n-1)$. It should be noted that the number of *unique* evaluations is much less than this quantity due to redundant evaluations caused by the assumptions made previously

$$\begin{cases}\dfrac{n(n-1)}{2} - 1 & \text{: redundant evaluations of } (x_1, \ldots, x_n)\\ n(n-2) & \text{: redundant evaluations of } \big(\ldots, x_i', \ldots\big).\end{cases}$$

Therefore, to calculate the total number of *unique* evaluations, the number of redundant evaluations should be subtracted from the total, which yields the following[2]:

$$\frac{n(n+1)}{2} + 1.$$

In Theorem S.1, we show that this is the minimum number of objective function evaluations needed to form the interaction structure matrix. Theorem S.1 and its proof can be found in the supplementary material accompanying this paper (Section S-I). Algorithm 2 is an implementation of the process that was described above and achieves the lower bound according to Theorem S.1. The `ISM` function, generates the interaction structure matrix ($\mathbf{\Lambda}$) which is used by the `DSM` function to find a reliable ϵ to establish the separability or nonseparability of all pairs of variables.

B. Improving the Grouping Accuracy of Differential Grouping

It was mentioned in Section II that the grouping accuracy of DG depends on ϵ. Theoretically, the value of ϵ can be set to zero, since any positive difference between $\Delta^{(1)}$ and $\Delta^{(2)}$ implies an interaction between the variables in examination. However, in practice, the floating-point operations incur computational roundoff errors and cause nonzero λ values even for separable variables. A major challenge for DG is to distinguish between a genuine nonzero λ due to variable interaction, and a nonzero λ due to computational errors. In this section, we show that the magnitude of roundoff errors is a function of the magnitude of the quantities used in a calculation. This makes a static threshold, such as the one used in DG, an ineffective method. Omidvar *et al.* [51] have shown that the nonuniform contribution of components in an objective function significantly affects the accuracy of DG with a static threshold. The functions in the CEC'2013 large-scale benchmark suite have such an imbalance property.

[2]A more detailed derivation is provided in Section S-I in the supplementary material accompanying this paper.

Algorithm 2: $(\mathbf{\Lambda}, \mathbf{F}, \check{\mathbf{f}}, f_{\text{base}}, \Gamma) = \texttt{ISM}(f, n, \overline{\mathbf{x}}, \underline{\mathbf{x}})$

```
1  $\mathbf{\Lambda} = \mathbf{0}_{n\times n}$;
2  $\mathbf{F}_{n\times n} = \text{NaN}_{n\times n}$ ;                // matrix of all NaNs
3  $\check{\mathbf{f}}_{n\times 1} = \text{NaN}_{n\times 1}$ ;               // vector of all NaNs
4  $\mathbf{x}^{(1)} = \underline{\mathbf{x}}$, $f_{\text{base}} = f(\mathbf{x}^{(1)})$, $\Gamma = 1$;
5  $\mathbf{m} = \frac{1}{2}(\overline{\mathbf{x}} + \underline{\mathbf{x}})$;
6  for $i = 1 \to n-1$ do
7      if $\neg\texttt{isnan}(\check{\mathbf{f}}_i)$ then
8          $\mathbf{x}^{(2)} = \mathbf{x}^{(1)}$, $\mathbf{x}^{(2)}_i = \mathbf{m}_i$;
9          $\check{\mathbf{f}}_i = f(\mathbf{x}^{(2)})$, $\Gamma = \Gamma + 1$;
10     for $j = i+1 \to n$ do
11         if $\neg\texttt{isnan}(\check{\mathbf{f}}_j)$ then
12             $\mathbf{x}^{(3)} = \mathbf{x}^{(1)}$, $\mathbf{x}^{(3)}_j = \mathbf{m}_j$;
13             $\check{\mathbf{f}}_j = f(\mathbf{x}^{(3)})$, $\Gamma = \Gamma + 1$;
14         $\mathbf{x}^{(4)} = \mathbf{x}^{(1)}$, $\mathbf{x}^{(4)}_i = \mathbf{m}_i$, $\mathbf{x}^{(4)}_j = \mathbf{m}_j$;
15         $\mathbf{F}_{ij} = f(\mathbf{x}^{(4)})$, $\Gamma = \Gamma + 1$;
16         $\Delta^{(1)} = \check{\mathbf{f}}_i - f(\mathbf{x}^{(1)})$;
17         $\Delta^{(2)} = \mathbf{F}_{ij} - \check{\mathbf{f}}_j$;
18         $\mathbf{\Lambda}_{ij} = |\Delta^{(1)} - \Delta^{(2)}|$;
```

The new method of calculating a threshold value estimates the greatest lower bound (e_{inf}) and the least upper bound (e_{sup}) for the roundoff error by a mechanism which will be explained later. These values are calculated separately for each pair of variables based on the available information, such as function values and the quantity $\lambda = |\Delta^{(1)} - \Delta^{(2)}|$ to maximize the detection accuracy. Once the bounds are found, two variables are considered to interact if $\lambda > e_{\text{sup}}$, and separable if $\lambda < e_{\text{inf}}$. In order words, this interval defines a *safe* region that determines genuine zero and nonzero λ values. The λ values that fall outside this region may or may not be genuine nonzero values. To overcome this, we calculate the relative proportion of genuine zero and nonzero values and use it to bias the threshold toward either e_{inf} or e_{sup}. The details of this process is given next.

Based on the IEEE 754 Standard [60], the mapping of a real number x to a floating-point number [detonated by $fl(x)$] may impose a rounding to the nearest representable number. According to the IEEE 754 Standard, the representation error for a number x is a function of itself because: $fl(x) = x(1+\delta) = x+\delta x$, where the bounds for δ is determined by a machine dependent constant called the machine epsilon (μ_{M}) such that $|\delta| < \mu_{\text{M}}$ (see Theorem S.2 in Section S-II in the supplementary material). Therefore, the error term δx will grow with x. Since DG may deal with large numbers, it is essential to take the magnitude of the function values into account when estimating the threshold value (ϵ).

In addition to the representational rounding error that was explained above, the floating-point arithmetic also incurs computational rounding error. The IEEE Standard guarantees that $x \oplus y = fl(x+y)$, where $\oplus$ represents floating-point summation operator.[3] In other words, the floating-point sum of two numbers is guaranteed to be equal to the floating-point number closest to the real sum of the two numbers. In most models of error analysis, this is generalized to other operations, such as subtraction, multiplication, division, and sometimes the square root function. This statement does not hold for a sequence of floating-point operations, such as $x_1 \oplus x_2 \oplus + \cdots \oplus x_n$ due to the accumulation of errors.

[3]All other basic floating-point operations are shown in a circle in a similar way.

Theorem S.3 [61] can be used to find an upper bound for the accumulated arithmetic error in any calculation. An example of applying Theorem S.3 to calculate an upper bound for a dot-product example is given in Section S-II-A (the supplementary material). In this paper, we use Theorem S.3 to find a reasonable upper and lower bounds for the error involved in calculating $\mathbf{\Lambda}$. To estimate the greatest lower bound (infimum) for the magnitude of the roundoff error, we assume that the calculation of $f(\mathbf{x})$ is error free, and the only source of error is in the application of DG, i.e., the calculation of $\lambda = |\Delta^{(1)} - \Delta^{(2)}|$. Thus

$$\begin{aligned}
\hat{\Delta}_1 &= f(\mathbf{x}) \ominus f(\mathbf{x}') = \big(f(\mathbf{x}) - f(\mathbf{x}')\big)(1+\delta_1) = \Delta^{(1)}(1+\delta_1)\\
\hat{\Delta}_2 &= f(\mathbf{y}) \ominus f(\mathbf{y}') = \big(f(\mathbf{y}) - f(\mathbf{y}')\big)(1+\delta_2) = \Delta^{(2)}(1+\delta_2)\\
\hat{\lambda} &= \Big|\hat{\Delta}_1 \ominus \hat{\Delta}_2\Big| = \Big|\hat{\Delta}_1 - \hat{\Delta}_2\Big|(1+\delta_3)\\
&= \big|f(\mathbf{x})(1+\delta_1)(1+\delta_3) - f(\mathbf{x}')(1+\delta_1)(1+\delta_3)\\
&\qquad - f(\mathbf{y})(1+\delta_2)(1+\delta_3) + f(\mathbf{y}')(1+\delta_2)(1+\delta_3)\big|.
\end{aligned}$$

We can see that the maximum number of products of the form $(1+\delta_i)$ is 2 ($k = 2$). Therefore, by applying Theorem S.3 we have

$$\begin{aligned}
\Big|\lambda - \hat{\lambda}\Big| &\le \gamma_2\big|\big(f(\mathbf{x}) - f(\mathbf{x}')\big) - \big(f(\mathbf{y}) - f(\mathbf{y}')\big)\big|\\
&= \gamma_2\big|\big(f(\mathbf{x}) + f(\mathbf{y}')\big) - \big(f(\mathbf{y}) + f(\mathbf{x}')\big)\big|\\
&\le \gamma_2 \cdot \max\big\{\big(f(\mathbf{x}) + f(\mathbf{y}')\big), \big(f(\mathbf{y}) + f(\mathbf{x}')\big)\big\} := e_{\text{inf}}. \quad (3)
\end{aligned}$$

Equation (3) is based on the assumption that the codomain of f is non-negative, i.e., $f : \mathbb{R} \to \mathbb{R}^+_0$. A more general form for $f : \mathbb{R} \to \mathbb{R}$ is as follows:

$$e_{\text{inf}} = \gamma_2\big(|f(\mathbf{x})| + \big|f(\mathbf{y}')\big| + |f(\mathbf{y})| + \big|f(\mathbf{x}')\big|\big). \quad (4)$$

In this paper, the calculation of e_{inf} is based on (3).

To estimate an upper bound for the roundoff error, we cannot assume that function evaluations are error free. However, the difficulty here is that the functions are black-box; therefore, we do not know the exact number of error terms $(1+\delta_i)$ in the calculation of $f(\cdot)$. As a rule of thumb in the field of error analysis, it is customary to assume that the error grows with the square root of the number of floating-point operations (ϕ) involved in a calculation [62]. In other words, to calculate an upper bound for the error based on Theorem S.3, we assume that $k \approx \sqrt{\phi}$. We also assume that the error in calculating $|\lambda - \hat{\lambda}|$ is negligible with respect to the error in $f(\cdot)$. Therefore, an estimate of the least upper bound can be calculated as follows:

$$\Big|f(\cdot) - \hat{f}(\cdot)\Big| \le \gamma_{\sqrt{\phi}} f(\cdot) := e_{\text{sup}}. \quad (5)$$

The problem with (5) is that in black-box optimization, we do not know the number of floating-point operations involved in calculating the objective function $f(\cdot)$. To overcome this difficulty, instead of finding the exact number of floating-point operations, we make some assumptions about the relationship between the dimensionality of the problem (n) and the number of floating-point operations (ϕ) that it may require. The simplest mapping of n variables into a scalar by a series of floating-point operations can be done by a simple summation, i.e., $f(\mathbf{x}) = \sum_{i=1}^{n} x_i$. In this example, the total number of floating-point operations is $\phi(n) = n - 1$. The dot-product of two n-dimensional vectors is another common example, i.e., $\mathbf{x} \cdot \mathbf{y} = \sum_{i=1}^{n} x_i y_i$. This calculation requires n multiplications and $n - 1$ additions. Therefore, the total number of operations required to calculate the dot-product of two n-dimensional vectors is $\phi(n) = 2n - 1$. At a higher level of abstraction, we can say that the number of operations involved in the calculation of these functions is of order $\mathcal{O}(n)$. An example of a function with floating-point complexity of order $\mathcal{O}(n^2)$ is $\sum_{i=1}^{n} x_i^n$. It turns out that these complexity classes constitute a large body of numerical operations. Matrix operations are among the most computationally expensive, whose complexity does not exceed $\mathcal{O}(n^3)$. Table S-I (in the supplementary material, Section S-III) contains a short list of common numerical operations and their complexity classes [62].

In an ideal situation, we need to find the least upper bound (e_{sup}) of the roundoff errors. If such a bound is available, any λ larger than e_{sup} can be treated as a genuine nonzero value. Generally speaking, the bounds calculated based on Theorem S.3 are very conservative, and the actual roundoff errors are much smaller in practice [61]. For example, Sterbenz's Theorem [63] states that $x \ominus y = x - y$ if $y/2 \leq x \leq 2y$. In other words, if two floating-point numbers are sufficiently close, their floating-pint subtraction is exact. Additionally, modern computers have a fused multiply add (FMA) instruction that involves a floating-point multiplication followed by an addition. Although FMA involves three floating-point operations, it commits only one rounding error in the worst case. We will also show in Section IV-D that underestimation of e_{sup} is not detrimental to the detection of interacting variables. In general, underestimation of e_{sup} results in accurate detection of interacting variables at the expense of missing some separable variables. Conversely, overestimation of e_{sup} results in high detection accuracy of separable variables at the expense of missing interacting variables. It is clear that the latter case is more detrimental to the optimization performance. Therefore, to get a tighter bound, we assume a linear complexity and define e_{sup} as follows:

$$e_{\text{sup}} = \gamma_{\sqrt{n}} \max\{f(\mathbf{x}), f(\mathbf{x}'), f(\mathbf{y}), f(\mathbf{y}')\}. \tag{6}$$

By estimating the least upper bound (e_{sup}) and the greatest lower bound (e_{inf}), we can identify reliable λ values. More specifically, all the λ values greater than e_{sup} will be treated as genuine nonzero (interacting variables), and all the values smaller than e_{inf} are treated as genuine zeros (separable variables). Finally, for the values in the range $(e_{\text{inf}}, e_{\text{sup}})$, the following weighted average of the bounds is used to set the threshold:

$$\epsilon = \frac{\eta_0}{\eta_0 + \eta_1} e_{\text{inf}} + \frac{\eta_1}{\eta_0 + \eta_1} e_{\text{sup}} \tag{7}$$

where η_0 is the number of entries in $\mathbf{\Lambda}$ which are less than e_{inf}, and η_1 is the number of entries in $\mathbf{\Lambda}$ which are greater than e_{inf}.

Algorithm 3: $\mathbf{\Theta} = \mathtt{DSM}(\mathbf{\Lambda}, \mathbf{F}, \check{\mathbf{f}}, f_{\text{base},n})$

```
1  $\mathbf{\Theta} = \mathtt{NaN}_{n\times n}$;
2  $\eta_1 = \eta_2 = 0$;
3  for $i = 1 \to n-1$ do
4      for $j = i+1 \to n$ do
5          $f_{\max} = \max\{f_{\text{base}}, \mathbf{F}_{ij}, \check{\mathbf{f}}_i, \check{\mathbf{f}}_j\}$;
6          $e_{\text{inf}} = \gamma_2 \cdot \max\{f_{\text{base}} + \mathbf{F}_{ij}, \check{\mathbf{f}}_i + \check{\mathbf{f}}_j\}$;
7          $e_{\text{sup}} = \gamma_{\sqrt{n}} \cdot f_{\max}$;
8          if $\mathbf{\Lambda}_{ij} < e_{\text{inf}}$ then
9              $\mathbf{\Theta}_{i,j} = 0$; $\eta_0 = \eta_0 + 1$;
10         else if $\mathbf{\Lambda}_{ij} > e_{\text{sup}}$ then
11             $\mathbf{\Theta}_{i,j} = 1$; $\eta_1 = \eta_1 + 1$;
12 for $i = 1 \to n-1$ do
13     for $j = i+1 \to n$ do
14         $f_{\max} = \max\{f_{\text{base}}, \mathbf{F}_{ij}, \check{\mathbf{f}}_i, \check{\mathbf{f}}_j\}$;
15         $e_{\text{inf}} = \gamma_2 \cdot \max\{f_{\text{base}} + \mathbf{F}_{ij}, \check{\mathbf{f}}_i + \check{\mathbf{f}}_j\}$;
16         $e_{\text{sup}} = \gamma_{\sqrt{n}} \cdot f_{\max}$;
17         if $\mathbf{\Theta}_{i,j} \neq \mathtt{NaN}$ then
18             $\epsilon = \frac{\eta_0}{\eta_0+\eta_1} \cdot e_{\text{inf}} + \frac{\eta_1}{\eta_0+\eta_1} \cdot e_{\text{sup}}$;
19             if $\mathbf{\Lambda}_{ij} > \epsilon$ then
20                 $\mathbf{\Theta}_{i,j} = 1$;
21             else
22                 $\mathbf{\Theta}_{i,j} = 0$;
```

Equation (7) is a natural choice for setting a threshold for non-reliable λ values. If $\eta_0 = \eta_1$, then (7) reduces to the arithmetic mean of e_{inf} and e_{sup}. This is intuitive, because when the number of reliably detected separable and nonseparable variables is equal, the middle point of the interval between e_{inf} and e_{sup} is the least biased choice of ϵ. Conversely, if the number of reliable calculations is skewed to one side, the threshold value should be biased to the same side. In the extreme case, if $\eta_0 = 0$, then $\epsilon = e_{\text{sup}}$. Similarly, if $\eta_1 = 0$, then $\epsilon = e_{\text{inf}}$.

Algorithm 3 contains the details of the DSM algorithm, in which the threshold on $\mathbf{\Lambda}$ is calculated by considering roundoff errors. The goal of Algorithm 3 is to convert the interaction structure matrix $\mathbf{\Lambda}$, which is calculated by ISM, into a binary design structure matrix ($\mathbf{\Theta}$) that represent variable interactions.

IV. Experimental Results and Analysis

In this section, we briefly compare the efficiency of DG2 with several state-of-the-art decomposition algorithms, namely DG [32], GDG [37], XDG [58], and CCVIL [33]. Next, we assess the grouping accuracy of DG2 on the CEC'2010 and the CEC'2013 large-scale benchmark suites, and compare it with several other state-of-the-art decomposition algorithms. Next, we use DG2 in a cooperative co-evolutionary framework to test its efficiency on the final optimization performance. Finally, we assess the sensitivity of DG2 to the imbalance level between the components of an objective function, and the assumptions about complexity of floating-point operation in black-box functions.

A. Comparative Analysis of Grouping Efficiency

According to Theorem S.1, DG2 requires the least number of function evaluations to detect *all* interactions as compared

TABLE I

GROUPING ACCURACY OF DG2, XDG, GDG, DG, AND CCVIL ON THE CEC'2010 AND 2013 LARGE-SCALE BENCHMARKS. ρ_1 MEASURES THE ACCURACY OF DETECTING INTERACTIONS, ρ_2 MEASURES THE ACCURACY OF DETECTING SEPARABLE VARIABLES, AND ρ_3 MEASURES THE OVERALL ACCURACY. DG2 GENERALIZES BETTER ON THE CEC'2013 BENCHMARKS AND OUTPERFORMS OTHER ALGORITHMS BY A WIDE MARGIN

		CCVIL			DG									GDG									XDG			DG2		
					$\epsilon = 10^{-3}$			$\epsilon = 10^{-3}$			$\epsilon = 10^{-6}$			$\alpha = 10^{-8}$			$\alpha = 10^{-9}$			$\alpha = 10^{-10}$			$\epsilon = 10^{-3}$			parameter-free		
Benchmarks	Statistics	ρ_1	ρ_2	ρ_3	ρ_1	ρ_2	ρ_3	ρ_1	ρ_2	ρ_3	ρ_1	ρ_2	ρ_3	ρ_1	ρ_2	ρ_3	ρ_1	ρ_2	ρ_3	ρ_1	ρ_2	ρ_3	ρ_1	ρ_2	ρ_3	ρ_1	ρ_2	ρ_3
CEC'2010	Median	3.5	100	99.7	100	100	100	100	100	100	100	91.6	92.8	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100
	Mean	2.6	99.9	93.4	95.5	97.5	97.5	99.9	93.4	93.7	100	87.7	88.4	100	98.5	98.5	100	95.0	95.2	100	93.7	94.1	99.9	77.0	78.2	100	90.2	90.7
	Std.	1.6	0.05	22.0	12.6	7.5	7.2	0.2	10.9	10.7	0.0	12.9	12.8	0.1	6.3	6.1	0.0	18.4	17.9	0.0	22.3	21.7	0.03	37.7	37.0	0.0	26.3	25.7
	Success rate		4			17			13			7			15			17			17			14			16	
CEC'2013	Median	2.60	100	98.3	100	100	99.0	100	99.5	98.8	100	89.7	91.0	82.1	100	99.5	93.9	100	99.8	95.5	100	99.8	100	85.1	98.8	100	100	100
	Mean	2.25	100	90.0	87.1	94.4	94.6	93.3	93.7	94.2	98.5	87.4	88.5	71.6	95.3	94.4	79.5	93.4	92.9	85.2	93.0	92.8	97.8	56.5	61.4	97.5	89.3	90.0
	Std.	1.08	0	25.1	20.8	9.4	8.6	11.7	9.3	8.6	4.6	11.0	10.8	30.5	17.5	16.7	27.6	24.7	23.7	23.1	26.4	25.3	4.9	48.2	45.0	8.4	28.9	27.9
	Success rate		4			4			5			4			4			5			5			6			9	

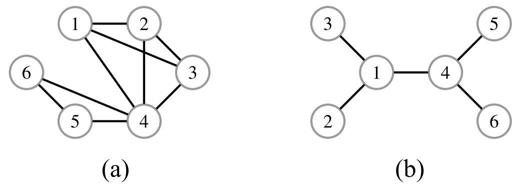

Fig. 2. Interaction structures represented by (a) and (b) cannot be distinguished by XDG.

to other decomposition algorithms. The total number of function evaluations needed by DG2 is constant and is equal to $[(n^2 + n + 2)/2]$ for an n-dimensional problem (Theorem S.1). The GDG algorithm requires $[(n^2 + 3n + 2)/2]$ evaluations [37], which is larger than what is needed by DG2 to construct the entire interaction structure matrix. Unlike DG2 and GDG, XDG does not construct a full interaction structure matrix and cannot identify the overlapping functions. If XDG detects that variables x_i and x_j both interact with a common variable x_k, it does not check the interaction between x_i and x_j explicitly. Therefore, XDG fails to distinguish between the interaction structures represented by the graphs shown in Fig. 2. For example, if XDG learns that variables x_2–x_4 all interact with x_1, it will assume that the following pairs also interact: (x_2, x_3), (x_3, x_4), and (x_2, x_4). This can have implications on decomposition of overlapping functions. XDG uses this strategy to reduce the number of function evaluations in the detection phase; however, it still requires slightly less than $n^2 + n$ function evaluations, which is significantly more than what is needed by DG2 [58]. DG also does not have the ability of detecting overlapping functions. Even if the algorithm is modified to check all pairs of variables, it would require $n^2 + n$ function evaluations.

B. Comparative Analysis of Grouping Accuracy

Table I contains the summary statistics for the grouping accuracy of CCVIL, DG, GDG, XDG, and DG2 on the CEC'2010 and the CEC'2013 large-scale benchmark suites. In this paper, we use the metric proposed by Mei *et al.* [37], which consists of three measures: 1) ρ_1 (interaction); 2) ρ_2 (independence); and 3) ρ_3 (interaction and independence). The statistics are taken over all the functions in each benchmark suite. The detailed results for individual functions can be found in Tables S-II and S-III (in the supplementary material, Section S-IV). The success rate indicates the number of functions for which the correct decomposition is identified. The overlapping functions are not counted since their optimal decomposition is unknown. It should be noted that XDG and CCVIL start with an interaction structure matrix of all zeros ($\boldsymbol{\Theta} = \mathbf{0}$; full separability assumption). Therefore, if a pair of variables are not checked for interaction, the relevant entry of $\boldsymbol{\Theta}$ assumes its default value for the calculations of the ρ-metrics.

Table I shows that DG2 outperforms all other decomposition algorithms on the CEC'2010 and CEC'2013 suites. It is notable that the difference is more pronounced on the more difficult CEC'2013 benchmark suite. The performance of DG2 appears to be slightly lower than some variants of GDG according to ρ_2 (measure of independence). However, this is caused by three instances of the Ackley function, which affects the mean values in Table I. This behavior can be seen in Tables S-II and S-III (in the supplementary material). It should be noted that the Ackley function is not *additively separable* [64], which is correctly identified by DG2. However, in the benchmark suites these functions are reported as separable according to Definition 1.

Overall, DG2 shows better generalizability over a wider range of functions than all other decomposition algorithms. CCVIL shows the worst performance, while DG shows the strongest sensitivity to its control parameter (ϵ), especially on the CEC'2010 benchmarks. This can be attributed to its static choice of ϵ. The problem with this approach is that it ignores the fact that the magnitude of the computational error in $\lambda = |\Delta^{(1)} - \Delta^{(2)}|$ is correlated with the magnitude of the objective function. Therefore, on some functions, when ϵ is smaller than the inherent computational errors, some separable variables will be considered as nonseparable. This is why increasing ϵ generally results in a lower ρ_1 and a higher ρ_2 for DG. Unlike DG, GDG sets ϵ proportional to the magnitude of the objective function. This is based on the rationale that a higher objective function value results in a high computational

TABLE II
ϵ VALUES OF GDG AND DG2 ON SELECTED FUNCTIONS. GDG SYSTEMATICALLY OVERESTIMATES THE COMPUTATIONAL ERROR WHICH RESULTS IN LARGE ϵ VALUES AS COMPARED TO DG2

	GDG			DG2		
Function	$\alpha = 10^{-8}$	$\alpha = 10^{-9}$	$\alpha = 10^{-10}$	min	median	max
f_1	8.93e+03	8.93e+02	8.93e+01	4.06e-04	4.16e-04	4.18e-04
f_4	2.75e+06	2.75e+05	2.75e+04	2.34e-01	3.14e-01	3.78e-01
f_8	1.94e+11	1.94e+10	1.94e+09	1.76e+04	2.37e+04	3.48e+04
f_{13}	8.11e+12	8.11e+11	8.11e+10	1.64e+06	2.72e+06	1.20e+07
f_{15}	1.19e+04	1.19e+03	1.19e+02	1.25e-02	1.25e-02	1.27e-01

error in λ. Equation (8) is a very simple way of choosing ϵ proportional to the computational error

$$\epsilon = \alpha \cdot \min\{f(\mathbf{x}_1), \ldots, f(\mathbf{x}_k)\} \tag{8}$$

where $\mathbf{x}_1, \ldots, \mathbf{x}_k$ are k random sample points.

Table I clearly shows that the method used by GDG is effective on the CEC'2010 benchmarks, but does not generalize well on the CEC'2013 benchmarks. To understand the reason for this behavior, the ϵ values calculated by GDG and DG2 for selected functions are reported in Table II.[4] Since DG2 uses a different ϵ to detect interaction between each pair of variables, we report the overall mean and the median of all ϵ values as indicators. Table II clearly shows that GDG systematically overestimates the computational error which results in large ϵ values. By comparing (3) and (6) with (8), we can see that the calculation of ϵ in both DG2 and GDG is a function of the objective function value. However, GDG differs in two major ways. First, the constant $\alpha = \{10^{-8}, 10^{-9}, 10^{-10}\}$ is significantly larger than both γ_2 ($\approx 2.2204 \times 10^{-16}$) and $\gamma_{\sqrt{n}}$ ($\approx 3.5108 \times 10^{-15}$ for $n = 1000$), which results in overestimation of ϵ by GDG. Second, k sample objective function values used by GDG to detect *all* interactions, whereas in DG2 the quantities used in (3) and (6) are only those which are involved in the calculation of λ for a particular pair of variables. These differences contribute to the overestimation of ϵ by GDG which explains its high accuracy of detecting separable variables (ρ_2) at the expense of a low interaction detection accuracy (ρ_1). When ϵ is set to a large number, the algorithm has a tendency to classify most variables as fully separable. Conversely, a high value for ϵ makes the algorithm insensitive to weak interactions. Therefore, the algorithm may treat many weakly interacting variables as fully separable. This behavior is magnified on the CEC'2013 benchmarks due to the imbalance in the contribution of each component to the overall objective value.

C. Optimization Results

In this section, we investigate the effectiveness DG2 when it is used as a decomposition algorithm within a cooperative co-evolutionary framework. The empirical results are based on the CEC'2013 benchmark suite [65]. Finally, we show that in conjunction with an accurate decomposition, a contribution-based cooperative co-evolutionary algorithm shows comparable results to the state-of-the-art algorithms.

[4]Also see Table S-IV in the supplementary material.

TABLE III
PERFORMANCE COMPARISON OF DG2 AGAINST DG, XDG, CCVIL, AND IDEAL GROUPING ON THE CANONICAL COOPERATIVE CO-EVOLUTION AND A CONTRIBUTION-BASED COOPERATIVE CO-EVOLUTIONARY FRAMEWORK. DG2'S NUMBER OF WINS, TIES, AND LOSSES AGAINST OTHER DECOMPOSITION METHODS IS REPORTED. DG2 OUTPERFORMS OTHER DECOMPOSITION METHODS AND CAN PERFORM AS WELL AS THE IDEAL DECOMPOSITION ON MOST FUNCTIONS

	Canonical CC				Contribution-based CC			
	DG w/t/l	XDG w/t/l	CCVIL w/t/l	Ideal w/t/l	DG w/t/l	XDG w/t/l	CCVIL w/t/l	Ideal w/t/l
C1: f_1-f_3	1/2/0	2/1/0	0/1/2	0/1/2	1/2/0	3/0/0	2/0/1	0/1/2
C2: f_4-f_7	3/1/0	2/1/1	3/0/1	0/3/1	3/1/0	1/2/1	4/0/0	0/3/1
C3: f_8-f_{11}	2/1/1	2/0/2	3/0/1	0/4/0	2/1/1	1/2/1	3/1/0	0/0/4
C4: f_{12}-f_{15}	2/0/2	1/0/3	2/0/2	0/1/3	2/0/2	1/0/3	2/0/2	0/0/4
Total	8/4/3	7/2/6	8/1/6	0/9/6	8/4/3	6/4/5	11/1/3	0/4/11

1) Performance Comparison of Decomposition Methods: Table III contains the summary of the experimental results to compare the performance of DG2, DG, XDG, CCVIL, and ideal grouping, within a co-evolutionary framework.[5] We use two different co-evolutionary frameworks: 1) the canonical cooperative co-evolution framework in which all components are optimized in a round-robin fashion and 2) a contribution-based framework in which components with higher contribution to the overall solution quality are given more resources (based on the CBCC3 algorithm [66]). In this paper, the p_t parameter of CBCC3 is set to zero. The component optimizer of both frameworks is SaNSDE [67], and the population size of all algorithms is set to 50 as suggested by Yang *et al.* [67]. The maximum number of fitness evaluations is set to 3×10^6 as suggested by Li *et al.* [65]. Except for ideal grouping, the number of objective function evaluations used in the decomposition stage is deducted from the maximum available evaluations (a complete table is included in the supplementary material). All experimental results are based on 25 independent runs. To test the statistical significance of the results, DG and ideal grouping are compared with the baseline (DG2) using a two-tailed Wilcoxon rank-sum test with $\alpha = 0.05$.

Table III clearly shows that DG2 has an overall better performance than the other decomposition methods when it is used in a cooperative co-evolutionary framework. This is the case on both the canonical cooperative co-evolution and the contribution-based cooperative co-evolution frameworks. When comparing DG2 with ideal grouping, we can see that ideal grouping performs better on six functions and performs statistically similar on nine functions, when the canonical cooperative co-evolution is used. It should be noted that the comparison between the ideal grouping and DG2 is unfair because the ideal grouping is manually given to optimization algorithm, which results in it having access to 500 501 extra function evaluations. In spite of this difference, Table III shows that DG2 managed to perform as well as the ideal case on nine functions (60% of the functions). This difference is even tighter on partially separable functions (f_4–f_{11}) where DG2 performs worse than the ideal case on only one function. This clearly shows the benefit of first using some

[5]Also see Table S-VIII in the supplementary material.

TABLE IV
PERFORMANCE OF CONTRIBUTION-BASED AND CANONICAL COOPERATIVE CO-EVOLUTION USING DIFFERENT DECOMPOSITION METHODS. THE NUMBER OF WINS, TIES, AND LOSSES OF THE CONTRIBUTION-BASED FRAMEWORK AGAINST CANONICAL COOPERATIVE CO-EVOLUTION IS REPORTED. DG2 HAS THE BEST IMPROVING EFFECT ON THE CONTRIBUTION-BASED FRAMEWORK (AS WELL AS IDEAL GROUPING), ESPECIALLY ON THE PARTIALLY SEPARABLE FUNCTIONS (C2 AND C3)

Categories	CCVIL w/t/l	DG w/t/l	DG2 w/t/l	XDG w/t/l	Ideal w/t/l
C1: f_1-f_3	1/1/1	0/3/0	0/3/0	0/3/0	0/3/0
C2: f_4-f_7	1/0/3	1/1/2	4/0/0	2/0/2	4/0/0
C3: f_8-f_{11}	3/1/0	2/1/1	4/0/0	3/0/1	4/0/0
C4: f_{12}-f_{15}	2/2/0	2/2/0	0/4/0	0/4/0	0/4/0
Total	7/4/4	5/7/3	8/7/0	5/7/3	8/7/0

portion of the available computational resources to find an accurate decomposition of the problem before carrying out optimization. Unlike the canonical cooperative co-evolution, the difference of DG2 and the ideal grouping is wider on a contribution-based framework. Table III shows that DG2 is outperformed by the ideal grouping on 11 out of 15 functions. Since the final grouping of DG2 and the ideal grouping is identical for most of the functions, the difference can be attributed to the extra function evaluations which is available to the ideal case.

2) How the Grouping Accuracy Affects the Contribution-Based Cooperative Co-Evolution: To investigate the effect of an accurate decomposition on the performance of contribution-based cooperative co-evolution, we compare the standard round-robin cooperative co-evolution with its contribution-based counterpart across different grouping algorithms. The results of pair-wise Wilcoxon rank-sum tests are summarized in Table IV. We can see that the contribution-based framework generally performs better than the canonical cooperative co-evolution; however, there is a performance loss when DG is used as the decomposition algorithm. When DG2 and ideal grouping are used, the contribution-based framework outperforms the canonical cooperative co-evolution on eight functions and performs statistically similar on seven functions, whereas with DG the number of wins is reduced to 5 and the number of losses is increased to 3. It is notable that the overall behavior of XDG is similar to that of DG. It is also interesting to note that despite its low grouping accuracy, CCVIL benefited from the use a contribution-framework.

A closer look at Table IV shows that DG2 achieves most of the ties on fully separable functions (C1) and overlapping and nonseparable functions (C4) for which no decomposition is done. Although DG2 can find the entire interaction structure matrix, no decomposition is performed because unlike partially separable functions, the decomposition of these functions is not unique. Therefore, both the contribution-based and the canonical cooperative co-evolution frameworks reduce to SaNSDE which is the component optimizer of both frameworks. An interesting exception is the behavior of DG and CCVIL on C4, where the contribution-based framework outperforms the canonical cooperative co-evolution framework on two cases. This is not the case for DG2, XDG, and the ideal grouping. It should be noted that DG and CCVIL decompose some functions in that category into smaller components, all of which are overlapping functions. This shows that decomposition of overlapping functions can be beneficial. This observation suggests that, by using DG2 we can learn the exact interaction pattern of the variables and identify the shared decision variables between the components in order to devise an effective decomposition for overlapping functions. However, with DG and CCVIL this is done arbitrarily depending on the order in which the variables are visited and their interaction pattern.

On the partially separable functions (C2 and C3), the contribution-based framework outperforms the canonical cooperative co-evolution when DG2 and the ideal grouping were used. This is not the case with other decomposition algorithms. Overall, the results in Table IV show that decomposition accuracy can affect the optimization performance. The experimental results suggest that the contribution-based framework requires relatively accurate decomposition in order to estimate the contribution of each component. This observation is consistent with the sensitivity analysis conducted by Kazimipour *et al.* [69]. In general, the contribution-based family of algorithms are sensitive to grouping noise, but in the worst case they preform as well as the canonical cooperative co-evolution which makes them a safe choice for black-box problems [69]. It should be noted that, Kazimipour *et al.* [69] used uniform grouping noise in their study, which equally affects both strong and weak interactions. However, we learned in Section IV-B that DG2's grouping error is mostly attributed to detecting weakly interacting variables. This suggests that 100% accuracy is not essential in order to benefit from a divide-and-conquer scheme, but it is important on which variables does the decomposition algorithm commits the errors. An example of such a case is f_8 for which DG2 treated two components with weakly interacting variables as fully separable (Table S-V in the supplementary material). However, this did not affect the overall optimization performance as reflected in Table III. Further analysis of this case is given in the supplementary material (Section S-V).

3) Comparison With the State-of-the-Art: Finally, we compare the performance of the contribution-based cooperative co-evolution that uses DG2 as its decomposition method with some well-known algorithms, such as multiple offspring framework (MOS) [21], MA-SW-Chains [22], and CMA-ES [68]. The parameter settings of these algorithms match the reported values in the original papers. MOS and MA-SW-Chains ranked first in the CEC'2013 and CEC'2010 competition on large-scale optimization, respectively. Table V contains the experimental results using 25 independent runs on f_4–f_{11} from the CEC'2013 large-scale benchmark suite [65]. For this comparison, we have focused on the partially separable functions. This is because no decomposition is done for f_1–f_3 (fully separable) and f_{12}–f_{15} (overlapping), in which case CBCC3-DG2 reduces to SaNSDE. It should be noted that DG2 managed to discover the underlying variable interaction structure of these functions. Although some preliminary studies focused on the effect of decomposition on fully separable and nonseparable functions [34], [70], the optimal decomposition

TABLE V
EXPERIMENTAL RESULTS OF THE CONTRIBUTION-BASED FRAMEWORK WITH DG2 (CBCC3-DG2), MOS, AND CMA-ES ON THE CEC'2013 LARGE-SCALE BENCHMARK SUITE USING 25 INDEPENDENT RUNS. THE HIGHLIGHTED ENTRIES ARE SIGNIFICANTLY BETTER (WILCOXON RANK-SUM TEST WITH HOLM p-VALUE CORRECTION, $\alpha = 0.05$). DG2 ALLOWS THE CONTRIBUTION-BASED FRAMEWORK TO PERFORM AS WELL AS THE STATE-OF-THE-ART EVEN USING A MEDIOCRE COMPONENT OPTIMIZER

	stats.	CBCC3-DG2	MOS [21]	CMA-ES [68]	MA-SW-Chains [22]
f_4	Median	**2.77e+07**	1.56e+08	4.10e+08	4.27e+09
	Mean	3.39e+07	1.74e+08	4.30e+08	4.58e+09
	StDev	1.77e+07	8.02e+07	1.17e+08	2.51e+09
f_5	Median	**2.11e+06**	6.79e+06	**2.06e+06**	**1.81e+06**
	Mean	2.14e+06	6.94e+06	2.04e+06	1.87e+06
	StDev	4.24e+05	9.03e+05	2.64e+05	3.13e+05
f_6	Median	1.05e+06	**1.39e+05**	6.09e+05	1.01e+06
	Mean	1.05e+06	1.48e+05	6.01e+05	1.01e+06
	StDev	3.37e+03	6.56e+04	1.28e+05	1.56e+04
f_7	Median	2.94e+07	1.62e+04	**6.83e+02**	3.92e+06
	Mean	2.95e+07	1.62e+04	3.00e+03	3.45e+06
	StDev	2.78e+07	9.29e+03	5.23e+03	1.29e+06
f_8	Median	**1.41e+10**	8.08e+12	1.00e+13	4.90e+13
	Mean	4.28e+10	8.00e+12	1.13e+13	4.85e+13
	StDev	8.86e+10	3.13e+12	6.08e+12	1.04e+13
f_9	Median	1.68e+08	3.87e+08	1.74e+08	**1.08e+08**
	Mean	1.70e+08	3.83e+08	1.80e+08	1.07e+08
	StDev	3.16e+07	6.42e+07	2.28e+07	1.71e+07
f_{10}	Median	9.30e+07	**1.18e+06**	1.42e+07	9.18e+07
	Mean	9.28e+07	9.01e+05	1.64e+07	9.18e+07
	StDev	7.16e+05	5.17e+05	1.44e+07	1.08e+06
f_{11}	Median	5.83e+08	4.48e+07	**7.72e+06**	2.15e+08
	Mean	6.59e+08	5.22e+07	9.37e+06	2.19e+08
	StDev	2.80e+08	2.10e+07	6.53e+06	3.04e+07

of these categories of functions is an open question beyond the scope of this paper.

Table V shows that no single algorithm outperforms other algorithms. It is notable that on f_8, on which two of the weakly interacting variables were grouped as separable, CBCC3-DG2 performs the best. The results indicated that although CBCC3-DG2 uses SaNSDE which is not a competitive optimizer as compared to MOS, MA-SW-Chains, or CMA-ES, a contribution-based framework with an accurate decomposition can make it comparable with the state-of-the-art. It has been shown that a cooperative co-evolutionary framework can scale up the performance of many optimizers, such as particle swarm optimization, evolution strategies, differential evolution, and evolution programs [31], [37], [53], [55], [71]. In this paper, we have also established the efficacy of DG2 against other decomposition algorithms. We believe that as a general and effective decomposition method, DG2 can be used with other promising large-scale optimization algorithms, such as MOS and MA-SW-Chains, to further boost their performance. This will be the subject of our future work.

D. Sensitivity Analysis of DG2

It was mentioned in Section III-B that the exact calculation of the least upper bound (e_{sup}) is not possible due to the black-box nature of the objective function. To alleviate this problem, we proposed to estimate the number of floating-point operations based on assumptions about the complexity class of the objective function. We argued that most of the numerical calculations that do not involve complex matrix operations are of order $\mathcal{O}(n^2)$. Additionally, error cancellations, subtraction of close numbers (Sterbenz's Theorem [63]), and the FMA operation make the actual computational error much lower than the worst case scenario. In the previous section, we assumed a linear complexity. Here, we provide empirical results based

TABLE VI
SENSITIVITY ANALYSIS OF DG2 ON VARIOUS COMPLEXITY CLASSES ON THE CEC'2013 LARGE-SCALE BENCHMARK SUITE. DG2 BEHAVES SIMILARLY WHEN GROWTH RATE OF FLOATING-POINT OPERATIONS IS NOT OVERESTIMATED (LINEAR AND QUADRATIC CASES). HOWEVER, DG2 STARTS TO OVERESTIMATE THE ROUNDOFF ERRORS WHEN A CUBIC GROWTH IS ASSUMED, WHICH CAUSES IT TO TREAT WEAKLY INTERACTING VARIABLES AS SEPARABLE

	$\mathcal{O}(n)$				$\mathcal{O}(n^2)$				$\mathcal{O}(n^3)$			
Fun.	ρ_1	ρ_2	ρ_3		ρ_1	ρ_2	ρ_3		ρ_1	ρ_2	ρ_3	
f_1	—	100	100	✓	—	100	100	✓	—	100	100	✓
f_2	—	100	100	✓	—	100	100	✓	—	100	100	✓
f_3	—	0	0	✕	—	0.0016	0.0016	✕	—	0.08	0.08	✕
f_4	100	100	100	✓	100	100	100	✓	100	100	100	✓
f_5	99.97	100	100	✓	99.93	100	99.99	✓	98.69	100	99.97	✓
f_6	99.98	50.45	51.30	✕	99.97	54.56	55.35	✕	99.77	68.86	69.40	✕
f_7	100	100	100	✓	100	100	100	✓	100	100	100	✓
f_8	70.72	100	98.01	✕	70.07	100	97.97	✕	63.58	100	97.53	✕
f_9	99.99	100	100	✓	99.99	100	100	✓	99.78	100	99.98	✓
f_{10}	99.93	100	99.99	✓	99.07	100	99.93	✓	85.56	100	99.02	✕
f_{11}	99.95	100	99.99	✓	99.47	100	99.96	✓	97.72	100	99.84	✕
f_{12}	100	100	100	–	100	100	100	–	100	100	100	–
f_{13}	100	100	100	–	100	100	100	—	99.99	100	100	–
f_{14}	99.97	100	99.99	–	99.75	100	99.98	—	99.12	100	99.92	–
f_{15}	100	—	100	✓	100	—	100	✓	100	—	100	✓
Success rate				9				9				7
Median	100	100	100		100	100	100		99.8	100	100	
Mean	97.5	89.3	90.0		97.4	89.6	90.2		95.4	90.6	91.0	
Std.	8.4	28.9	27.9		8.6	28.5	27.5		10.8	27.4	26.4	

TABLE VII
DETAILED GROUPING MATRIX OF DG2 ON f_{10} BASED ON $\mathcal{O}(n^3)$ FOR ESTIMATING e_{sup}. THE ROWS INDICATE THE GROUPS FORMED BY DG2 AND THE COLUMNS REPRESENT THE PERMUTATION GROUPS FROM WHICH THE VARIABLES IN EACH GROUP WERE EXTRACTED. P13 IS A COMPONENT WITH WEAKLY INTERACTING VARIABLES THAT IS NOT IDENTIFIED BY DG2 PROPERLY DUE TO THE CUBIC ASSUMPTION

	Group	P1	P2	P3	P4	P5	P6	P7	P8	P9	P10	P11	P12	P13	P14	P15	P16	P17	P18	P19	P20
Groups	Size	50	50	25	25	100	100	25	25	50	25	100	25	100	50	25	25	25	100	50	25
G01	50	50	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
G02	100	0	0	0	0	0	100	0	0	0	0	0	0	0	0	0	0	0	0	0	0
G03	25	0	0	0	25	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
G04	100	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	100	0	0
G05	100	0	0	0	0	0	0	0	0	0	0	100	0	0	0	0	0	0	0	0	0
G06	25	0	0	25	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
G07	100	0	0	0	0	100	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
G08	55	0	0	0	0	0	0	0	0	0	0	0	0	55	0	0	0	0	0	0	0
G09	50	0	50	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
G10	25	0	0	0	0	0	0	0	0	0	0	0	25	0	0	0	0	0	0	0	0
G11	25	0	0	0	0	0	0	0	25	0	0	0	0	0	0	0	0	0	0	0	0
G12	50	0	0	0	0	0	0	0	0	50	0	0	0	0	0	0	0	0	0	0	0
G13	25	0	0	0	0	0	0	0	0	0	25	0	0	0	0	0	0	0	0	0	0
G14	25	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	25	0	0	0
G15	25	0	0	0	0	0	0	0	0	0	0	0	0	0	0	25	0	0	0	0	0
G16	50	0	0	0	0	0	0	0	0	0	0	0	0	0	50	0	0	0	0	0	0
G17	50	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	50	0
G18	25	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	25
G19	25	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	25	0	0	0	0
G20	6	0	0	0	0	0	0	0	0	0	0	0	0	6	0	0	0	0	0	0	0
G21	25	0	0	0	0	0	0	25	0	0	0	0	0	0	0	0	0	0	0	0	0
G22	2	0	0	0	0	0	0	0	0	0	0	0	0	2	0	0	0	0	0	0	0
G23	2	0	0	0	0	0	0	0	0	0	0	0	0	2	0	0	0	0	0	0	0
Total	965	# separable variables = 1000 - 965 = 35																			

TABLE VIII
SENSITIVITY ANALYSIS OF DG2 TO VARIOUS IMBALANCE LEVELS (CEC'2013). DG2 TOLERATES LOW, MEDIUM, AND HIGH IMBALANCE LEVELS; HOWEVER, ITS GROUPING ACCURACY DROPS WHEN THE IMBALANCE LEVEL IS EXTREME ($10^{4\mathcal{N}(0,1)}$)

	$w_i = 1$				$w_i = 10^{\mathcal{N}(0,1)}$				$w_i = 10^{2\mathcal{N}(0,1)}$				$w_i = 10^{3\mathcal{N}(0,1)}$				$w_i = 10^{4\mathcal{N}(0,1)}$			
Fun.	ρ_1	ρ_2	ρ_3		ρ_1	ρ_2	ρ_3		ρ_1	ρ_2	ρ_3		ρ_1	ρ_2	ρ_3		ρ_1	ρ_2	ρ_3	
f_4	100	100	100	✓	100	100	100	✓	100	100	100	✓	100	100	100	✓	100	100	100	✓
f_5	100	100	100	✓	100	100	100	✓	100	100	100	✓	99.97	100	99.99	✓	100	100	100	✓
f_6	100	50.16	51.02	×	100	50.16	51.02	×	100	50.22	51.08	×	99.98	50.45	51.30	×	99.98	50.30	51.15	×
f_7	100	100	100	✓	100	100	100	✓	100	100	100	✓	100	100	100	✓	96.64	100	99.94	×
f_8	100	100	100	✓	100	100	100	✓	99.99	100	99.99	✓	70.72	100	98.01	×	95.85	100	99.71	×
f_9	100	100	100	✓	100	100	100	✓	100	100	100	✓	99.99	100	99.99	✓	78.43	100	98.53	×
f_{10}	100	100	100	✓	100	100	100	✓	100	100	100	✓	99.93	100	99.99	✓	80.28	100	98.66	×
f_{11}	100	100	100	✓	100*	100	100*	×*	100	100	100	✓	99.95	100	99.99	✓	80.34	100	98.66	×
f_{13}	100	100	100	–	100	100	100	–	100	100	100	–	100	100	100	—	97.94	100	99.83	–
f_{14}	100	100	100	–	100	100	100	–	100	99.99	99.99	–	99.97	100	99.99	—	99.85	100	99.98	–
Success rate				7				6				7				6				2
Mean	100	95.0	95.1		100	95.0	95.1		100.0	95.0	95.1		97.1	95.0	94.9		92.9	95.0	94.6	
Median	100	100	100		100	100	100		100	100	100		100	100	100		97.3	100	99.8	
Std.	0.0	15.8	15.5		0.0	15.8	15.5		0.0	15.7	15.5		9.3	15.7	15.3		9.3	15.7	15.3	

* The values are not exact due to rounding. Two interactions are miscalculated which cause a small error. Hence, the final grouping does not match the ideal case.

on quadratic and cubic complexity classes to investigate the robustness of DG2 with respect to deviations from our initial assumption.

1) Sensitivity to Complexity of the Objective Function: Table VI shows the grouping accuracy of DG2 based on different complexity classes. We can see that DG2 behaves similarly when linear and quadratic complexity classes are assumed for the number of floating-point operations. However, when a cubic complexity class is assumed, the grouping accuracy drops. From Section III-B, we know that the assumption about floating-point complexity class of the function affects the least upper bound (e_{sup}). Table VI shows that the cubic complexity class causes overestimation of e_{sup}, which affected the grouping accuracy of DG2 on f_{10} and f_{11}. Overestimation of roundoff errors will cause DG2 to treat weakly interacting variables as separable. The detailed grouping matrix of f_{10}, as shown in Table VII, reveals that DG2 detected 35 separable variables which mostly belong to P13. Table VII shows that P13 contains 100 variables 55 of which are detected in G08, 35 of which are considered to be fully separable, and the remaining ten variables are grouped into three smaller groups (G20, G22, and G23). It is interesting to note that P13 has the lowest weight (6.81×10^{-5}) among all other components in the CEC'2013 large-scale benchmark suite. The function f_{11} behaves in a similar way, but we do not include the details for the sake of brevity. Overall, Table VI shows that DG2 is not susceptible to moderate overestimation of e_{sup}, but underestimation of e_{sup} is less detrimental to its performance.

2) Sensitivity to the Imbalance Level: Next, we analyze the sensitivity of DG2 with respect to imbalance level among the components of the benchmark functions. The functions f_4–f_{11} and f_{13}–f_{14} have the following general form: $\sum_{i=1}^{m} w_i f_i(\mathbf{x}_i)$, where $w_i = 10^{c\mathcal{N}(0,1)}$. The parameter c is a constant that determines the variance among the weights. In the CEC'2013 large-scale benchmark suite, c is set to 3. For our sensitivity analysis, we tested the performance of DG2 with $c \in \{0, 1, 2, 3, 4\}$, the result of which is reported in Table VIII. The table shows that the overall grouping accuracy of DG2 is stable with various imbalance levels, except when $c = 4$. It is notable that the detection accuracy of separable variables (ρ_2) is very high and stable across various imbalance levels. However, the detection accuracy of interacting variables (ρ_1) drops when the imbalance level increases. Our detailed analysis on f_4–f_{11} for $c = 4$ showed that the nonseparable components which are missed by DG2 are always among the components with the smallest weight. For example, the detailed grouping matrix of f_9 with $c = 4$ (Table IX) shows that the missing components (P3, P4, P13, P19, and P20) are the top five components with the smallest weights. For simplicity, the log of the weights associated to each component is shown at the top of each column. Other functions have a similar behavior, but we do not include them in the analysis for the sake of brevity. Overall, this analysis shows that DG2 is not sensitive to moderate imbalance levels.

TABLE IX
DETAILED GROUPING MATRIX OF DG2 ON f_9 FOR $w_i = 10^{4\mathcal{N}(0,1)}$. THE ROWS INDICATE THE GROUPS FORMED BY DG2 AND THE COLUMNS REPRESENT THE PERMUTATION GROUPS FROM WHICH THE VARIABLES IN EACH GROUP WERE EXTRACTED. WHEN THE IMBALANCE LEVEL IS EXTREME, DG2 ONLY MISSES COMPONENTS WITH *Weakly* INTERACTING VARIABLES (P3, P4, P13, P19, AND P20)

	Group	P1	P2	P3	P4	P5	P6	P7	P8	P9	P10	P11	P12	P13	P14	P15	P16	P17	P18	P19	P20
Groups	Size	50	50	25	25	100	100	25	25	50	25	100	25	100	50	25	25	25	100	50	25
$\lfloor \log w_i \rfloor$		1	-3	-6	-7	-1	-1	-2	7	2	-3	1	0	-6	3	10	3	-1	4	-5	-7
G01	50	0	50	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
G02	100	0	0	0	0	0	100	0	0	0	0	0	0	0	0	0	0	0	0	0	0
G03	25	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	25	0	0	0
G04	50	0	0	0	0	0	0	0	0	0	0	0	0	0	50	0	0	0	0	0	0
G05	100	0	0	0	0	0	0	0	0	0	0	100	0	0	0	0	0	0	0	0	0
G06	50	50	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
G07	100	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	100	0	0
G08	100	0	0	0	0	100	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
G09	25	0	0	0	0	0	0	25	0	0	0	0	0	0	0	0	0	0	0	0	0
G10	25	0	0	0	0	0	0	0	0	0	25	0	0	0	0	0	0	0	0	0	0
G11	25	0	0	0	0	0	0	0	0	0	0	0	25	0	0	0	0	0	0	0	0
G12	25	0	0	0	0	0	0	0	0	0	0	0	0	0	0	25	0	0	0	0	0
G13	50	0	0	0	0	0	0	0	0	50	0	0	0	0	0	0	0	0	0	0	0
G14	25	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	25	0	0	0	0
G15	25	0	0	0	0	0	0	0	25	0	0	0	0	0	0	0	0	0	0	0	0
G16	4	0	0	0	0	0	0	0	0	0	0	0	0	4	0	0	0	0	0	0	0
Total	779	# separable variables = 1000 - 779 = 221																			

When the imbalance level is very high, the inaccuracy of DG2 comes from considering very weakly interacting variables as fully separable. It should be noted that the generated weights when $c = 4$ are very extreme and rarely occur in real-world scenarios. Nonetheless, if this happens, treating very weakly interacting variables as fully separable is not detrimental to the optimization performance as we saw in the previous section.

V. Conclusion

In this paper, we proposed an improved version of the DG algorithm. This new algorithm, DG2, has the following major advantages over its predecessor.

1) *Efficiency:* Lower computational cost, especially on fully separable functions.
2) *Accuracy:* Higher interaction detection accuracy.
3) *Robustness:* Lower sensitivity to computational roundoff errors.
4) *Applicability:* The ability to detect objective functions with overlapping components, i.e., components that share decision variables. This makes it applicable to a wide array of continuous functions.
5) *Practicality:* No need for the user to specify a threshold parameter (ϵ); in other words, DG2 is parameter-free.

With respect to efficiency, we have shown mathematically that DG2 achieves the lower bound on the total number of function evaluations needed to test all pairs of variables for interaction. This effectively reduces the total number of required function evaluations by half. In addition to the improvements on efficiency, DG2 uses the information that is calculated in the process of applying the DG theorem to estimate a reliable threshold value (ϵ) that takes the computational error into account. The experimental results showed that DG2 significantly outperforms its predecessor on the CEC'2010 and the CEC'2013 large-scale benchmark suites.

Finally, we have shown empirically that in conjunction with DG2, the contribution-based cooperative co-evolution performs as well as the top performers of the CEC'2010 and CEC'2013 competition on large-scale optimization, as well as the well-known CMA-ES, on partially separable functions.

DG2 can also detect overlapping functions and can return a complete interaction structure matrix. However, due to the use of the connected components algorithm, it returns a single group containing all the decision variables. This limits the optimizer from exploiting the structural information that is found by DG2. Potential future research can focus on finding an effective decomposition for overlapping functions. Lack of a unique optimal decomposition for overlapping functions makes this a challenging task.

Acknowledgment

The authors would like to thank K. Kouhkiloui for implementing the C++ version of the DG2 algorithm.

Software Implementation

The MATLAB/Octave and C++ implementations of the DG2 algorithm can be accessed from the following link: https://bitbucket.org/mno/differential-grouping2.

References

[1] M. Lozano, D. Molina, and F. Herrera, "Editorial scalability of evolutionary algorithms and other metaheuristics for large-scale continuous optimization problems," *Soft Comput.*, vol. 15, no. 11, pp. 2085–2087, Nov. 2011.

[2] X. Li, K. Tang, P. N. Suganthan, and Z. Yang, "Editorial for the special issue of information sciences journal (ISJ) on 'nature-inspired algorithms for large scale global optimization,'" *Inf. Sci.*, vol. 316, pp. 437–439, Sep. 2015.

[3] G. N. Vanderplaats, "Very large scale optimization," Nat. Aeronautics Space Admin., Washington, DC, USA, and Langley Res. Center, Hampton, VA, USA, Tech. Rep. NASA/CR-2002-211768, 2002.

[4] G. E. Hinton and R. R. Salakhutdinov, "Reducing the dimensionality of data with neural networks," *Science*, vol. 313, no. 5786, pp. 504–507, Jul. 2006.

[5] J. Ngiam *et al.*, "On optimization methods for deep learning," in *Proc. 28th Int. Conf. Mach. Learn.*, Bellevue, WA, USA, 2011, pp. 265–272.

[6] Z.-H. Zhou, N. V. Chawla, Y. Jin, and G. J. Williams, "Big data opportunities and challenges: Discussions from data analytics perspectives," *IEEE Comput. Intell. Mag.*, vol. 9, no. 4, pp. 62–74, Nov. 2014.

[7] Z. Yang, B. Sendhoff, K. Tang, and X. Yao, "Target shape design optimization by evolving B-splines with cooperative coevolution," *Appl. Soft Comput.*, vol. 48, pp. 672–682, Nov. 2016.

[8] H.-F. Teng, Y. Chen, W. Zeng, Y.-J. Shi, and Q.-H. Hu, "A dual-system variable-grain cooperative coevolutionary algorithm: Satellite-module layout design," *IEEE Trans. Evol. Comput.*, vol. 14, no. 3, pp. 438–455, Jun. 2010.

[9] S. Kimura *et al.*, "Inference of S-system models of genetic networks using a cooperative coevolutionary algorithm," *Bioinformatics*, vol. 21, no. 7, pp. 1154–1163, Apr. 2005.

[10] C. Wang and J. Gao, "High-dimensional waveform inversion with cooperative coevolutionary differential evolution algorithm," *IEEE Geosci. Remote Sens. Lett.*, vol. 9, no. 2, pp. 297–301, Mar. 2012.

[11] Y. Wang *et al.*, "Two-stage based ensemble optimization framework for large-scale global optimization," *Eur. J. Oper. Res.*, vol. 228, no. 2, pp. 308–320, Jul. 2013.

[12] H. Y. Benson, D. F. Shanno, and R. J. Vanderbei, "A comparative study of large-scale nonlinear optimization algorithms," in *High Performance Algorithms and Software for Nonlinear Optimization*, G. Di Pillo and A. Murli, Eds. New York, NY, USA: Springer, 2003, pp. 95–127, doi: 10.1007/978-1-4613-0241-4_5.

[13] W. W. Hager, D. W. Hearn, and P. M. Pardalos, *Large Scale Optimization: State of the Art*. New York, NY, USA: Springer, 2013.

[14] 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.

[15] 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.

[16] K. Deb, A. R. Reddy, and G. Singh, "Optimal scheduling of casting sequence using genetic algorithms," *Mater. Manuf. Processes*, vol. 18, no. 3, pp. 409–432, 2003.

[17] K. Deb and C. Myburgh, "Breaking the billion-variable barrier in real-world optimization using a customized evolutionary algorithm," in *Proc. Genet. Evol. Comput. Conf.*, Denver, CO, USA, 2016, pp. 653–660.

[18] X. Li and X. Yao, "Cooperatively coevolving particle swarms for large scale optimization," *IEEE Trans. Evol. Comput.*, vol. 16, no. 2, pp. 210–224, Apr. 2012.

[19] S.-Z. Zhao, J. J. Liang, P. N. Suganthan, and M. F. Tasgetiren, "Dynamic multi-swarm particle swarm optimizer with local search for large scale global optimization," in *Proc. IEEE Congr. Evol. Comput.*, Hong Kong, 2008, pp. 3845–3852.

[20] R. Cheng and Y. Jin, "A competitive swarm optimizer for large scale optimization," *IEEE Trans. Cybern.*, vol. 45, no. 2, pp. 191–204, Feb. 2015.

[21] A. LaTorre, S. Muelas, and J.-M. Peña, "Large scale global optimization: Experimental results with MOS-based hybrid algorithms," in *Proc. IEEE Congr. Evol. Comput.*, Cancún, Mexico, 2013, pp. 2742–2749.

[22] D. Molina, M. Lozano, and F. Herrera, "MA-SW-Chains: Memetic algorithm based on local search chains for large scale continuous global optimization," in *Proc. IEEE Congr. Evol. Comput.*, Barcelona, Spain, 2010, pp. 1–8.

[23] L.-Y. Tseng and C. Chen, "Multiple trajectory search for large scale global optimization," in *Proc. IEEE Congr. Evol. Comput.*, Hong Kong, 2008, pp. 3052–3059.

[24] J. Brest, A. Zamuda, B. Bošković, M. S. Maučec, and V. Žumer, "High-dimensional real-parameter optimization using self-adaptive differential evolution algorithm with population size reduction," in *Proc. IEEE Congr. Evol. Comput.*, Hong Kong, 2008, pp. 2032–2039.

[25] J. Zhang and A. C. Sanderson, "JADE: Adaptive differential evolution with optional external archive," *IEEE Trans. Evol. Comput.*, vol. 13, no. 5, pp. 945–958, Oct. 2009.

[26] I. Loshchilov, "LM-CMA: An alternative to L-BFGS for large-scale black box optimization," *Evol. Comput.*, vol. 25, no. 1, pp. 143–171, 2017.

[27] W. Dong, T. Chen, P. Tino, and X. Yao, "Scaling up estimation of distribution algorithms for continuous optimization," *IEEE Trans. Evol. Comput.*, vol. 17, no. 6, pp. 797–822, Dec. 2013.

[28] A. Kabán, J. Bootkrajang, and R. J. Durrant, "Toward large-scale continuous EDA: A random matrix theory perspective," *Evol. Comput.*, vol. 24, no. 2, pp. 255–291, Jun. 2016.

[29] T. Bhowmik, H. Liu, Z. Ye, and S. Oraintara, "Dimensionality reduction based optimization algorithm for sparse 3-D image reconstruction in diffuse optical tomography," *Sci. Reports*, vol. 6, Mar. 2016, Art. no. 22242.

[30] R. G. Regis and C. A. Shoemaker, "Local function approximation in evolutionary algorithms for the optimization of costly functions," *IEEE Trans. Evol. Comput.*, vol. 8, no. 5, pp. 490–505, Oct. 2004.

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

[32] 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.

[33] 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 Nat.*, vol. 6239. Kraków, Poland, 2011, pp. 300–309.

[34] M. N. Omidvar, Y. Mei, and X. Li, "Effective decomposition of large-scale separable continuous functions for cooperative co-evolutionary algorithms," in *Proc. IEEE Congr. Evol. Comput.*, Beijing, China, 2014, pp. 1305–1312.

[35] B. Kazimipour, M. N. Omidvar, X. Li, and A. K. Qin, "A novel hybridization of opposition-based learning and cooperative co-evolutionary for large-scale optimization," in *Proc. IEEE Congr. Evol. Comput.*, Beijing, China, 2014, pp. 2833–2840.

[36] M. N. Omidvar, X. Li, and X. Yao, "Smart use of computational resources based on contribution for cooperative co-evolutionary algorithms," in *Proc. Genet. Evol. Comput. Conf.*, Dublin, Ireland, 2011, pp. 1115–1122.

[37] 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, Jun. 2015, Art. no. 13.

[38] L. Sun, S. Yoshida, X. Cheng, and Y. Liang, "A cooperative particle swarm optimizer with statistical variable interdependence learning," *Inf. Sci.*, vol. 186, no. 1, pp. 20–39, Mar. 2012.

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

[40] S. Mahdavi, M. E. Shiri, and S. Rahnamayan, "Cooperative co-evolution with a new decomposition method for large-scale optimization," in *Proc. IEEE Congr. Evol. Comput.*, Beijing, China, 2014, pp. 1285–1292.

[41] F. Wei, Y. Wang, and T. Zong, "A novel cooperative coevolution for large scale global optimization," in *Proc. IEEE Int. Conf. Syst. Man Cybern.*, San Diego, CA, USA, 2014, pp. 738–741.

[42] K. Zhang and B. Li, "Cooperative coevolution with global search for large scale global optimization," in *Proc. IEEE Congr. Evol. Comput.*, Brisbane, QLD, Australia, 2012, pp. 1–7.

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

[44] Y. Wang, B. Li, and X. Lai, "Variance priority based cooperative co-evolution differential evolution for large scale global optimization," in *Proc. IEEE Congr. Evol. Comput.*, 2009, pp. 1232–1239.

[45] J. Fan, J. Wang, and M. Han, "Cooperative coevolution for large-scale optimization based on kernel fuzzy clustering and variable trust region methods," *IEEE Trans. Fuzzy Syst.*, vol. 22, no. 4, pp. 829–839, Aug. 2014.

[46] A. Zamuda, J. Brest, B. Bošković, and V. Žumer, "Large scale global optimization using differential evolution with self-adaptation and cooperative co-evolution," in *Proc. IEEE Congr. Evol. Comput.*, Hong Kong, 2008, pp. 3718–3725.

[47] L. M. Antonio and C. A. C. Coello, "Use of cooperative coevolution for solving large scale multiobjective optimization problems," in *Proc. IEEE Congr. Evol. Comput.*, Cancún, Mexico, 2013, pp. 2758–2765.

[48] M. A. Potter and K. A. De Jong, "A cooperative coevolutionary approach to function optimization," in *Proc. Int. Conf. Parallel Problem Solving Nat.*, vol. 2. Taormina, Italy, 1994, pp. 249–257.

[49] M. Perc and A. Szolnoki, "Coevolutionary games—A mini review," *BioSystems*, vol. 99, no. 2, pp. 109–125, Feb. 2010.

[50] K. Tang, X. Li, P. N. Suganthan, Z. Yang, and T. Weise, "Benchmark functions for the CEC'2010 special session and competition on large-scale global optimization," Nat. Inspired Comput. Appl. Lab., USTC, Hefei, China, Tech. Rep., 2009.

[51] M. N. Omidvar, X. Li, and K. Tang, "Designing benchmark problems for large-scale continuous optimization," *Inf. Sci.*, vol. 316, pp. 419–436, Sep. 2015.

[52] Y. Shi, H. Teng, and Z. Li, "Cooperative co-evolutionary differential evolution for function optimization," in *Proc. Int. Conf. Nat. Comput.*, Changsha, China, 2005, pp. 1080–1088.

[53] F. van den Bergh and A. P. Engelbrecht, "A cooperative approach to particle swarm optimization," *IEEE Trans. Evol. Comput.*, vol. 8, no. 3, pp. 225–239, Jun. 2004.

[54] 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.*, Barcelona, Spain, 2010, pp. 1–8.

[55] J. Liu and K. Tang, "Scaling up covariance matrix adaptation evolution strategy using cooperative coevolution," in *Proc. Int. Conf. Intell. Data Eng. Autom. Learn.*, Hefei, China, 2013, pp. 350–357.

[56] Z. Yang, K. Tang, and X. Yao, "Multilevel cooperative coevolution for large scale optimization," in *Proc. IEEE Congr. Evol. Comput.*, Hong Kong, Jun. 2008, pp. 1663–1670.

[57] H. H. Rosenbrock, "An automatic method for finding the greatest or least value of a function," *Comput. J.*, vol. 3, no. 3, pp. 175–184, Mar. 1960.

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

[59] J. E. Hopcroft and R. E. Tarjan, "Algorithm 447: Efficient algorithms for graph manipulation," *Commun. ACM*, vol. 16, no. 6, pp. 372–378, Jun. 1973.

[60] *IEEE Standard for Floating-Point Arithmetic*, IEEE Standard 754-2008," Aug. 2008.

[61] R. M. Corless and N. Fillion, *A Graduate Introduction to Numerical Methods*. New York, NY, USA: Springer-Verlag, 2013.

[62] N. J. Higham, *Accuracy and Stability of Numerical Algorithms*. Philadelphia, PA, USA: SIAM, 2002.

[63] P. H. Sterbenz, *Floating-Point Computation*. Englewood Cliffs, NJ, USA: Prentice-Hall, 1973.

[64] N. Hansen and S. Kern, "Evaluating the CMA evolution strategy on multimodal test functions," in *Proc. Int. Conf. Parallel Problem Solving Nat.*, Birmingham, U.K., 2004, pp. 282–291.

[65] 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," School Comput. Sci., and Inf. Technol., RMIT Univ., Melbourne, VIC, Australia, Tech. Rep., 2013.

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

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

[68] N. Hansen, S. D. Müller, and P. Koumoutsakos, "Reducing the time complexity of the derandomized evolution strategy with covariance matrix adaptation (CMA-ES)," *Evol. Comput.*, vol. 11, no. 1, pp. 1–18, 2003.

[69] B. Kazimipour, M. N. Omidvar, X. Li, and A. K. Qin, "A sensitivity analysis of contribution-based cooperative co-evolutionary algorithms," in *Proc. IEEE Congr. Evol. Comput.*, Sendai, Japan, 2015, pp. 417–424.

[70] Y. Sun, M. Kirley, and S. K. Halgamuge, "On the selection of decomposition methods for large scale fully non-separable problems," in *Proc. Companion Genet. Evol. Comput. Conf.*, Madrid, Spain, 2015, pp. 1213–1216.

[71] Y. Liu, X. Yao, Q. Zhao, and T. Higuchi, "Scaling up fast evolutionary programming with cooperative coevolution," in *Proc. IEEE Congr. Evol. Comput.*, Seoul, South Korea, 2001, pp. 1101–1108.

Mohammad Nabi Omidvar (M'09) received the first bachelor's (First Class Hons.) degree in computer science, the second bachelor's degree in applied mathematics, and the Ph.D. degree in computer science from RMIT University, Melbourne, VIC, Australia, in 2010, 2014, and 2016, respectively.

He is a Research Fellow in Evolutionary Computation with the School of Computer Science, University of Birmingham, Birmingham, U.K. His current research interests include large-scale global optimization, decomposition methods for optimization, and multiobjective optimization.

Dr. Omidvar was a recipient of the IEEE TRANSACTION ON EVOLUTIONARY COMPUTATION Outstanding Paper Award for his research on large-scale global optimization in 2017, the Australian Postgraduate Award in 2010, and the Best Computer Science Honours Thesis Award from the School of Computer Science and IT, RMIT University. He is a member of IEEE Taskforce on Large-Scale Global Optimization.

Ming Yang received the B.Sc., M.Sc., and Ph.D. degrees in computer science from the China University of Geosciences, Wuhan, China, in 2005, 2008, and 2012, respectively.

He carried out a Post-Doctoral Research with the School of Computer Science, University of Birmingham, Birmingham, U.K., from 2014 to 2015. He is currently an Associate Professor with the School of Computer Science, China University of Geosciences. He also with the Hubei Key Laboratory of Intelligent Geo-Information Processing, Wuhan. His current research interests include swarm intelligence, large-scale optimization, and multiobjective optimization and their applications.

Yi Mei (M'09) received the B.Sc. and Ph.D. degrees from the University of Science and Technology of China, Hefei, China, in 2005 and 2010, respectively.

He is a Lecturer with the School of Engineering and Computer Science, Victoria University of Wellington, Wellington, New Zealand. He has over 50 fully referred publications, including the top journals in EC and Operations Research (OR), such as the IEEE TRANSACTIONS ON EVOLUTIONARY COMPUTATION, the IEEE TRANSACTIONS ON CYBERNETICS, the *European Journal of Operational Research*, and the ACM TRANSACTIONS ON MATHEMATICAL SOFTWARE. His current research interests include evolutionary computation in scheduling, routing, and combinatorial optimization, as well as evolutionary machine learning, genetic programming, feature selection, and dimensional reduction.

Dr. Mei currently serves as the Vice-Chair of the IEEE CIS Emergent Technologies Technical Committee, and a member of three IEEE CIS Task Forces. He is a Guest Editor of a special issue of the *Genetic Programming Evolvable Machine* journal. He serves as a Reviewer of over 25 international journals, including the top journals in EC and OR.

Xiaodong Li (M'03–SM'07) received the B.Sc. degree from Xidian University, Xi'an, China, and the Ph.D. degree in information science from the University of Otago, Dunedin, New Zealand.

He is a Professor with the School of Science (Computer Science and Software Engineering), RMIT University, Melbourne, VIC, Australia. His current research interests include evolutionary computation, neural networks, data analytics, multiobjective optimization, multimodal optimization, and swarm intelligence.

Dr. Li was a recipient of the 2013 ACM SIGEVO Impact Award and the 2017 IEEE CIS IEEE TRANSACTIONS ON EVOLUTIONARY COMPUTATION Outstanding Paper Award. He serves as an Associate Editor for the IEEE TRANSACTIONS ON EVOLUTIONARY COMPUTATION, *Swarm Intelligence* (Springer), and the *International Journal of Swarm Intelligence Research*. He is a Founding Member of IEEE CIS Task Force on Swarm Intelligence, the Vice-Chair of IEEE Task Force on Multimodal Optimization, and the Former Chair of IEEE CIS Task Force on Large Scale Global Optimization.

Xin Yao (F'03) received the B.Sc. degree from the University of Science and Technology of China (USTC), Hefei, China, in 1982, the M.Sc. degree from the North China Institute of Computing Technology (NCI), Haidian, China, in 1985, and the Ph.D. degree from USTC in 1990.

He is a Chair Professor of Computer Science with the Southern University of Science and Technology, Shenzhen, China, and a part-time Professor of Computer Science with the University of Birmingham, Birmingham, U.K. His current research interests include evolutionary computation, and ensemble learning and their applications in software engineering. With his Ph.D. student and colleague, he pioneered the recent effort in large scale evolutionary optimization in a 2008 paper "Large scale evolutionary optimization using cooperative co-evolution."

Dr. Yao was a recipient of the 2017 IEEE TRANSACTIONS ON EVOLUTIONARY COMPUTATION Outstanding Paper Award with his research fellows and colleagues on this topic "Cooperative Co-Evolution With Differential Grouping for Large Scale Optimization," the Prestigious Royal Society Wolfson Research Merit Award in 2012, the IEEE CIS Evolutionary Computation Pioneer Award in 2013, the 2001 IEEE Donald G. Fink Prize Paper Award, the 2010 and 2016 IEEE TRANSACTIONS ON EVOLUTIONARY COMPUTATION Outstanding Paper Awards, the 2011 IEEE TRANSACTIONS ON NEURAL NETWORKS Outstanding Paper Award, and many other best paper awards. He is a Distinguished Lecturer of IEEE Computational Intelligence Society (CIS). He was the President of IEEE CIS from 2014 to 2015, and the Editor-in-Chief of the IEEE TRANSACTIONS ON EVOLUTIONARY COMPUTATION from 2003 to 2008.