Cooperative Co-Evolution With Differential Grouping for Large Scale Optimization

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

***Abstract*—Cooperative co-evolution has been introduced into evolutionary algorithms with the aim of solving increasingly complex optimization problems through a divide-and-conquer paradigm. In theory, the idea of co-adapted subcomponents is desirable for solving large-scale optimization problems. However, in practice, without prior knowledge about the problem, it is not clear how the problem should be decomposed. In this paper, we propose an automatic decomposition strategy called differential grouping that can uncover the underlying interaction structure of the decision variables and form subcomponents such that the interdependence between them is kept to a minimum. We show mathematically how such a decomposition strategy can be derived from a definition of partial separability. The empirical studies show that such near-optimal decomposition can greatly improve the solution quality on large-scale global optimization problems. Finally, we show how such an automated decomposition allows for a better approximation of the contribution of various subcomponents, leading to a more efficient assignment of the computational budget to various subcomponents.**

***Index Terms*—Cooperative co-evolution, large-scale optimization, nonseparability, numerical optimization, problem decomposition.**

I. Introduction

OPTIMIZATION problems in science and engineering are often very complex and solutions cannot be readily found with a direct approach. As a result, it is imperative to investigate ways of simplifying a given complex problem. The number of decision variables is a major contributing factor to the complexity of an optimization problem [1]. There are a number of approaches for solving large-scale problems with a large number of decision variables. One such approach is to decompose the original large-scale problem into a set of smaller and simpler subproblems, which are more manageable and easier to solve. Once such a decomposition is realized, the whole problem can be solved by separately optimizing the individual subproblems. This so-called divide-and-conquer strategy can be traced back to René Descartes' famous book *A Discourse on Method* [2]. The effectiveness of decomposition has been established in many classical optimization methods [3]–[5].

The focus of this paper is to examine large-scale global optimization of real-valued functions, using automatic decomposition. Evolutionary algorithms (EAs) [6] are effective optimization methods and have been extensively used for solving a wide range of optimization problems [7]. However, their performance deteriorates rapidly as the dimensionality of the problem increases [8]. This is referred to as the curse of dimensionality [9]. Cooperative co-evolution (CC) has been proposed by Potter and De Jong [10] as an explicit means of problem decomposition in evolutionary algorithms. A major difficulty in applying CC is the choice of a good decomposition strategy. Moreover, the performance of optimization is potentially sensitive to the chosen decomposition. It was shown by Salomon [11] that interdependence between variables can greatly affect the performance of optimization algorithms in continuous domains. In classical genetic algorithm (GA) research, these variable interdependencies are referred to as linkage or epistasis [1], [12] and have been extensively investigated in the context of binary GAs [13].

The decomposition strategy in CC is very similar to the problem of ordering genes in the early days of genetic algorithm research [12]. Ordering of genes on a chromosome can have a significant impact on the performance of EAs. In an experiment conducted by Goldberg *et al.* [14], it was shown that good ordering of genes is the difference between success and failure of a simple genetic algorithm. The dependence between ordering of genes and the performance of EAs is directly related to the gene interaction problem.

Although decomposition plays a crucial role in the performance of EAs, there is often insufficient knowledge about the structure of a given problem to be able to manually devise a suitable decomposition strategy. It is therefore desirable to design new procedures capable of exploiting the hidden structure of a problem to automatically find a suitable decomposition.

In addition to the impact that a near-optimal decomposition can have on the performance of CC, it has been shown recently that it is possible to quantify the contribution of

Manuscript received December 26, 2012; revised May 18, 2013; accepted May 20, 2013. Date of publication September 11, 2013; date of current version May 27, 2014. This work was supported in part by the EPSRC under Grant EP/J017515/1 and the ARC Discovery under Grant DP120102205.

M. N. Omidvar, X. Li, and Y. Mei are with the Evolutionary Computing and Machine Learning Group, School of Computer Science and IT, RMIT University, Melbourne VIC 3001, Australia (e-mail: mohammad.omidvar@rmit.edu.au; xiaodong.li@rmit.edu.au; yi.mei@rmit.edu.au).

X. Yao is with the Centre 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).

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.2013.2281543

a subcomponent to the global fitness [15]. Once this contribution information is calculated, the computational budget can be divided between the subcomponents according to their contributions, unlike traditional CC where the computational budget is equally divided between all subcomponents. It has been shown that a contribution-based scheme outperforms traditional CC [15].

In this paper, we propose a decomposition method called differential grouping that allows an automatic near-optimal decomposition of decision variables. More specifically, we have the following research objectives:

1) to provide a theoretical foundation for identifying interacting variables and to propose an algorithm to group the interacting variables with high accuracy;
2) to design an automatic decomposition mechanism, which is equally applicable to both traditional and evolutionary optimization algorithms;
3) to show how a near-optimal decomposition is beneficial in solving large-scale global optimization problems with up to 1000 decision variables;
4) to show how an automatic near-optimal decomposition strategy combined with contribution-based cooperative co-evolution (CBCC) can further improve the performance of an optimization process, especially on large-scale problems.

The remainder of this paper is organized as follows. In Section II, a review of variable interaction problems and various decomposition methods is given. In Section III, the proposed differential grouping algorithm is derived from a definition of partial separability. Section IV outlines the benchmark problems used to evaluate the performance of differential grouping. In Section V, first the performance of the differential grouping algorithm is compared to another state-of-the-art decomposition method, then the effectiveness of the differential grouping algorithm in improving the optimization performance of evolutionary algorithms is investigated; and finally, the performance of differential grouping is benchmarked within a contribution-based framework. Section VI summarizes and concludes the paper.

II. Related Work

This section defines the notion of nonseparability and provides a survey of cooperative co-evolutionary models and various decomposition and linkage detection methods.

A. Gene Interaction

In natural genetics, two genes are said to interact with each other if they collectively represent a feature at the phenotype level [16]. Another form of interaction happens when the value taken by one gene activates or deactivates an effect of other genes [16]. The term epistasis is used to refer to any type of gene interaction [1], [17], [18]. In the context of GAs, this is also referred to as linkage [1], [13]. Nonseparability refers to the same concept, but it is more widely used in the continuous optimization literature. The formal definition of separability and nonseparability is as follows [1], [19]:

Definition 1: A function $f(x_1, \ldots, x_n)$ is separable iff

$$\arg\min_{(x_1,\ldots,x_n)} f(x_1, \ldots, x_n) = \left(\arg\min_{x_1} f(x_1, \ldots), \ldots, \arg\min_{x_n} f(\ldots, x_n)\right) \quad (1)$$

and nonseparable otherwise (assuming minimization).

In other words, if it is possible to find the global optimum of a function by optimizing one dimension at a time regardless of the values taken by other dimensions, then the function is said to be separable. It is nonseparable otherwise. One way of creating a nonseparable function is by rotating the fitness landscape of the original objective function around its coordinate axes [11].

B. Cooperative Co-Evolution

CC is an effective method for solving large-scale optimization problems. This effectiveness is attributed to the decomposition of a large-scale problem into a set of smaller subproblems. This has been empirically verified in [8]. However, one drawback of CC is that its performance is sensitive to the choice of decomposition strategy. Here, we review various decomposition strategies suggested for CC with more emphasis on techniques proposed in the context of large-scale global optimization.

In the original implementation of the cooperative co-evolutionary genetic algorithm (CCGA), Potter and De Jong [10] decomposed an n-dimensional problem into n 1-D problems. Once the subcomponents are identified, they undergo optimization using an evolutionary optimizer in a round-robin fashion. It was shown that a variant of CCGA, CCGA-1, did not perform well on the Griewank function [20], a nonseparable function. Further experiments on the Rosenbrock function [20], another nonseparable function, confirmed that the poor performance of CCGA-1 on nonseparable problems is due to interdependencies between the decision variables. In this original CCGA study [10], the problems only had a maximum of 30 dimensions. Liu *et al.* [8] made the first attempt to solve large-scale optimization problems using a CC framework. They applied fast evolutionary programming with CC [8] on benchmark problems with up to 1000 dimensions. The experimental results showed that a cooperative co-evolutionary approach scales better as the dimensionality of the problem increases. However, since they mostly used separable functions for their experiments, it is unclear how their algorithm will scale up on nonseparable functions.

Van den Bergh and Engelbrecht [21] were the first to apply particle swarm optimization (PSO) [22] to a cooperative co-evolutionary framework (CPSO). Unlike CCGA, they decomposed an n-dimensional problem into k s-dimensional problems for some $s << n$. However, CPSO was not tested against large-scale problems. CC was also used with differential evolution [23] by Shi *et al.* [24], where the decision variables were divided into two equally sized subcomponents. It is clear that this decomposition strategy does not scale well as the dimensionality increases.

Random grouping is a more recent decomposition strategy proposed by Yang *et al.* [25]. Similar to CPSO, random

grouping decomposes a problem into k s-dimensional subproblems, but instead of using a static grouping, it randomly allocates the decision variables to subcomponents in every coevolutionary cycle. It was shown mathematically that with random grouping the probability of placing two interacting variables in the same subcomponent for several cycles is reasonably high. Random grouping achieved a good performance on a set of benchmark functions with up to 1000 variables [25]. Li and Yao [26] developed CCPSO2 (an improved version of CPSO) based on a revised random grouping scheme, and tackled problems with up to 2000 dimensions. Despite the success of random grouping, it has been shown that it is ineffective when the number of interacting variables grows beyond approximately five variables [27]. An alternative approach called delta grouping [28] was shown to outperform random grouping on most functions from a set of 20 large-scale benchmark problems [20]. However, a drawback of delta grouping is its low performance when there is more than one nonseparable subcomponent in the objective function.

All of the grouping strategies described so far use a predefined and fixed subcomponent size. For example, random grouping and delta grouping decompose an n-dimensional problem into k s-dimensional problems. A major drawback of these techniques is that the user needs to specify a value for either k or s. If there are large groups of interacting variables in the objective function then a small value of s may degrade the performance of the algorithm. If the problem contains many small groups of interacting variables then a large value of s does not utilize the power of a decomposition approach to its full potential. To alleviate this problem, Yang *et al.* proposed a multilevel cooperative co-evolution (MLCC) algorithm [29]. In MLCC, instead of using a fixed number for s, a set of possible s values is provided to the algorithm. During the course of evolution, the performance of each of using these subcomponent sizes is measured and the values with better performance are given a higher probability of being selected in the next co-evolutionary cycle. This technique partially solves the problem of specifying a single s value. However, the user still needs to decide about a set of potential s values. Another drawback of this multilevel scheme is that once an s value is chosen, the decision variables are divided into a set of equally sized subcomponents. It is unlikely that in most real-world problems the sizes of interacting groups will be equal. Hence, it is desirable that a decomposition strategy can automatically determine the number of subcomponents and their sizes.

C. Classification of Decomposition Strategies

Decomposition methods have been studied extensively in the field of binary GAs [13]. Such algorithms are commonly referred to as linkage learning algorithms. The main motivation in classical linkage learning research is to design crossover operators, which take into account the linkage structure, and allow a set of linked genes to be inherited together in the mating process. More recently, especially in the context of continuous global optimization, the grouping which is discovered using an automatic decomposition strategy is superimposed on a CC framework to form the co-evolving subcomponents [28], [30].

Linkage learning algorithms were classified by Yu *et al.* [31] into three major categories: perturbation, interaction adaptation, and model building. Here, we include a fourth category, random methods, for a more complete treatment of various decomposition strategies in both conventional and co-evolutionary algorithms.

1) *Random Methods:* These algorithms do not rely on a systematic or smart procedure to discover the interdependencies. Instead, they randomly permute the decision variables to increase the probability of placing interacting variables close to each other for a few evolutionary cycles. The inversion operator [12], [32], one of the early attempts to overcome the gene interaction problem, inverts (reverses) the order of genes on a randomly chosen portion of the chromosome. Since the cutting points are selected at random, an arbitrary ordering of the genes can be achieved by repeatedly applying the inversion operator. This is why it should be classified as a random method. In the context of CC, random grouping [25] randomly permutes the order of the decision variables in every co-evolutionary cycle to increase the probability of placing two interacting variables in the same subcomponent for at least a cycle. This technique has two major drawbacks. First, the user has to decide about the number and the size of each subcomponent. Second, if there are more than two interacting variables, the probability of placing all of them in one subcomponent, even for one co-evolutionary cycle, approaches zero as the number of interacting variables increases [27].

2) *Perturbation:* These methods perturb the decision variables using various heuristics. By monitoring the changes to the objective function, detection of the interactions between decision variables is attempted. In most cases, the decomposition stage is performed offline. When the full interaction structure is realized, the representation is modified accordingly and the optimization process starts. Algorithms that rely on perturbation include mGA [14], fmGA [33], gemGA [34], LINC [35], and LIMD [36]. These methods are typically incorporated into a binary GA. A limited number of techniques have also been developed for real-valued GAs, such as LINC-R [37]. However, the experimental results for LINC-R were limited to low dimensional functions with up to 40 dimensions. More techniques have been developed for continuous domains in the context of CC, such as adaptive co-evolutionary optimization [38] and CC with variable interaction learning (CCVIL) [30]. All perturbation techniques mentioned here rely on various heuristics to identify interacting variables. However, there is a very limited theoretical basis for these heuristics. The differential grouping proposed in this paper can be considered as a perturbation technique.

3) *Interaction Adaptation:* These methods incorporate the interaction detection mechanism into the chromosome and simultaneously evolve the order of genes and the decision variables of the original optimization problem. These methods assign a higher reproduction probability to individuals with a tighter grouping of interacting variables. Unlike perturbation methods, adaptive models evolve the decomposition structure through the evolutionary process. Examples of these methods include LEGO [39] and LLGA [40].

4) *Model Building:* These methods build a probabilistic model based on promising solutions in the population. This model is updated iteratively in the evolutionary process, and the next generation is built from the model. Estimation of distribution algorithms [41], [42] fall into this category. Popular model building algorithms include cGA [43], BOA [44], and hBOA [45]. Some of these algorithms, such as BOA, are also used for real-valued optimization [46].

In addition to the work in the field of evolutionary optimization, a number of techniques in classical optimization [3], [4] have been devised to deal with large-scale problems by using a decomposition strategy. Griewank and Toint [3] proposed the partitioned quasi-Newton method to deal with large-scale optimization of partially separable problems. In their work, instead of approximating the global Hessian matrix, they approximate smaller partitions of this matrix by applying the quasi-Newton formula on the component functions. In other words, the Hessian matrix is partitioned into a set of block matrices, where each block is based on independent subfunctions, the sum of which forms the value of the global objective function.

D. Automatic Decomposition in Cooperative Co-Evolution

A number of recent studies have focused on developing automatic decomposition strategies for cooperative co-evolutionary algorithms. The main driving force behind such studies is that CC is a suitable framework for large-scale optimization due to its modular nature. However, a major difficulty in applying CC lies in the decomposition of the decision variables into a set of subcomponents. Without any knowledge of the underlying structure, a given problem can be decomposed in many different ways without any indication of the superiority of one decomposition over another. Ideally, the subcomponents should be formed according to the interaction pattern of the decision variables so that the interactions between the subcomponents are kept to a minimum. Weicker and Weicker [38] proposed a CC technique to identify interacting variables. Although this technique has not been applied to high dimensional problems, to the best of our knowledge it is the first attempt at automatic formation of subcomponents in a CC framework. Recently, Chen *et al.* [30] improved this technique and applied it to large-scale global optimization. Delta grouping [28] is another technique for automatic identification of the interacting variables. However, delta grouping is more effective when there is only one group of interacting variables.

III. Differential Grouping

This section describes the details of differential grouping, the decomposition strategy proposed in this paper. Differential grouping is derived from the definition of partially additively separable functions. These types of functions conveniently represent the modular nature of many real-world problems [47].

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

$$f(\vec{x}) = \sum_{i=1}^{m} f_i(\vec{x}_i) \tag{2}$$

where $\vec{x}_i$ are mutually exclusive decision vectors of f_i, $\vec{x} = \langle x_1, \ldots, x_n \rangle$ is a global decision vector of n dimensions, and m is the number of independent subcomponents.

For a function of the above form if all subcomponent functions are 1-D, then it is called completely additively separable or fully separable for short. Hereafter, the phrase additively separable is used to refer to partially additively separable.

Theorem 1: Let $f(\vec{x})$ be an additively separable function. $\forall a, b_1 \neq b_2, \delta \in \mathbb{R}, \delta \neq 0$, if the following condition holds:

$$\Delta_{\delta,x_p}[f](\vec{x})|_{x_p=a,x_q=b_1} \neq \Delta_{\delta,x_p}[f](\vec{x})|_{x_p=a,x_q=b_2} \tag{3}$$

then x_p and x_q are nonseparable where

$$\Delta_{\delta,x_p}[f](\vec{x}) = f(\ldots, x_p + \delta, \ldots) - f(\ldots, x_p, \ldots) \tag{4}$$

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

Theorem 1 simply states that given an additively separable function $f(\vec{x})$, two variables x_p and x_q interact if (4) evaluated with any two different values for x_q yields different results (i.e., inequality of delta values $\Rightarrow$ nonseparability). In order to prove the theorem it is sufficient to prove its contrapositive, which states that if two variables x_p and x_q are separable, then (4) evaluated with any two different values for x_q yields the same answer (i.e., separability $\Rightarrow$ equality of delta values).

Lemma 1: If $f(\vec{x})$ is additively separable, then for any $x_p \in \vec{x}$ we have

$$\frac{\partial f(\vec{x})}{\partial x_p} = \frac{\partial f_i(\vec{x}_i)}{\partial x_p}, \forall x_p \in \vec{x}_i. \tag{5}$$

Proof: Since $f(\vec{x})$ is additively separable, we have

$$\frac{\partial f(\vec{x})}{\partial x_p} = \frac{\partial \sum_{i=1}^{m} f_i(\vec{x}_i)}{\partial x_p} = \frac{\partial f_1(\vec{x}_1)}{\partial x_p} + \cdots + \frac{\partial f_m(\vec{x}_m)}{\partial x_p} \tag{6}$$

$$\forall x_p \in \vec{x}_i$$

where $\vec{x}_1, \ldots, \vec{x}_m$ are mutually exclusive decision vectors. Therefore, $\frac{\partial f(\vec{x}_j)}{\partial x_p} = 0, \forall j \neq i$. Hence

$$\frac{\partial f(\vec{x})}{\partial x_p} = \frac{\partial f_i(\vec{x}_i)}{\partial x_p}, \forall x_p \in \vec{x}_i. \tag{7}$$

■

Proof of Theorem 1: According to Lemma 1

$$\frac{\partial f(\vec{x})}{\partial x_p} = \frac{\partial f_i(\vec{x}_i)}{\partial x_p}, \forall x_p \in \vec{x}_i.$$

Then, $\forall x_q \notin \vec{x}_i$ we have

$$\left.\frac{\partial f(\vec{x})}{\partial x_p}\right|_{x_q=b_1} = \left.\frac{\partial f(\vec{x})}{\partial x_p}\right|_{x_q=b_2} = \frac{\partial f_i(\vec{x}_i)}{\partial x_p}, \ \forall b_1 \neq b_2.$$

$$\int_a^{a+\delta} \left.\frac{\partial f(\vec{x})}{\partial x_p} dx_p\right|_{x_q=b_1} = \int_a^{a+\delta} \left.\frac{\partial f(\vec{x})}{\partial x_p} dx_p\right|_{x_q=b_2}$$

$$\Delta_{\delta,x_p}[f](\vec{x})|_{x_p=a,x_q=b_1} = \Delta_{\delta,x_p}[f](\vec{x})|_{x_p=a,x_q=b_2}$$

$$\forall a, b_1 \neq b_2, \delta \in \mathbb{R}, \delta \neq 0.$$

Example 1: Consider the nonseparable objective function $f(x_1, x_2) = x_1^2 + \lambda x_1 x_2 + x_2^2$, $\lambda \neq 0$. According to (6) we have

$$\frac{\partial f(x_1, x_2)}{\partial x_1} = 2x_1 + \lambda x_2.$$

This clearly shows that the change in the global objective function with respect to x_1 is a function of x_1 and x_2. Now by applying (4) we have

$$\Delta_{\delta,x_1}[f] = \left[(x_1+\delta)^2 + \lambda(x_1+\delta)x_2 + x_2^2\right] - \left[x_1^2 + \lambda x_1 x_2 + x_2^2\right] = \delta^2 + 2\delta x_1 + \lambda x_2 \delta.$$

It can be seen that the difference equation $\Delta_{\delta,x_1}[f]$ is a function of both x_1 and x_2. Therefore, evaluating $\Delta_{\delta,x_1}[f]$ for two different values of x_2 does not give the same answer. So, according to Theorem 1, we conclude that x_1 and x_2 interact (they are nonseparable). Note that λ reflects the strength of nonseparability. Setting λ to zero makes the function fully separable.

A. Differential Grouping Algorithm

Algorithm 1 shows how Theorem 1 can be used to identify and group the interacting variables into common subcomponents. The algorithm starts by examining the interaction of the first decision variable with all other decision variables in a pairwise fashion by applying Theorem 1. If the algorithm detects an interaction between the first variable and any other variable, it excludes that variable from the set of all decision variables and places it in a subcomponent. This process is repeated until all the variables that interact with the first variable are detected and the first subcomponent is formed. If no interaction is detected, then the variable under examination is considered to be a separable variable. This process is repeated for the remaining variables until there are no more decision variables left. Lines 10, 13, and 14 of Algorithm 1 show how Theorem 1 is used to identify the interacting variables. Note that all the variables are initialized to the lower bound of the function in vector $\vec{p}_1$ (line 7). In order to check for interaction between the ith and the jth dimensions, the vector $\vec{p}_2$ is set to be equal to $\vec{p}_1$ except for the ith dimension. The ith element of vector $\vec{p}_2$ is set to the upper bound of the domain. This allows us to calculate the value of Δ_1. Then, the jth element of $\vec{p}_2$ is set to the center of the search space for that dimension and Δ_2 is calculated. If the quantity $|\Delta_1 - \Delta_2|$ is greater than a small number ϵ, then it is concluded that the ith and the jth dimensions interact with each other (lines 7-16). The jth dimension is then removed from the set of decision variables and is grouped with the ith dimension in a common subcomponent. The same process is repeated until all variables interacting with the ith dimension are extracted. The algorithm then identifies all variables interacting with the $(i+1)$th dimension until there are no more dimensions to be examined. It should be noted that the choices of upper bound, lower bound, and the center of the search space to construct $\vec{p}_1$ and $\vec{p}_2$ are arbitrary. These points can be generated randomly as long as they do not coincide with each other to give a difference value of zero.

Algorithm 1: *allgroups* ←grouping(*func*, *lbounds*, *ubounds*, *n*)

```
1:  dims ← {1, 2, ..., n}
2:  seps ← {}
3:  allgroups ←
    {}      // contains a set of all identified groups.
4:  for i ∈ dims do
5:    group ← {i}
6:    for j ∈ dims ∧ i ≠ j do
7:      p1 ← lbound × ones(1, n)
8:      p2 ← p1
9:      p2(i) ← ubound
10:     Δ1 ← func(p1) − func(p2)
11:     p1(j) ← 0
12:     p2(j) ← 0
13:     Δ2 ← func(p1) − func(p2)
14:     if |Δ1 − Δ2| > ε then
15:       group ← group ∪ j
16:     end if
17:   end for
18:   dims ← dims − group
19:   if length(group) = 1 then
20:     seps ← seps ∪ group
21:   else
22:     allgroups ← allgroups ∪ {group}
23:   end if
24: end for
25: allgroups ← allgroups ∪ {seps}
```

The choice of ϵ in Algorithm 1 affects the sensitivity of the algorithm in detecting the interactions between the variables. A smaller ϵ makes the algorithm more sensitive to very weak interactions between the decision variables.

In Section II, it was mentioned that perturbation methods, such as LINC-R [37], lack a theoretical basis. Using the interpretation given in this section, we can show that the heuristic used in LINC-R [37] can be derived by applying Theorem 1.

In LINC-R, an interaction between two variables x_i and x_j is identified by comparing the difference values calculated from the following equations:

$$\Delta_{x_i,x_j}[f] = f(x_i+\delta_i, x_j+\delta_j) - f(x_i, x_j) \quad (8)$$

$$\Delta_{x_i}[f] = f(x_i+\delta_i, x_j) - f(x_i, x_j) \quad (9)$$

$$\Delta_{x_j}[f] = f(x_i, x_j+\delta_j) - f(x_i, x_j)\,. \quad (10)$$

Given these difference values, two variables interact if the following condition holds:

$$\left|\Delta_{x_i,x_j}[f] - \left(\Delta_{x_i}[f] + \Delta_{x_j}[f]\right)\right| > \epsilon$$

or similarly

$$\Delta_{x_i,x_j}[f] \neq \Delta_{x_i}[f] + \Delta_{x_j}[f]\,. \quad (11)$$

By substituting $\Delta_{x_i,x_j}[f]$, $\Delta_{x_i}[f]$, and $\Delta_{x_j}[f]$ from (8)–(10) into (11) and reordering the terms we get

$$f(x_i+\delta_i, x_j+\delta_j) \neq f(x_i+\delta_i, x_j) + f(x_i, x_j+\delta_j) - f(x_i, x_j)\,. \quad (12)$$

Now, Theorem 1 can be used to show the equivalence of the method used in LINC-R and differential grouping. According to Theorem 1, the ith and jth dimensions interact if (4) evaluated at two different x_j yields different results, i.e.

$$f(x_i + \delta_i, x_j) - f(x_i, x_j) \neq f(x_i + \delta_i, x_j + \delta_j) - f(x_i, x_j + \delta_j) \ . \tag{13}$$

By rearranging the terms, it can be seen that this equation is identical to (12), showing how LINC-R and differential grouping are related. However, as mentioned in Section II, LINC-R was tested on a very limited set of low dimensional benchmark functions. The real benefit of such automatic decomposition methods is realized only when they are applied to large-scale optimization problems. Moreover, the LINC-R algorithm does not use a CC framework. Instead, it uses an island model with periodic migration of individuals between islands [37]. This island model is constructed from the discovered interaction groups. A disadvantage of this approach is that the periodic migration of individuals requires re-evaluation of individuals in all islands after each migration, which is not an effective use of computational resources. In Section III-C, we show how a CC framework can be used more efficiently in conjunction with differential grouping to solve large-scale optimization problems.

B. Time Complexity

This section describes how to calculate an upper bound for the total number of fitness evaluations (FEs) required by differential grouping under the assumption that there are $\frac{n}{m}$ nonseparable subcomponents, each with m variables. As shown in Algorithm 1 after each successful application of differential grouping, m variables are removed from the set of remaining decision variables. Based on the sum of an arithmetic progression, an upper bound (S) can be calculated for the number of times that the inner loop of Algorithm 1 is executed

$$\begin{aligned} S &= (n-1) + (n-m-1) + \cdots + \left(n - \left(\frac{n}{m} - 1\right) m - 1\right) \\ &= (n-1) + (n-m-1) + \cdots + (m-1) \\ &= \frac{n}{2m}(n+m-2) \ . \end{aligned} \tag{14}$$

Since there are four fitness evaluations in the inner loop (Algorithm 1, lines 10 and 13), a perfect grouping will require a total of $4S$ fitness evaluations. However, Algorithm 1 can be optimized further by realizing that Δ_1 is not changed during the execution of the inner loop and can be moved outside. The total number of required fitness evaluations therefore reduces to $2(S + \frac{n}{m})$. As an example, for $n = 1000$ and $m = 50$, the following number of fitness evaluations is required:

$$FE = 2(S + \frac{n}{m}) = 2\left[\frac{1000}{100}(1000 + 50 - 2) + 20\right] = 21000 \ .$$

Similarly for a fully separable function with $n = 1000$ and $m = 1$, the number of fitness evaluations is

$$FE = 2(S + \frac{n}{m}) = 2\left[\frac{1000}{2}(1000 - 1) + 1000\right] = 1001000 \ .$$

Algorithm 2: CC($func, lbounds, ubounds, n$)

```
1:  groups ←
    grouping(func, lbounds, ubounds, n)//grouping stage.
2:  pop ← rand(popsize, n)        //optimization stage.
3:  (best, best_val) ← min(func(pop))
4:  for i ← 1 to cycles do
5:    for j ← 1 to size(groups) do
6:      indicies ← groups[j]
7:      subpop ← pop[:, indicies]
8:      subpop ← optimizer(best, subpop, FE)
9:      pop[:, indicies] ← subpop
10:     (best, best_val) ← min(func(pop))
11:   end for
12: end for
```

The time complexity of differential grouping with respect to the maximum number of fitness evaluations is as follows:

$$O(FE) = O\left(2\left(S + \frac{n}{m}\right)\right) = O\left(\frac{n^2}{m}\right). \tag{15}$$

C. Differential Grouping Algorithm With CC

This section explains how the differential grouping algorithm is used in a CC framework for solving large-scale global optimization problems.

Algorithm 2 shows the CC framework used for this research. Note that the algorithm has two major stages: a grouping stage (line 1) and an optimization stage (lines 4–12). During the grouping stage the underlying interaction structure of the decision variables is discovered by the *grouping* function, and the subcomponents are formed accordingly. Note that the *grouping* function can refer to any offline grouping procedure, but in this paper it refers to the differential grouping procedure introduced in Algorithm 1. In the optimization stage the subcomponents that are formed in the grouping stage are optimized in a round-robin fashion for a predetermined number of cycles. The *optimizer* function can be any numerical optimization algorithm that can exploit the provided grouping information.

It has been shown recently that putting equal emphasis on all the subcomponents in a CC framework is not a very efficient use of the computational budget [15]. Unlike traditional CC, in contribution-based cooperative co-evolution (CBCC) [15], subcomponents are chosen based on their contributions to the improvement of the global fitness. As a result, a subcomponent with a higher contribution to the global fitness will be given more computational resources. However, one of the requirements for effective estimation of contributions is that the interdependencies between the subcomponents are kept to a minimum. In other words, all the interacting variables should be placed within the same subcomponents.

IV. Experimental Settings

In order to evaluate the performance of differential grouping a set of 20 benchmark functions were used. These benchmark functions were proposed for the IEEE CEC'2010 special

TABLE I
PERFORMANCE OF DIFFERENTIAL GROUPING AND CCVIL ON CEC'2010 BENCHMARK FUNCTIONS (SEPARATED BY ''/'')

Function	Sep Vars	Non-sep Vars	Non-sep Groups	Differential Grouping ($\epsilon = 10^{-3}$) / CCVIL					
				# Captured Sep Vars	# Captured Non-sep Vars	# Formed Non-sep Groups	# Misplaced Vars	# FE	Grouping Accuracy
f_1	1000	0	0	1000 / 1000	0 / 0	0 / 0	0 / 0	1001000 / 69990	100% / 100%
f_2	1000	0	0	1000 / 1000	0 / 0	0 / 0	0 / 0	1001000 / 69990	100% / 100%
f_3	1000	0	0	1000 / 938	0 / 0	0 / 0	0 / 31	1001000 / 1798666	100% / 93.8%
f_4	950	50	1	33 / 957	50 / 43	10 / 1	0 / 7	14564 / 1797614	100% / 86%
f_5	950	50	1	950 / 950	50 / 50	1 / 1	0 / 0	905450 / 1795705	100% / 100%
f_6	950	50	1	950 / 910	50 / 47	1 / 22	0 / 3	906332 / 1796370	100% / 94%
f_7	950	50	1	247 / 951	34 / 49	4 / 1	16 / 1	67250 / 1796475	69% / 98%
f_8	950	50	1	135 / 1000	46 / 0	5 / 0	4 / 50	23608 / 69842	92% / 0%
f_9	500	500	10	500 / 583	500 / 337	10 / 33	0 / 0	270802 / 1792212	100% / 67.4%
f_{10}	500	500	10	500 / 508	500 / 492	10 / 10	0 / 8	272958 / 1774642	100% / 98.4%
f_{11}	500	500	10	501 / 476	499 / 491	10 / 26	1 / 9	270640 / 1774565	99.2% / 98.2%
f_{12}	500	500	10	500 / 516	500 / 435	10 / 11	0 / 65	271390 / 1777344	100% / 87%
f_{13}	500	500	10	131 / 1000	126 / 0	40 / 0	374 / 500	49470 / 69990	25.2% / 0%
f_{14}	0	1000	20	0 / 150	1000 / 719	20 / 63	0 / 281	21000 / 1785975	100% / 71.9%
f_{15}	0	1000	20	0 / 18	1000 / 982	20 / 20	0 / 18	21000 / 1751241	100% / 98.2%
f_{16}	0	1000	20	4 / 11	996 / 989	20 / 20	4 / 11	21128 / 1751647	99.6% / 98.9%
f_{17}	0	1000	20	0 / 25	1000 / 975	20 / 20	0 / 25	21000 / 1752340	100% / 97.5%
f_{18}	0	1000	20	85 / 1000	173 / 0	49 / 0	827 / 1000	34230 / 69990	17.3% / 0%
f_{19}	0	1000	1	0 / 0	1000 / 1000	1 / 1	0 / 0	2000 / 48212	100% / 100%
f_{20}	0	1000	1	42 / 972	82 / 20	16 / 14	918 / 980	22206 / 1798708	8.2% / 2%

session on large-scale global optimization and the associated competition [20]. The CEC'2010 benchmark functions are classified into the following five groups making an ideal test set for evaluating differential grouping:

1) separable functions (f_1–f_3);
2) single-group m-nonseparable functions (f_4–f_8);
3) $\frac{n}{2m}$-group m-nonseparable functions (f_9–f_{13});
4) $\frac{n}{m}$-group m-nonseparable functions(f_{14}–f_{18});
5) nonseparable functions (f_{19}–f_{20});

where n is the dimensionality of the problem and m is the number of variables in each nonseparable subcomponent. For this research, n and m are set to 1000 and 50, respectively.

A. Parameter Settings

The subcomponent optimizer used in this paper is SaNSDE [48], a variant of differential evolution (DE) [49]. SaNSDE self-adapts the crossover rate and the scaling factor of DE. The population size is 50 as suggested in [48]. All experimental results are based on 25 independent runs for each algorithm. The maximum number of fitness evaluations was set to 3×10^6 as suggested in [20]. We used these settings to compare our results with other algorithms that were benchmarked against the same test suite. For the grouping stage, the value of ϵ was arbitrarily set to 10^{-3} (Algorithm 1, line 14). Other values such as 10^{-1} and 10^{-6} were used to test the sensitivity of differential grouping to ϵ (see Section V-C).

V. ANALYSIS OF RESULTS

This section provides an analysis of the effectiveness of differential grouping in terms of identifying the interacting variables and a comparison with the CCVIL algorithm [30]. Experimental results are provided to analyze the performance of differential grouping in the context of a CC framework for large-scale optimization problems. Additionally, this section also shows how a CBCC can further enhance the optimization performance.

A. Performance of Differential Grouping

Table I shows the experimental results for the grouping performance of differential grouping and the CCVIL grouping algorithm. The entries of the two algorithms are separated by the symbol "/." The last column shows the grouping accuracies of nonseparable variables for both algorithms. The double lines separate different classes of functions according to the description in Section IV. This section focuses on the performance of differential grouping and the next section is devoted to comparison with CCVIL. It can be seen from Table I that the grouping accuracy for 13 out of 20 benchmark functions is 100%. For functions f_1 to f_3, which are fully separable (class 1, see Section IV), all the variables were placed in one separable group. Differential grouping correctly identified the decision variables of these functions as fully separable. Another possibility would have been to place each of the decision variables in a separate subcomponent. However, this is not necessarily an optimal grouping arrangement in terms of both efficiency and accuracy for a large-scale fully separable problem. Studies [21] and [24] have shown that an intermediate decomposition between these two extreme cases is more efficient. Since the focus of this paper is on the decomposition of nonseparable subcomponents, in all of our experiments, the separable variables identified by the differential grouping algorithm were placed into one common subcomponent.

For the second class of benchmark functions (f_4–f_8), where there is one nonseparable subcomponent with 50 variables and another separable group with 950 variables, the grouping accuracy for three out of these five functions is 100%. It may seem odd that the grouping accuracy on f_4 is reported

TABLE II

DETAILED GROUPING MATRIX OF SOME SELECTED FUNCTIONS ($\epsilon = 10^{-3}$). THE ROWS INDICATE THE GROUPS FORMED BY THE DIFFERENTIAL GROUPING ALGORITHM AND THE COLUMNS REPRESENT THE PERMUTATION GROUPS FROM WHICH THE VARIABLES IN EACH GROUP WERE EXTRACTED

Func	Groups	Group Size	P1	P2	P3	P4	P5	P6	P7	P8	P9	P10	P11	P12	P13	P14	P15	P16	P17	P18	P19	P20
f_4	G01	145	0	8	10	9	4	8	7	8	9	6	10	7	6	14	7	7	12	4	5	4
	G02	63	0	2	2	3	3	5	4	8	2	3	3	4	1	2	1	5	3	3	4	5
	G03	177	0	9	8	13	9	8	12	7	13	9	7	6	11	6	6	13	6	9	12	13
	G04	110	0	7	6	7	6	4	5	4	5	7	8	4	10	4	8	4	4	6	5	6
	G05	276	0	16	14	15	16	14	15	13	16	15	12	20	16	12	13	13	13	16	12	15
	G06	50	50	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	G07	27	0	1	3	0	3	0	1	2	1	0	2	1	1	2	1	3	1	1	3	1
	G08	89	0	6	3	2	4	8	2	6	3	4	5	6	3	5	7	3	7	5	5	5
	G09	28	0	1	0	1	4	0	1	1	1	2	1	1	0	2	5	2	3	2	1	0
	G10	2	0	0	1	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0
f_5	G01	50	50	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
f_7	G01	289	0	14	13	16	21	24	15	13	16	10	9	12	9	17	23	18	17	12	13	17
	G02	204	34	9	14	6	6	9	8	9	9	11	12	9	6	7	7	7	9	8	10	14
	G03	244	0	17	15	17	10	8	9	13	11	16	18	13	14	13	12	10	13	13	12	10
	G04	16	16	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
f_8	G01	225	0	18	11	13	9	8	10	11	14	13	9	10	14	15	7	12	13	12	14	12
	G02	306	0	11	18	13	20	14	19	16	18	19	15	19	17	17	16	14	22	8	13	17
	G03	286	0	14	16	17	17	20	12	14	12	12	17	16	14	13	17	16	8	20	16	15
	G04	46	46	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	G05	2	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
f_9	G01	50	0	0	0	0	0	0	50	0	0	0	0	0	0	0	0	0	0	0	0	0
	G02	50	0	0	0	50	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	G03	50	0	50	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	G04	50	0	0	0	0	0	0	0	0	0	50	0	0	0	0	0	0	0	0	0	0
	G05	50	0	0	0	0	50	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	G06	50	0	0	0	0	0	50	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	G07	50	0	0	50	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	G08	50	0	0	0	0	0	0	0	0	50	0	0	0	0	0	0	0	0	0	0	0
	G09	50	50	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	G10	50	0	0	0	0	0	0	0	50	0	0	0	0	0	0	0	0	0	0	0	0
f_{14}	G01	50	0	0	50	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	G02	50	0	0	0	50	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	G03	50	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	50	0	0	0	0
	G04	50	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	50	0	0	0
	G05	50	0	0	0	0	0	0	0	0	0	0	0	0	50	0	0	0	0	0	0	0
	G06	50	0	50	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	G07	50	0	0	0	0	50	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	G08	50	0	0	0	0	0	0	0	0	0	0	0	0	0	50	0	0	0	0	0	0
	G09	50	50	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	G10	50	0	0	0	0	0	0	0	0	0	0	0	0	0	0	50	0	0	0	0	0
	G11	50	0	0	0	0	0	0	0	0	0	0	0	50	0	0	0	0	0	0	0	0
	G12	50	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	50	0
	G13	50	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	50	0	0
	G14	50	0	0	0	0	0	50	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	G15	50	0	0	0	0	0	0	0	0	0	0	50	0	0	0	0	0	0	0	0	0
	G16	50	0	0	0	0	0	0	0	50	0	0	0	0	0	0	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	0	50
	G18	50	0	0	0	0	0	0	50	0	0	0	0	0	0	0	0	0	0	0	0	0
	G19	50	0	0	0	0	0	0	0	0	0	50	0	0	0	0	0	0	0	0	0	0
	G20	50	0	0	0	0	0	0	0	0	50	0	0	0	0	0	0	0	0	0	0	0
f_{20}	G01	70	2	0	1	5	5	4	5	2	2	4	3	3	2	3	4	5	5	5	3	7
	G02	15	2	0	2	1	0	0	0	0	1	0	0	1	2	0	2	2	0	1	1	0
	G03	402	21	18	22	20	16	19	22	25	22	22	24	18	23	17	18	18	14	19	21	23
	G04	2	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	1	0	0	0
	G05	115	4	7	2	7	7	8	5	6	8	3	4	5	7	11	4	9	3	7	5	3
	G06	83	6	7	6	4	4	5	4	4	1	3	3	1	3	5	7	6	4	3	2	5
	G07	2	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	1
	G08	2	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	1	0	0	0	0
	G09	34	2	5	2	1	3	1	3	1	2	0	1	4	4	0	1	1	1	1	1	0
	G10	2	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	1	0	0	0	0
	G11	2	0	0	1	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0
	G12	74	3	5	3	3	7	5	3	1	2	3	6	6	2	3	6	3	7	0	3	3
	G13	84	4	5	5	4	3	3	2	5	3	6	2	7	1	2	5	1	10	9	4	3
	G14	2	0	0	0	0	0	0	0	0	0	0	1	0	0	0	1	0	0	0	0	0
	G15	52	3	1	2	2	1	2	3	2	5	4	5	3	3	5	1	1	3	1	4	1
	G16	17	3	0	2	0	2	0	0	1	2	1	0	0	0	2	0	0	0	0	0	4

to be 100% while the number of groups is incorrect. The reason for this is that, although the number of groups is not correct, all 50 nonseparable variables were correctly grouped into a common subcomponent. The reason for the incorrect number of groups is that the algorithm unexpectedly subdivided some separable variables into nine other nonseparable groups. Since the grouping of separable variables does not affect the nonseparable ones, we report 100% accuracy as long as the nonseparable variables were not misplaced. Further details of how the variables were grouped is shown in Table II for some representative functions. The grouping accuracy for the remaining two functions in this category (f_7 and f_8) is also acceptable, especially for f_8 where only four variables were misplaced. In the case of f_7, 16 variables were mixed with separable variables in a total of four nonseparable groups.

For the next set of functions (f_9–f_{13}), there are ten nonseparable subcomponents, each with 50 variables and one separable subcomponent with 500 variables. Except for f_{13}, the grouping accuracy for this class is very high, with an accuracy of 100% for 3 functions out of the five. The grouping accuracy on f_{11} is 99.2% because one nonseparable variable

TABLE III
RESULTS OF DIFFERENTIAL GROUPING WITH PARAMETER ϵ SET TO 10^{-1} AND 10^{-6} ON CEC'2010 BENCHMARK FUNCTIONS (SEPARATED BY ''/'')

Function	Sep Vars	Non-sep Vars	Non-sep Groups	Differential Grouping $(\epsilon = 10^{-1})$ / Differential Grouping $(\epsilon = 10^{-6})$					
				# Captured Sep Vars	# Captured Non-sep Vars	# Formed Non-sep Groups	# Misplaced Vars	# FE	Grouping Accuracy
f_1	1000	0	0	1000 / 96	0 / 904	0 / 20	0 / 904	1001000 / 25036	100% / 9.6%
f_2	1000	0	0	1000 / 1000	0 / 0	0 / 0	0 / 0	1001000 / 1001000	100% / 100%
f_3	1000	0	0	1000 / 862	0 / 138	0 / 4	0 / 138	1001000 / 757476	100% / 86.2%
f_4	950	50	1	34 / 34	50 / 50	9 / 9	0 / 0	14546 / 14546	100% / 100%
f_5	950	50	1	950 / 950	50 / 50	1 / 1	0 / 0	905450 / 905450	100% / 100%
f_6	950	50	1	950 / 732	50 / 50	1 / 7	0 / 0	906332 / 562748	100% / 100%
f_7	950	50	1	950 / 247	50 / 34	1 / 4	0 / 16	906822 / 67250	100% / 68%
f_8	950	50	1	135 / 135	46 / 46	5 / 5	4 / 4	23608 / 23608	92% / 92%
f_9	500	500	10	500 / 26	500 / 128	10 / 15	0 / 327	270802 / 9350	100% / 25.6%
f_{10}	500	500	10	500 / 500	500 / 500	10 / 10	0 / 0	272958 / 272958	100% / 100%
f_{11}	500	500	10	512 / 158	291 / 500	36 / 14	209 / 0	329938 / 42186	58.2% / 100%
f_{12}	500	500	10	500 / 500	500 / 492	10 / 10	0 / 8	271390 / 271182	100% / 98.4%
f_{13}	500	500	10	500 / 131	27 / 126	173 / 40	473 / 374	636686 / 49470	5.4% / 25.2%
f_{14}	0	1000	20	0 / 1	1000 / 407	20 / 19	0 / 593	21000 / 10574	100% / 40.7%
f_{15}	0	1000	20	1 / 0	999 / 1000	20 / 20	1 / 0	21012 / 21000	99.9% / 100%
f_{16}	0	1000	20	20 / 0	640 / 1000	72 / 20	360 / 640	46476 / 21000	64% / 100%
f_{17}	0	1000	20	0 / 0	1000 / 506	20 / 20	0 / 494	21000 / 11490	100% / 50.6%
f_{18}	0	1000	20	79 / 85	60 / 173	359 / 49	940 / 827	383540 / 34230	6% / 17.3%
f_{19}	0	1000	1	0 / 0	1000 / 1000	1 / 1	0 / 0	2000 / 2000	100% / 100%
f_{20}	0	1000	1	0 / 42	40 / 82	500 / 16	960 / 918	501000 / 22206	4% / 8.2%

was misplaced. A further experiment revealed that with a smaller value of ϵ in Algorithm 1, it is possible to perform a fully accurate grouping on this function (see Section V-C). The fourth category of functions (f_{14}–f_{18}) has a very similar grouping accuracy as the previous group. Note that there are no separable subcomponents in these functions and all 20 subcomponents are nonseparable.

In the last category, where all the variables interact with each other, the grouping accuracy for f_{19} is 100% and all the variables were correctly placed into one big group. However, the grouping accuracy for f_{20} is poor.

An interesting pattern that can be seen in Table I is the overall low grouping accuracy on almost all instances of the Rosenbrock function (f_8, f_{13}, f_{18}, and f_{20}). For example, in the case of f_{13} and f_{18}, 40 and 49 nonseparable groups were formed where there are only 10 and 20 such subcomponents. A detailed investigation on this behavior is beyond the scope of the current paper.

Table II shows in detail how the subcomponents were found for a number of functions. Each row shows a nonseparable group, which is formed by differential grouping. The column group size shows the size of each group. Columns (P1–P20) are permutation groups that contain the indices of 50 randomly chosen dimensions. The numbers in each column shows how many variables of a group belong to each permutation group. For example, in the case of f_4 from a total of 145 variables in the first group, none is from P1, 8 is from P2, 10 is from P3, and so forth. The numbers in columns P1 to P20 add up to the group size. This function has only one 50-D nonseparable subcomponent, which is represented by P1 and one 950-D separable subcomponent which is represented by the remaining permutation groups. It can be seen that in group 6 (G06), 50 variables are from P1 and none from the rest of the permutation groups. Take f_9 as another example. For this function, an ideal grouping should form ten nonseparable groups with 50 variables from the first ten permutation groups (P1–P10) and none from the remaining permutation groups. Table II shows how differential grouping formed such an ideal grouping for this function.

One final remark about the performance of differential grouping relates to the number of fitness evaluations used for each function to discover the underlying grouping structure. It can be seen from Table I that the number of required fitness evaluations to identify the interaction structure for fully separable functions (f_1–f_3) is relatively high. The reason for this behavior is that to find out whether a variable interacts with another variable, a pairwise comparison is required over all decision variables. In each full scan of all variables no interaction was detected and only one variable was excluded from the list of all decision variables. As a result, approximately $n \times (n + 1)$ fitness evaluations were required. This is a special case of the result obtained in Section III-B where $m = 1$. For the second class of functions (f_4–f_8), slightly fewer fitness evaluations were needed because in the first scan, 50 variables were extracted for each accurate grouping. This effect is also present in the fourth group where there are 20 nonseparable groups. The least number of fitness evaluations was required for f_{19} where all the variables were excluded in the first pass of the algorithm. This behavior is implied by the complexity analysis presented in Section III-B.

B. Comparison With CCVIL

This section discusses the similarities and differences between differential grouping and another recently proposed automatic grouping procedure, CCVIL [30].

CCVIL is a two-stage algorithm where the grouping structure is discovered prior to the optimization stage. However, unlike the technique proposed in this paper, the grouping stage of CCVIL is also based on a CC framework. According

to [30], two variables x_i and x_j are said to interact with each other if the following condition holds:

$$\begin{aligned}&\exists \vec{x}, x_i', x_j' : \\ &f(x_1, ..., x_i, ..., x_j, ..., x_n) < f(x_1, ..., x_i', ..., x_j, ..., x_n) \wedge \\ &f(x_1, ..., x_i, ..., x_j', ..., x_n) > f(x_1, ..., x_i', ..., x_j', ..., x_n) \quad (16)\end{aligned}$$

where $\vec{x}$ is a candidate decision vector and x_i', x_j' are to be replaced by the ith and jth decision variables, respectively. The way these two values are chosen is similar to the method proposed by Weicker and Weicker [38]. However, the approach taken by CCVIL is more accurate and reduces the number of falsely detected interactions.

CCVIL initially places each variable in a separate subcomponent. Then, by repeatedly applying the above equation to any two variables x_i and x_j, the subcomponents containing the interacting variables are merged until the termination criteria is met.

Since the focus of this paper is on proposing a decomposition algorithm, we omit the details of the optimization stage of the CCVIL algorithm. The interested reader is referred to [30].

Table I shows the performance of CCVIL on the CEC'2010 benchmark functions (right-hand side entries). It can be seen from the table that differential grouping provides a more accurate grouping with considerably fewer fitness evaluations on most of the functions. Exceptions are f_1, f_2, and f_7. It is notable that, like differential grouping, CCVIL also behaved differently on all instances of the Rosenbrock function. Indeed, CCVIL classified all variants of the Rosenbrock function as fully separable functions. An advantage of CCVIL is its ability to quickly detect fully separable variables with a relatively low number of fitness evaluations, whereas in differential grouping, approximately one million fitness evaluations were required to discover the underlying grouping structure.

An example shows why differential grouping detects interacting variables much faster than the CCVIL algorithm. Fig. 1 shows three regions A, B, and C on the contour plot of a 2-D Schwefel's Problem 1.2 (lighter areas have smaller function values). For this function, both variables interact over the entire search space. The condition given in (16) is only satisfied by points in region A. If the points are in regions B or C, the condition will be false and the algorithm will need to search further to find values of the decision variables that satisfy (16). This kind of behavior is expected since (16) uses an existential quantifier, and the amount of search effort required to find a set of points to satisfy the criteria in (16) is unknown. In order to alleviate this problem a stochastic approach is taken in [30]. If an interaction is not found by (16) after a small number of applications, the probability of there being an interaction becomes very small and the search is terminated.

Since differential grouping approximates the gradient, it uses a more accurate measure for detecting interacting variables without excessive search. Unlike CCVIL, which directly compares the fitness of the sample points, in differential grouping, the changes in the fitness values are compared to detect whether there is an interaction. As shown in Fig. 1, differential grouping compares the differences between the elevations of the two pairs of points $|f(x_1, x_2) - f(x_1 + \delta, x_2)|$ and $|f(x_1, x_2') - f(x_1 + \delta, x_2')|$, as shown by the dashed lines. If these two values are different, it is inferred that the corresponding dimensions are nonseparable. In other words, this is like forming a difference equation based on (4) ($\Delta_{x_1}[f]$) and evaluating it for two different values of x_2 and comparing the results. The figure shows that, regardless of the chosen region, differential grouping can detect an interaction in its first attempt. However, differential grouping may fail when a portion of the search space is fully separable while other regions are fully nonseparable. In such scenarios, if all four chosen points fall inside the separable region, the interaction will not be detected, but if at least one point falls in the nonseparable region, the interaction will be correctly detected. The situation is even worse with CCVIL, because even if at least one of the four points falls inside the nonseparable region, it is still not guaranteed that (16) is satisfied. For most of CEC'2010 test functions, the interaction occurs over the whole search space, and this is why differential grouping managed to accurately and efficiently detect the interactions.

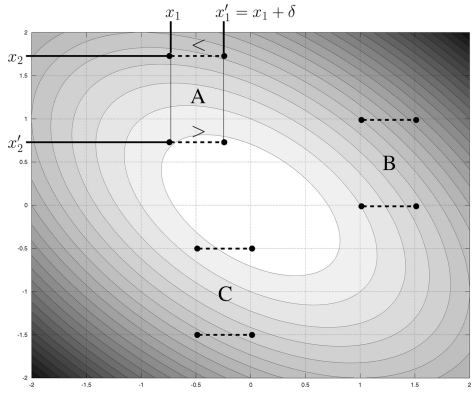


Fig. 1. Detection of interacting variables using differential grouping and CCVIL on different regions of a 2-D Schwefel Problem 1.2.

The results in Table I clearly show that differential grouping is superior to CCVIL. It is clear that if the same subcomponent optimizer is used under identical conditions, it is highly likely that the algorithm with the better grouping would perform better in the optimization stage. The fact that differential grouping had roughly twice as many fitness evaluations for the optimization stage also increased this possibility.

C. Sensitivity Analysis

In order to test the sensitivity of differential grouping to the parameter ϵ, the differential grouping algorithm was tested with two additional ϵ values, the result of which is reported in Table III. Therefore, by considering the results provided in Table I, differential grouping was tested with three different ϵ values which are: 10^{-1}, 10^{-3}, and 10^{-6}.

As it can be seen from Tables I and III, the differential grouping algorithm with three different ϵ values consistently outperform CCVIL. This shows that the performance of differential grouping is not very sensitive to this parameter as long as it is set to a relatively small value. A general trend that can be seen is that more separable variables are correctly classified when a larger ϵ (10^{-1}) is used. This behavior is evident in functions f_1–f_{13}, which have a separable

TABLE IV
DESCRIPTION OF VARIOUS ALGORITHMS THAT ARE USED IN THE EMPIRICAL STUDIES

Algorithms	Description	Decomposition Strategy
DECC-G [25]	Differential Evolution with Cooperative Co-evolution and Random Grouping.	Random Grouping
MLCC [29]	Similar to DECC-G, but instead of using a fixed grouping a set of potential group sizes is used.	Random Grouping
DECC-D [28]	Differential Evolution with Cooperative Co-evolution and Delta Grouping.	Delta Grouping
DECC-DML [28]	Similar to DECC-D uses delta grouping, but similar to MLCC uses a set of potential group sizes.	Delta Grouping
DECC-I	DECC using an ideal grouping which is performed manually using the knowledge of benchmark functions.	Ideal Grouping
DECC-DG	DECC using the differential grouping that we proposed in this paper.	Differential Grouping
CBCC-DG	Contribution Based Cooperative Co-evolution [15] with differential grouping.	Differential Grouping
MA-SW-Chains [50]	The top rank algorithm in the IEEE CEC'210 competition on large-scale global optimization.	N/A

subcomponent. When a smaller ϵ (10^{-3}) was used differential grouping was able to identify the interacting variables with a higher accuracy. This is evident in functions f_{14}–f_{20} where there is no separable subcomponent. However, when ϵ is too small (10^{-6}), more separable variables were classified as nonseparable. This might be due to the precision error in calculating the difference of the delta values. In the current implementation of Algorithm 1, choosing a very small ϵ may influence the grouping accuracy of nonseparable variables. This is because in each scan of the decision variables in Algorithm 1, all the variables that are found to interact with the variable in examination are extracted from the set of decision variables and grouped in a common subcomponent. Therefore, a wrongly detected interaction between a separable variable and a set of nonseparable variables may break a nonseparable subcomponent into a set of smaller groups, which reduces the overall accuracy of the grouping. Examples of such a drop in grouping accuracy due to a very small ϵ (10^{-6}) are f_7, f_9.f_{12}, f_{14}, and f_{17} in Table III. Taking f_9 as an example, Table III shows that decreasing ϵ causes the grouping accuracy to drop from 100% to 25.6%. This function has 500 separable variables, but when $\epsilon = 10^{-6}$ the differential grouping algorithm misclassified these variables into a set of nonseparable groups. This reduced the number of correctly classified separable variables from 500 to only 26 variables. Consequently, since variable interaction is a two-way relationship, nonseparable subcomponents are divided up into smaller groups due to the interference of separable variables. For this reason instead of forming ten nonseparable subcomponents, differential grouping formed 15 nonseparable subcomponents. In short, the above observations show that despite the changes in the grouping accuracy due to variations in ϵ, the performance of differential grouping especially on nonseparable functions is high. Even in the case that ϵ is very small (10^{-6}), out of 20 test functions, differential grouping outperformed CCVIL on 11 functions, performed equally well on three functions, but performed worse on six functions. It should be noted that on the functions where CCVIL has a better performance, two functions are fully separable.

D. Differential Grouping With CC

In this section, we present the experimental results for CC with differential grouping and compare it against other similar algorithms with various decomposition strategies. Specifically, we compare differential grouping with random grouping [25], delta grouping [28], and an ideal grouping that was constructed manually using knowledge of the benchmark functions. All of the algorithms used in our empirical studies are summarized in Table IV. The experimental results can be found in Table V. The entries shown in bold are significantly better than other algorithms as determined by a two-sided Wilcoxon test with a confidence interval of 95%.

TABLE V
COMPARISON OF DIFFERENTIAL GROUPING AND OTHER GROUPING TECHNIQUES ON THE CEC'2010 BENCHMARK FUNCTIONS USING 25 INDEPENDENT RUNS. THE HIGHLIGHTED ENTRIES ARE SIGNIFICANTLY BETTER (WILCOXON TEST, $\alpha = 0.05$)

Functions		DECC-DG	MLCC	DECC-D	DECC-DML	DECC-I
f_1	Mean	5.47e+03	1.53e-27	1.01e-24	1.93e-25	1.73e+00
	Std	2.02e+04	7.66e-27	1.40e-25	1.86e-25	2.55e+00
f_2	Mean	4.39e+03	5.57e-01	2.99e+02	**2.17e+02**	4.40e+03
	Std	1.97e+02	2.21e+00	1.92e+01	2.98e+01	1.90e+02
f_3	Mean	1.67e+01	9.88e-13	**1.81e-13**	**1.18e-13**	1.67e+01
	Std	3.34e-01	3.70e-12	6.68e-15	8.22e-15	3.75e-01
f_4	Mean	4.79e+12	9.61e+12	**3.99e+12**	**3.58e+12**	6.13e+11
	Std	1.44e+12	3.43e+12	1.30e+12	1.54e+12	2.08e+11
f_5	Mean	**1.55e+08**	3.84e+08	4.16e+08	2.98e+08	1.34e+08
	Std	2.17e+07	6.93e+07	1.01e+08	9.31e+07	2.31e+07
f_6	Mean	**1.64e+01**	1.62e+07	1.36e+07	7.93e+05	1.64e+01
	Std	2.71e-01	4.97e+06	9.20e+06	3.97e+06	2.66e-01
f_7	Mean	**1.16e+04**	6.89e+05	6.58e+07	1.39e+08	2.97e+01
	Std	7.41e+03	7.37e+05	4.06e+07	7.72e+07	8.59e+01
f_8	Mean	**3.04e+07**	4.38e+07	5.39e+07	**3.46e+07**	3.19e+05
	Std	2.11e+07	3.45e+07	2.93e+07	3.56e+07	1.10e+06
f_9	Mean	**5.96e+07**	1.23e+08	**6.19.+07**	**5.92e+07**	4.84e+07
	Std	8.18e+06	1.33e+07	6.43e+06	4.71e+06	6.56e+06
f_{10}	Mean	**4.52e+03**	**3.43e+03**	1.16e+04	1.25e+04	4.34e+03
	Std	1.41e+02	8.72e+02	2.68e+03	2.66e+02	1.46e+02
f_{11}	Mean	1.03e+01	1.98e+02	4.76e+01	**1.80e-13**	1.02e+01
	Std	1.01e+00	6.98e-01	9.53e+01	9.88e-15	1.13e+00
f_{12}	Mean	**2.52e+03**	3.49e+04	1.53e+05	3.79e+06	1.47e+03
	Std	4.86e+02	4.92e+03	1.23e+04	1.50e+05	4.28e+02
f_{13}	Mean	4.54e+06	2.08e+03	**9.87e+02**	**1.14e+03**	7.51e+02
	Std	2.13e+06	7.27e+02	2.41e+02	4.31e+02	3.70e+02
f_{14}	Mean	3.41e+08	3.16e+08	1.98e+08	**1.89e+08**	3.38e+08
	Std	2.41e+07	2.77e+07	1.45e+07	1.49e+07	2.40e+07
f_{15}	Mean	**5.88e+03**	7.11e+03	1.53e+04	1.54e+04	5.87e+03
	Std	1.03e+02	1.34e+03	3.92e+02	3.59e+02	9.89e+01
f_{16}	Mean	**7.39e-13**	3.76e+02	1.88e+02	5.08e-02	2.47e-13
	Std	5.70e-14	4.71e+01	2.16e+02	2.54e-01	9.17e-15
f_{17}	Mean	**4.01e+04**	1.59e+05	9.03e+05	6.54e+06	3.91e+04
	Std	2.85e+03	1.43e+04	5.28e+04	4.63e+05	2.75e+03
f_{18}	Mean	1.11e+10	7.09e+03	**2.12e+03**	**2.47e+03**	1.17e+03
	Std	2.04e+09	4.77e+03	5.18e+02	1.18e+03	9.66e+01
f_{19}	Mean	**1.74e+06**	1.36e+06	1.33e+07	1.59e+07	1.74e+06
	Std	9.54e+04	7.35e+04	1.05e+06	1.72e+06	9.54e+04
f_{20}	Mean	4.87e+07	2.05e+03	**9.91e+02**	**9.91e+02**	4.14e+03
	Std	2.27e+07	1.80e+02	2.61e+01	3.51e+01	8.14e+02

Table V shows that DECC-DG outperformed other algorithms. The performance of DECC-DML is very similar to that of DECC-DG. On closer inspection, one can see that it outperformed DECC-DG on all separable functions. However,

on nonseparable functions, DECC-DG outperformed DECC-DML when the grouping accuracy (see Table I) is high. The same trend continues when comparing DECC-DG against DECC-D. Another observation is that the performance of DECC-DG is either worse or the same as DECC-DML on instances of rotated elliptic functions (f_4, f_9, and f_{14}) even though according to Table I differential grouping discovered the optimal grouping.

In order to show how DECC-DG compares against an ideal decomposition, the performance of DECC-I is also reported. The grouping for DECC-I was done manually using prior knowledge of the benchmark functions. Although this is not a fair comparison, it serves as a good benchmark for evaluating the performance of differential grouping. It is not fair because the grouping information were provided to DECC-I and all the allotted fitness evaluations were used for optimization, whereas in the case of DECC-DG, a considerable number of fitness evaluations had to be used to discover the grouping structure, and the remaining fitness evaluations were used for the actual optimization.

Fig. 2 shows the convergence behavior of different algorithms. Each point on the plot was calculated by taking the average of 25 independent runs. Although for some functions a considerable number of fitness evaluations were used to discover the grouping structure, this effort was compensated for during the optimization stage. In Figs. 2 and 3, the algorithms that use differential grouping initially do not have any improvement for some number of iterations, but once the grouping structure is identified there is a significant improvement thereafter. Note that there are some other algorithms in the convergence plots that will be discussed in Section V-E

Overall, the experimental results in Table V and the convergence plots in Fig. 2 show that using an automatic grouping that can identify the underlying structure of the benchmark functions (in terms of nonseparability of the decision variables) is highly beneficial, and it is advantageous to spend some fraction of the computational budget to find such a structure before running the optimizers.

E. Differential Grouping With CBCC

This section shows how the performance of DECC-DG can be improved by using a CBCC instead of traditional CC where the subcomponents are optimized in a round-robin fashion.

It has been shown recently that considerably better solutions can be obtained by spending more computational budget on the subcomponents with a higher contribution toward the global fitness [15]. One general assumption in CBCC is that the interdependencies between subcomponents should be kept to a minimum. The algorithm proposed in [15] relies on a manual grouping of the decision variables to show that a contribution scheme is beneficial. The differential grouping algorithm proposed in this paper allows the use of a CBCC without relying on a manual decomposition of the decision variables. In the remainder of this section the following two comparisons are made: 1) the traditional CC is compared with CBCC (DECC-DG versus CBCC-DG); and 2) CBCC-DG is compared with the MA-SW-Chains [50] algorithm, the top ranked algorithm in the IEEE CEC'2010 special session and competition on large-scale global optimization (CBCC-DG versus MA-SW-Chains).

TABLE VI

COMPARISON OF TRADITIONAL CC WITH CBCC WITH DIFFERENTIAL GROUPING ON THE CEC'2010 BENCHMARK FUNCTIONS USING 25 INDEPENDENT RUNS. THE HIGHLIGHTED ENTRIES ARE SIGNIFICANTLY BETTER (WILCOXON TEST, $\alpha = 0.05$). THE ENTRIES MARKED WITH THE SYMBOL '‡' ARE USED TO COMPARE CBCC WITH MA-SW-CHAINS

Functions		DECC-DG	CBCC1-DG	CBCC2-DG	MA-SW-Chains [50]
f_1	Mean	5.47e+03	1.32e+04	8.34e+03	2.10e-14‡
	Std	2.02e+04	6.25e+04	3.41e+04	1.99e-14
f_2	Mean	**4.39e+03**	4.44e+03	4.44e+03	8.10e+02‡
	Std	1.97e+02	1.60e+02	1.80e+02	5.88e+01
f_3	Mean	1.67e+01	1.66e+01	1.67e+01	7.28e-13‡
	Std	3.34e-01	3.79e-01	3.28e-01	3.40e-13
f_4	Mean	4.79e+12	**2.31e+12**	**2.36e+12**	3.53e+11‡
	Std	1.44e+12	7.43e+11	7.92e+11	3.12e+10
f_5	Mean	1.55e+08	**1.35e+08**‡	**1.36e+08**‡	1.68e+08
	Std	2.17e+07	2.18e+07	2.46e+07	1.04e+08
f_6	Mean	1.64e+01	1.65e+01‡	1.64e+01‡	8.14e+04
	Std	2.71e-01	3.99e-01	3.46e-01	2.84e+05
f_7	Mean	**1.16e+04**	1.81e+04	1.35e+04	1.03e+02‡
	Std	7.41e+03	4.59e+04	3.92e+04	8.70e+01
f_8	Mean	3.04e+07	**3.34e+06**‡	**8.70e+05**‡	1.41e+07
	Std	2.11e+07	2.29e+06	1.71e+06	3.68e+07
f_9	Mean	**5.96e+07**	6.79e+07	7.97e+07	1.41e+07‡
	Std	8.18e+06	6.92e+06	1.08e+07	1.15e+06
f_{10}	Mean	4.52e+03	**4.01e+03**	**4.04e+03**	2.07e+03‡
	Std	1.41e+02	1.37e+02	1.21e+02	1.44e+02
f_{11}	Mean	1.03e+01	1.05e+01‡	1.03e+01‡	3.80e+01
	Std	1.01e+00	9.31e-01	8.47e-01	7.35e+00
f_{12}	Mean	**2.52e+03**	4.19e+03	4.00e+03	3.62e-06‡
	Std	4.86e+02	1.25e+03	8.63e+02	5.92e-07
f_{13}	Mean	4.54e+06	**9.10e+03**	**4.54e+03**	1.25e+03‡
	Std	2.13e+06	3.75e+03	1.91e+03	5.72e+02
f_{14}	Mean	**3.41e+08**	3.64e+08	3.69e+08	3.11e+07‡
	Std	2.41e+07	2.61e+07	2.42e+07	1.93e+06
f_{15}	Mean	5.88e+03	5.89e+03	5.88e+03	2.74e+03‡
	Std	1.03e+02	9.10e+01	8.81e+01	1.22e+02
f_{16}	Mean	7.39e-13	**3.08e-12**‡	**4.44e-12**‡	9.98e+01
	Std	5.70e-14	3.19e-13	4.22e-13	1.40e+01
f_{17}	Mean	**4.01e+04**	4.50e+04	4.73e+04	1.24e+00‡
	Std	2.85e+03	3.18e+03	2.77e+03	1.25e-01
f_{18}	Mean	1.11e+10	**1.34e+09**	**3.47e+08**	1.30e+03‡
	Std	2.04e+09	4.94e+08	1.39e+08	4.36e+02
f_{19}	Mean	1.74e+06	1.74e+06	1.74e+06	2.85e+05‡
	Std	9.54e+04	8.46e+04	8.46e+04	1.78e+04
f_{20}	Mean	4.87e+07	**9.53e+04**	**8.42e+03**	1.07e+03‡
	Std	2.27e+07	1.02e+05	2.36e+03	7.29e+01

DECC-DG versus CBCC-DG: Table VI presents the results on CEC'2010 benchmark functions. It is noteworthy that CBCC1 and CBCC2 only differ in the policy that they use to divide the computational budget between the subcomponents. The reader is referred to [15] for the details of these algorithms. For the purposes of this paper, it is sufficient to note that in both CBCC1 and CBCC2 the subcomponents with a higher contribution to the global fitness are given more of the computational budget. This can be contrasted with traditional CC, where the computational budget is equally divided between all subcomponents.

Table VI shows that both instances of the CBCC algorithm outperformed traditional cooperative co-evolution (DECC) where differential grouping was used as the decomposition procedure. At first glance, it might seem that DECC-DG and

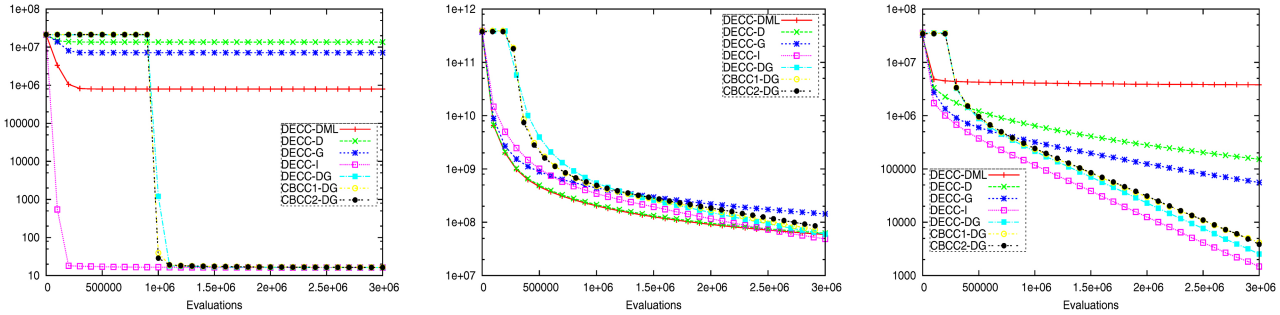


Fig. 2. Convergence plots of various algorithms on selected CEC'2010 benchmark functions. The plots are generated from 25 independent runs. (a) f_6. (b) f_9. (c) f_{12}.

CBCC-DG have similar performance. However, if we look at the results according to the classes of functions that were described in Section IV, we can see that the CBCC algorithm outperforms DECC-DG on the second class of functions (f_4–f_8). With respect to the other classes (f_9–f_{18}), none of the algorithms clearly outperforms the others. This behavior was reported in [15] and was attributed to the fact that there were equal contributions from all subcomponents. It is expected that CBCC will perform as well as DECC-DG in situations where there are equal contributions.

It is arguable that in most real-world problems some imbalance will exist between various subcomponents. In such cases, equal contribution to the global fitness is unlikely. In order to properly benchmark the performance of CBCC-DG on imbalanced problems, modified functions f_9–f_{18} to create artificial imbalance between the subcomponents. The effect of functions with a higher degree of imbalance is analyzed in the next section.

CBCC-DG versus MA-SW-Chains: In comparing CBCC-DG and MA-SW-chains (see Table VI), the symbol "‡" is used to indicate which algorithm performed significantly better than the other. Table VI shows that MA-SW-chains outperformed CBCC-DG algorithms on 15 out of 20 functions. However, two important facts should be noted here. First, there is no imbalance between the subcomponents of the functions f_9–f_{18}, but if we look at the performance of CBCC on f_4–f_8, which do have imbalance, it can be seen that CBCC is slightly better than MA-SW-Chains. Second, the optimizers used here are different in nature. It has been established that if any subcomponent optimizer is used with a CC framework using differential grouping as the decomposition method, the performance will be greatly enhanced.

F. Effect of More Imbalance

To further investigate the effect of imbalance, the functions in categories 3 and 4 can be modified in the following way:

$$F_{\text{cat3}} = \sum_{i=0}^{\frac{n}{2m}-1} 10^{2(i-9)} \times F_{\text{nonsep}} + F_{\text{sep}}$$

$$F_{\text{cat4}} = \sum_{i=0}^{\frac{n}{m}-1} 10^{(i-9)} \times F_{\text{nonsep}} + F_{\text{sep}} \ .$$

TABLE VII

EXPERIMENTAL RESULTS FOR IMBALANCED FUNCTIONS. THE EXPERIMENTS ARE BASED ON MODIFIED BENCHMARK FUNCTIONS USING 25 INDEPENDENT RUNS. THE HIGHLIGHTED ENTRIES ARE SIGNIFICANTLY BETTER (WILCOXON TEST, $\alpha = 0.05$). THE ENTRIES MARKED WITH THE SYMBOL '‡' ARE USED TO COMPARE CBCC WITH MA-SW-CHAINS

Functions		DECC-DG	CBCC1-DG	CBCC2-DG	MA-SW-Chains [50]
f'_9	Mean	3.81e+11	**2.18e+11**	**2.04e+11**	8.43+10‡
	Std	1.15e+11	5.51e+10	5.16e+10	3.21+10
f'_{10}	Mean	2.89e+07	**2.36e+07**‡	**2.43e+07**‡	2.24+07‡
	Std	1.51e+06	1.79e+06	1.89e+06	1.34+07
f'_{11}	Mean	**9.98e+00**	1.03e+01‡	1.05e+01‡	2.72+04
	Std	1.16e+00	1.11e+00	8.93e-01	4.74+04
f'_{12}	Mean	1.62e+07	**6.27e+06**	**5.02e+06**	6.93+05‡
	Std	7.53e+06	7.27e+06	1.73e+06	4.24+05
f'_{13}	Mean	**1.07e+07**	**1.06e+07**	**1.04e+07**	4.31+06‡
	Std	4.77e+06	4.69e+06	4.08e+06	3.31+06
f'_{14}	Mean	1.11e+13	**6.62e+12**	**6.50e+12**	4.69+11‡
	Std	2.64e+12	4.20e+12	3.40e+12	5.38+10
f'_{15}	Mean	1.98e+08	**1.73e+08**‡	**1.73e+08**‡	1.30+08‡
	Std	6.84e+06	1.35e+07	1.19e+07	8.40+07
f'_{16}	Mean	**2.78e-08**	**2.98e-08**‡	4.22e-08‡	1.71+05
	Std	1.55e-08	1.33e-08	2.87e-08	1.32+05
f'_{17}	Mean	1.36e+09	**7.34e+08**	**6.86e+08**	3.13+07‡
	Std	2.88e+08	5.01e+08	4.23e+08	1.56+07
f'_{18}	Mean	**4.25e+10**	**4.33e+10**‡	5.64e+10‡	5.39+12
	Std	9.76e+09	1.23e+10	3.13e+10	9.59+12

The overall structure of the functions remains unchanged, but the contribution of a component is multiplied by a coefficient to create the imbalance effect. The third category (f_9–f_{13}) and the fourth category (f_{14}–f_{18}) take the form of F_{cat3} and F_{cat4}, respectively.

The experimental results using the modified set of benchmark problems are given in Table VII. A prime symbol is used to indicate the modified functions (such as f'_9).

It can be seen from Table VII that in the presence of imbalance, the CBCC-DG algorithm outperforms the DECC-DG algorithm with a wider gap. This shows that, in the absence of knowledge about the imbalance between subcomponents, CBCC-DG performs at least as well as DECC-DG. However, if such an imbalance exists, which we believe is highly likely in many real-world problems, CBCC-DG finds better solutions and outperforms traditional CC.

By comparing the performance of CBCC-DG and MA-SW-chains, it can be seen that both algorithms perform similarly

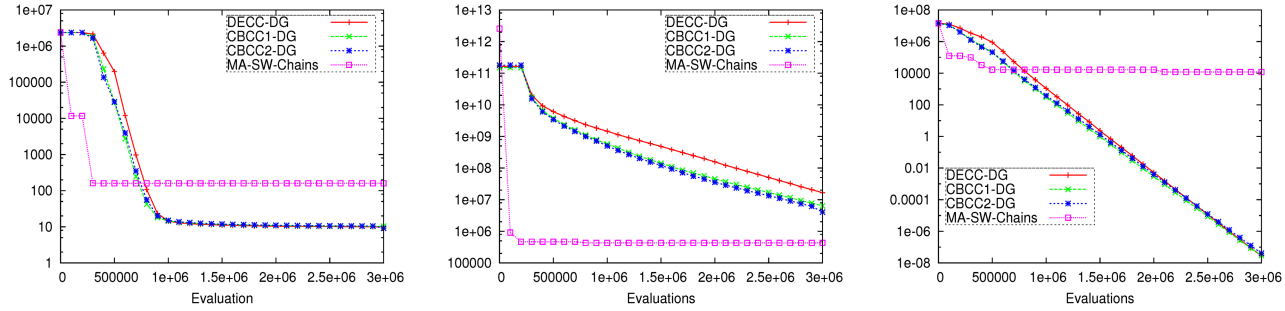


Fig. 3. Convergence of traditional and CBCC using differential grouping against MA-SW-chains on some representative imbalanced problems. (a) f'_{11}. (b) f'_{12}. (c) f'_{16}.

TABLE VIII
CBCC-DG'S NUMBER OF WINS, LOSSES, AND TIES AGAINST DECC-DG AND MA-SW-CHAINS BEFORE AND AFTER INCLUSION OF IMBALANCE IN BENCHMARK PROBLEMS. (BASED ON FUNCTIONS f_4–f_8, f'_9–f'_{18})

	Balanced			Imbalanced		
Algorithm	Wins	Loses	Ties	Wins	Loses	Ties
DECC-DG	7	5	3	9	2	4
MA-SW-Chains	5	10	0	6	7	2

on the imbalanced problems, but MA-SW-chains performs slightly better. This information is summarized in Table VIII. Since functions f_4-f_8 from CEC'2010 are also imbalanced, we have included them in Table VIII. For a better understanding of the behavior of both algorithms the convergence plots are shown in Fig. 3. In almost all cases, there is a drastic improvement in the value of the objective function and thereafter the fitness becomes stagnant. Since MA-SW-chains is a memetic algorithm [51], this behavior can be attributed to the local search performed during evolution. Both DECC-DG and CBCC-DG show a steady improvement in the global fitness and in the case of f'_{11} and f'_{16}, they overtake MA-SW-chains at some point during the evolutionary process. It can be seen that for some functions such as f'_{12} both DECC-DG and CBCC-DG show a steady improvement and it is possible that both algorithms would overtake MA-SW-chains with more evolutionary cycles. This suggests that given a limited number of fitness evaluations a local-search approach can find better solutions in the short term, but, in the long run, a CBCC co-evolutionary approach with differential grouping appears to be more stable and has the potential to further improve. This can be backed up by observing that CBCC-DG outperformed MA-SW-chains on almost all instances of multimodal functions. On nine imbalanced multimodal functions (f_5–f_8 and f'_9–f'_{18}) CBCC-DG outperformed MA-SW-chains on six, performed equally well on two and was worse on only one function.

VI. CONCLUSION

In this paper, we have proposed differential grouping, an automatic way of decomposing an optimization problem into a set of smaller problems where there are few or no interdependencies between the subcomponents. We have shown how differential grouping can be derived mathematically from the definition of additively separable functions. We have also shown how LINC-R [37] can be derived from our formulation. The proposed decomposition procedure has been evaluated using CEC'2010 benchmark functions and the results have shown that it is capable of grouping interacting variables with great accuracy for the majority of the benchmark functions. A comparative study with the grouping procedure of the CCVIL algorithm was conducted and the experimental results showed that differential grouping is superior to CCVIL both in terms of grouping accuracy and computational cost.

In order to evaluate the actual performance of differential grouping on optimization problems, we used the grouping structure identified by differential grouping in a CC framework for the optimization of large-scale additively separable functions. The experimental results revealed that the near-optimal grouping accuracy of differential grouping can greatly enhance the performance of optimization compared to the cases where the grouping is less accurate.

In the presence of an accurate grouping of decision variables, it is possible to accurately quantify the contribution of each of the subcomponents to the global fitness [15]. Once the contribution information is obtained, it is possible to divide the computational budget more wisely, according to the contribution of each subcomponent. Unlike traditional CC, where all subcomponents are given equal resources, in a contribution-based scheme subcomponents with higher contributions are given more resources. The differential grouping approach that is proposed in this paper makes it possible to accurately quantify the contributions.

It was shown that CBCC has the potential to greatly enhance the optimization performance for imbalanced problems. However, finding better strategies for allocation of computational resources to each of the subcomponents is the subject of future investigations.

Finally, we have shown for a given high performance evolutionary optimizer, it is possible to make it scale better to high dimensional problems by using it as a subcomponent optimizer in a contribution-based framework with differential grouping.

ACKNOWLEDGMENT

The authors would like to thank K. Tang and W. Chen for providing the source code of the CCVIL algorithm and also for their valuable comments. They would also like to thank K. Qin, J. Harland, V. Ciesielski, and M. Kirley for their valuable comments.

REFERENCES

[1] T. Weise, R. Chiong, and K. Tang, "Evolutionary optimization: Pitfalls and booby traps," *J. Comput. Sci. Technol.*, vol. 27, no. 5, pp. 907–936, 2012.

[2] R. Descartes, *Discourse on Method*, 1st ed. Prentice Hall, Upper Saddle River, NJ, USA, Jan. 1956.

[3] A. Griewank and P. L. Toint, "Local convergence analysis for partitioned quasi-Newton updates," *Numerische Mathematik*, vol. 39, no. 3, pp. 429–448, 1982.

[4] G. B. Dantzig and P. Wolfe, "Decomposition principle for linear programs," *Oper. Res.*, vol. 8, no. 1, pp. 101–111, 1960.

[5] D. Bertsekas, *Nonlinear Programming* (Optimization and Neural Computation Series). Belmont, MA, USA: Athena Scientific, 1995.

[6] T. Bäck, D. B. Fogel, and Z. Michalewicz, Eds., *Handbook of Evolutionary Computation*. Bristol, U.K./New York, NY, USA: Inst. Phys. Publishing/Oxford Univ. Press, 1997.

[7] R. Sarker, M. Mohammadian, and X. Yao, *Evolutionary Optimization* (International Series in Operations Research & Management Science), Boston, MA, USA: Kluwer Academic, vol. 48, 2003.

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

[9] R. E. Bellman, *Dynamic Programming* (Dover Books on Mathematics). Princeton, NJ, USA: Princeton Univ. Press, 1957.

[10] M. A. Potter and K. A. De Jong, "A cooperative coevolutionary approach to function optimization," in *Proc. Int. Conf. PPSN*, 1994, vol. 2, pp. 249–257.

[11] R. Salomon, "Reevaluating genetic algorithm performance under coordinate rotation of benchmark functions: A survey of some theoretical and practical aspects of genetic algorithms," *BioSystems*, vol. 39, no. 3, pp. 263–278, 1995.

[12] D. E. Goldberg, *Genetic Algorithms in Search, Optimization, and Machine Learning*. Reading, MA, USA: Addison-Wesley, 1989.

[13] Y. P. Chen, T. L. Yu, K. Sastry, and D. E. Goldberg, "A survey of linkage learning techniques in genetic and evolutionary algorithms," Illinois Genetic Algorithms Lib., Univ. Illinois, Urbana-Champaign, IL, USA, Tech. Rep. 2007014, Apr. 2007.

[14] D. Goldberg, B. Korb, and K. Deb, "Messy genetic algorithms: Motivation, analysis, and first results," *Complex Syst.*, vol. 3, no. 5, pp. 493–530, 1989.

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

[16] M. Ptashne, "How gene activators work," *Sci. Amer.*, vol. 260, no. 1, pp. 40–47, Jan. 1989.

[17] W. S. Klug, M. R. Cummings, C. Spencer, C. A. Spencer, and M. A. Palladino, *Concepts of Genetics*, 9th ed., Pearson, New York, USA. 2008.

[18] Y. Davidor, "Epistasis variance: Suitability of a representation to genetic algorithms," *Complex Syst.*, vol. 4, no. 4, pp. 369–383, 1990.

[19] A. Auger, N. Hansen, N. Mauny, R. Ros, and M. Schoenauer, "Bio-inspired continuous optimization: The coming of age," in *Proc. Invited Talk IEEE CEC*, Sep. 2007.

[20] K. Tang, X. Li, P. N. Suganthan, Z. Yang, and T. Weise. (2009). Benchmark functions for the CEC'2010 special session and competition on large-scale global optimization. Nature Inspired Comput. Applicat. Lab., University of Science and Technology of China. Hefei, China. [Online]. Available: http://goanna.cs.rmit.edu.au/~xiaodong/publications/lsgo-cec10.pdf

[21] 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.

[22] J. Kennedy and R. Eberhart, "Particle swarm optimization," in *Proc. IEEE Int. Conf. Neural Netw.*, vol. 4. 1995, pp. 1942–1948.

[23] R. Storn, "On the usage of differential evolution for function optimization," in *Proc. Biennial Conf. North Amer. Fuzzy Inform. Process. Soc.*, Jun. 1996, pp. 519–523.

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

[25] Z. Yang, K. Tang, and X. Yao, "Large scale evolutionary optimization using cooperative coevolution," *Inf. Sci.*, vol. 178, pp. 2986–2999, Aug. 2008.

[26] 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.

[27] M. N. Omidvar, X. Li, Z. Yang, and X. Yao, "Cooperative co-evolution for large scale optimization through more frequent random grouping," in *Proc. IEEE Congr. Evol. Comput.*, Jul. 2010, pp. 1754–1761.

[28] M. N. Omidvar, X. Li, and X. Yao, "Cooperative co-evolution with delta grouping for large scale non-separable function optimization," in *Proc. IEEE Congr. Evol. Comput.*, Jul. 2010, pp. 1762–1769.

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

[30] W. Chen, T. Weise, Z. Yang, and K. Tang, "Large-scale global optimization using cooperative coevolution with variable interaction learning," in *Proc. Int. Conf. PPSN*, 2011, LNCS 6239, pp. 300–309.

[31] T.-L. Yu, D. E. Goldberg, K. Sastry, C. F. Lima, and M. Pelikan, "Dependency structure matrix, genetic algorithms, and effective recombination," *Evol. Comput.*, vol. 17, pp. 595–626, Dec. 2009.

[32] J. H. Holland, *Adaptation in Natural and Artificial Systems*. Ann Arbor, MI, USA: Univ. Michigan Press, 1975.

[33] D. E. Goldberg, K. Deb, H. Kargupta, and G. Harik, "Rapid, accurate optimization of difficult problems using fast messy genetic algorithms," in *Proc. Int. Conf. Genetic Algorithms*, 1993, pp. 56–64.

[34] H. Kargupta, "The performance of the gene expression messy genetic algorithm on real test functions," in *Proc. IEEE Congr. Evol. Comput.*, May 1996, pp. 631–636.

[35] M. Munetomo and D. Goldberg, "A genetic algorithm using linkage identification by nonlinearity check," in *Proc. IEEE Int. Conf. Syst., Man, Cybern.*, vol. 1. 1999, pp. 595–600.

[36] M. Munetomo and D. E. Goldberg, "Linkage identification by non-monotonicity detection for overlapping functions," *Evol. Comput.*, vol. 7, pp. 377–398, Dec. 1999.

[37] M. Tezuka, M. Munetomo, and K. Akama, "Linkage identification by nonlinearity check for real-coded genetic algorithms," in *Proc. Genetic Evol. Comput. Conf.*, 2004, LNCS 3103, pp. 222–233.

[38] K. Weicker and N. Weicker, "On the improvement of coevolutionary optimizers by learning variable interdependencies," in *Proc. IEEE Congr. Evol. Comput.*, Jul. 1999, pp. 1627–1632.

[39] J. Smith and T. C. Fogarty, "An adaptive poly-parental recombination strategy," in *Sel. Papers From AISB Workshop on Evolutionary Computing*. London, U.K.: Springer-Verlag, 1995, pp. 48–61.

[40] G. R. Harik, "Learning gene linkage to efficiently solve problems of bounded difficulty using genetic algorithms," Ph.D. dissertation, Dept. Comput. Sci. Eng., University of Michigan, Ann Arbor, MI, USA, 1997.

[41] S. Baluja, "Population-based incremental learning: A method for integrating genetic search based function optimization and competitive learning," School Comput. Sci., Carnegie Mellon Univ., Pittsburgh, PA, USA, Tech. Rep. CMU-CS-94-163, 1994.

[42] H. Mühlenbein and G. Paaß, "From recombination of genes to the estimation of distributions I. Binary parameters," in *Proc. Int. Conf. PPSN*, 1996, pp. 178–187.

[43] G. Harik, F. Lobo, and D. Goldberg, "The compact genetic algorithm," *IEEE Trans. Evol. Comput.*, vol. 3, no. 4, pp. 287–297, Nov. 1999.

[44] M. Pelikan and D. E. Goldberg, "BOA: The Bayesian optimization algorithm," in *Proc. Genetic Evol. Comput. Conf.*, 1999, pp. 525–532.

[45] M. Pelikan, M. Pelikan, D. E. Goldberg, and D. E. Goldberg, "Escaping hierarchical traps with competent genetic algorithms," in *Proc. Genetic Evol. Comput. Conf.*, 2001, pp. 511–518.

[46] M. Pelikan, D. E. Goldberg, and S. Tsutsui, "Combining the strengths of Bayesian optimization algorithm and adaptive evolution strategies," in *Proc. Genetic Evol. Comput. Conf.*, 2002, pp. 512–519.

[47] P. L. Toint, "Test problems for partially separable optimization and results for the routine PSPMIN," Dept. Math., Univ. Namur, Namur, Belgium, Tech. Rep. 83/4, 1983.

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

[49] R. Storn and K. Price, "Differential evolution: A simple and efficient heuristic for global optimization over continuous spaces," *J. Global Optimiz.*, vol. 11, no. 4, pp. 341–359, 1995.

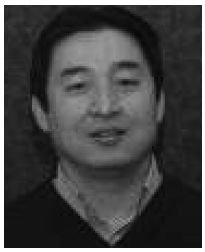
[50] D. Molina, M. Lozano, and F. Herrera, "MA-SW-Chains: Memetic algorithm based on local search chains for large scale continuous global optimization," in *Proc. IEEE Congr. Evol. Comput.*, Jul. 2010, pp. 3153–3160.

[51] P. Moscato, "On evolution, search, optimization, genetic algorithms and martial arts: Towards memetic algorithms," Caltech Concurrent Computation Program, California Inst. of Technol., CA, USA, [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.27.9474&rep=rep1&type=pdf, 1989.

Mohammad Nabi Omidvar (S'09) received the bachelor of computer science degree with first class honors from RMIT University, Melbourne, Australia, in 2010. He has been a member of the Evolutionary Computing and Machine Learning Group at RMIT University, Melbourne, Australia, since 2008, where he is currently working toward the Ph.D. degree in the field of evolutionary optimization and the bachelor program in applied mathematics.

His research interests include evolutionary computation and machine learning. His current research is on efficient optimization of large-scale partially separable real-valued functions.

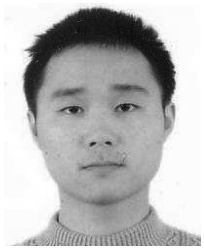
Mr. Omidvar has been a Student Member of ACM SIGEVO since 2009. He was a recipient of the Australian Postgraduate Award Scholarship in 2010 and also the Best Computer Science Honors Thesis Award from the School of Computer Science and IT, RMIT University in 2010.



Xiaodong Li (M'03–SM'07) received the B.Sc. degree in information science from Xidian University, Xi'an, China, in 1988, and the Dip. Com. and Ph.D. degrees in information science from the University of Otago, Dunedin, New Zealand, in 1992 and 1998, respectively.

He is currently an Associate Professor with the School of Computer Science and Information Technology, RMIT University, Melbourne, Australia. His research interests include evolutionary computation (in particular evolutionary multiobjective optimization, evolutionary optimization in dynamic environments, large scale optimization, and multimodal optimization), neural networks, complex systems, and swarm intelligence.

Dr. Li is an Associate Editor of the IEEE TRANSACTIONS ON EVOLUTIONARY COMPUTATION and the *International Journal of Swarm Intelligence Research*. He is currently Chair of the IEEE CIS Task Force on Large Scale Global Optimization and Vice-Chair of the IEEE CIS Task Force on Swarm Intelligence. He is a member of the editorial board of the *Journal of Swarm Intelligence* (Springer), and *Journal of Soft Computing* (Springer), and a member of the Technical Committee on Soft Computing, Systems, Man and Cybernetics Society, IEEE. He is an Advisor on the Scientific Advisory Board of *SolveIT Software*. He is a Vice-Chair of the IEEE Victorian Section CIS Chapter, Melbourne, Australia.



Yi Mei (S'09–M'13) received the bachelor's degree in mathematics from the University of Science and Technology of China (USTC), Hefei, China, in 2005, and the Ph.D. degree in computer science from the Nature Inspired Computation and Applications Laboratory, School of Computer Science and Technology, USTC, in 2010.

He is currently a Research Fellow with the School of Computer Science and Information Technology, RMIT University, Melbourne, Australia. His research interests include evolutionary algorithms, memetic algorithms, and other meta-heuristics with various real-world applications in the logistic area, such as arc routing problems, vehicle routing problems, and traveling salesman problems.

Xin Yao (F'03) is a Chair (Professor) of computer science and the Director of the Centre of Excellence for Research in Computational Intelligence and Applications, University of Birmingham, Birmingham, U.K. His major research interests include evolutionary computation and ensemble learning. He has more than 400 refereed publications in international journals and conferences.

Dr. Yao is a Distinguished Lecturer of the IEEE Computational Intelligence Society (CIS). His work won the 2001 IEEE Donald G. Fink Prize Paper Award, 2010 IEEE TRANSACTIONS ON EVOLUTIONARY COMPUTATION Outstanding Paper Award, 2010 BT Gordon Radley Award for Best Author of Innovation (Finalist), 2011 IEEE TRANSACTIONS ON NEURAL NETWORKS Outstanding Paper Award, and many other best paper awards at conferences. He won the prestigious Royal Society Wolfson Research Merit Award in 2012 and was selected to receive the 2013 IEEE CIS Evolutionary Computation Pioneer Award. He was the Editor-in-Chief of the IEEE TRANSACTIONS ON EVOLUTIONARY COMPUTATION from 2003 to 2008.