Contribution-based Cooperative Co-evolution for Non-separable Large-scale Problems with Overlapping Subcomponents

Ya-Hui Jia, Yi Mei, *Senior Member, IEEE,* and Mengjie Zhang, *Fellow, IEEE*

***Abstract*—Cooperative co-evolutionary algorithms have addressed many large-scale problems successfully, but the non-separable large-scale problems with overlapping subcomponents are still a serious difficulty that has not been conquered yet. First, the existence of shared variables makes the problem hard to be decomposed. Second, existing cooperative co-evolutionary frameworks usually cannot maintain the two crucial factors, high cooperation frequency and effective computing resource allocation, simultaneously when optimizing the overlapping subcomponents. Aiming at these two issues, this paper proposes a new contribution-based cooperative co-evolutionary algorithm to decompose and optimize the non-separable large-scale problems with overlapping subcomponents effectively and efficiently. 1) A contribution-based decomposition method is proposed to assign the shared variables. Among all the subcomponents containing a shared variable, the one that contributes the most to the whole problem will include the shared variable. 2) To achieve the two crucial factors at the same time, a new contribution-based optimization framework is designed to award the important subcomponents based on the round-robin structure. Experimental studies show that the proposed algorithm performs significantly better than the state-of-the-art algorithms due to the effective grouping structure generated by the proposed decomposition method and the fast optimizing speed provided by the new optimization framework.**

***Index Terms*—Cooperative Co-evolution, Evolution Strategy, Large-Scale Global Optimization, Overlapping Problem, Contribution-based Optimization**

I. Introduction

LARGE-SCALE global optimization (LSGO) is one of the most challenging problems in evolutionary computation (EC) [1], [2]. When the dimension of the problem grows, not only mathematical optimization methods but also traditional evolutionary algorithms (EAs) suffer from the curse of dimensionality [3]. To handle the complexity of large-scale problems and promote the usage of EC, scholars proposed many algorithms following two different directions. The first direction is considered from the algorithm perspective. Scholars tried to increase either the EAs' exploration abilities by designing new learning strategies [4], [5] or their exploitation abilities by incorporating local search algorithms [6], [7]. The second direction is considered from the problem perspective that the cooperative co-evolution (CC) scheme [8]–[10] is adopted. Based on the idea of divide-and-conquer, cooperative co-evolutionary algorithms (CCEAs) optimize a large-scale problem by decomposing it into small subcomponents and then solving these subcomponents collaboratively.

Ya-Hui Jia, Yi Mei, and Mengjie Zhang are with School of Engineering and Computer Science, Victoria University of Wellington, Wellington, New Zealand.

Numerous CCEAs have been proposed [1], [2], [11]–[21] and applied to real-world applications [22]–[28] since the first CCEA, namely CC genetic algorithm, was created by Potter and Jong [29]. A CCEA usually consists of a grouping method, a.k.a. decomposition or partition, to decompose problems and an EA to optimize the subcomponents.

Concerning the grouping method, at the beginning of CC development, fixed and random grouping methods were widely applied [17], [30]–[33]. However, their performance is not very good. Later on, several learning-based grouping methods were proposed based on different interdependence detecting techniques, such as CC with variable interaction learning [11], differential grouping (DG) [12], extended DG [34], global DG (GDG) [13], DG2 [14], and recursive DG (RDG) [15], [35], [36]. The essential idea of these methods is to gather interdependent variables into the same group and separate independent variables into different groups. These grouping methods have largely improved the performance of CC on LSGO problems. However, most CCEAs using learning-based grouping methods require the problem to be at least partially separable. When the problem is naturally non-separable, these algorithms usually cannot provide satisfactory results.

Non-separable large-scale problems with overlapping subcomponents (i.e. *overlapping problem*) widely exist in real-world applications [37]–[39]. In an overlapping problem, the interdependence among variables shows an overlapping topology that within a subcomponent, the variables strongly interact with each other. Subcomponents are connected by a few shared variables. GDG and DG2 cannot divide an overlapping problem into subcomponents since they use a depth-first search or breadth-first search method on the graph of variable interdependence to decompose problems. DG and RDG use a greedy decomposition scheme to assign the shared variables randomly to one of the subcomponents that interact with them. Although an overlapping problem can be decomposed into subcomponents in this way, the generated partition may not be the most effective one for the optimization process. Some parallel CC methods duplicate the shared variables in related subcomponents and select their values through competition, such as distributed CC [40], factored evolutionary algorithm (FEA) [41], and competitive-cooperative co-evolution [42], [43], However, their performance is usually not as competitive as the serial CC methods under limited computing resources [19], [40].

Regarding the optimization process, the traditional round-robin (RR) optimization structure of CC treats all subcomponents equally, but different subcomponents may have different levels of importance to the whole problem. To make the most of the computing resources, several new CC frameworks have been proposed. Omidvar *et al.* [44]–[46] proposed three contribution-based CC frameworks, CBCC1, CBCC2, and CBCC3. Inspired by CBCC1, Trunfio [47] proposed a CCEA called CCAOI. Yang *et al.* [48] and De Rainville *et al.* [49] proposed CCFR and CCEA-MAB respectively in which the subcomponent that has the largest contribution will be chosen to evolve. These new CC frameworks have a good performance on separable problems. However, on overlapping problems, they may be defeated by traditional CC [36] since the contribution of a subcomponent may be affected by other subcomponents due to the existence of shared variables. Zhang *et al.* [50] proposed a dynamic CC (DCC) to solve overlapping problems, but the contribution is accumulated for each variable which usually leads to poor performance [46], [48].

Reviewing the existing CCEAs, we can find two main drawbacks of them when they are used to solve large-scale overlapping problems:

- The existing decomposition methods cannot generate effective groupings on large-scale overlapping problems.
- The existing optimization methods or frameworks cannot optimize the overlapping subcomponents effectively.

Goal: To address these two drawbacks, in this paper we propose a new cooperative co-evolution algorithm called CBCCO to solve the large-scale overlapping problems. Corresponding to addressing the two drawbacks, the main contributions of CBCCO are shown in two aspects:

- From the perspective of decomposition of CBCCO, a contribution-based decomposition (CBD) method for overlapping problems is proposed. By designing a graphic decomposition method and setting a shared variable allocation stage, CBD assigns the shared variables to the subcomponents that have larger contributions.
- From the perspective of optimization of CBCCO, a new contribution-based optimization (CBO) framework is proposed for the overlapping problems. By designing a new awarding scheme that awards a group subcomponents each time, CBD can allocate the computing resources effectively and maintain a high cooperation frequency among the optimizers.

Through careful orchestration of CBD and CBO, the resultant algorithm CBCCO is compared with seven state-of-the-art EAs for LSGO problems, including the winners of CEC2018 and CEC2019 LSGO competitions. Moreover, further analyses about CBO and CBD are also made to gain insights into the reasons that CBCCO is better than the compared algorithms on the large-scale overlapping problems.

The rest of this paper is organized as follows. First, backgrounds including the definition of the overlapping problem and the related CC works are introduced in Section II. Afterward, Section III demonstrates the proposed algorithm CBCCO in detail. Experiments are set up and conducted in Section IV and Section V, respectively. Finally, the conclusions are drawn in Section VI.

II. Background

A. Large-scale Problem with Overlapping Subcomponents

The definition of the overlapping problem is built based on variable interdependence. According to DG [12], variable interdependence is defined by differential relationships.

Definition 1 (Explicit Interaction)**.** Given a n-dimensional function $f(\mathbf{x})$, $\mathbf{x} = (x_1, x_2, \ldots, x_n)$, $n \geq 2$, x_i and x_j $(1 \leq i < j \leq n)$ are said to *explicitly interact* ($x_i \leftrightarrow x_j$) with each other, if and only if:

$$\exists (a, b), \frac{\partial f}{\partial x_i \partial x_j}|_{x_i=a, x_j=b} \neq 0 \tag{1}$$

Definition 2 (Implicit Interaction)**.** Given a n-dimensional function $f(\mathbf{x})$, $\mathbf{x} = (x_1, x_2, \ldots, x_n)$, $n \geq 3$, denoting the set of all variables as Θ, x_i and x_j $(1 \leq i < j \leq n)$ are said to *implicitly interact* with each other, if and only if they do not explicitly interact with each other, and there is a subset $\Theta' \subset \Theta$, $\Theta' = \{x'_1, x'_2, \ldots, x'_{n'}\}$, $0 < n' \leq n-2$, $x_i \leftrightarrow x'_1 \leftrightarrow \cdots \leftrightarrow x'_{n'} \leftrightarrow x_j$.

Definition 3 (Independence)**.** Given a n-dimensional function $f(\mathbf{x})$, $\mathbf{x} = (x_1, x_2, \ldots, x_n)$, $n \geq 2$, x_i and x_j $(1 \leq i < j \leq n)$ are said to be *independent* of each other, if and only if they neither explicitly nor implicitly interact with each other.

According to these three definitions, we can roughly classify LSGO problems into two types: non-separable and separable.

Definition 4 (Non-separable)**.** A function $f(x_1, x_2, \ldots, x_n)$ is *non-separable* if $\forall i, j \in \{1, 2, \ldots, n\}$, x_i and x_j explicitly or implicitly interact with each other.

Definition 5 (Separable)**.** A function $f(x_1, x_2, \ldots, x_n)$ is *separable* if $\exists i, j \in \{1, 2, \ldots, n\}, i \neq j$, x_i and x_j are independent of each other.

Taking variable as node, explicit interaction as edge, the interdependence between every pair of variables can be represented by a graph. Four examples are shown in Fig. 1. Fig. 1(a) and Fig. 1(b) show two separable functions. Fig. 1(b) shows a totally separable function that can be seen as an extreme case of the partially separable function of Fig. 1(a). Fig. 1(c) and Fig. 1(d) show two non-separable functions. The function in Fig. 1(d) is totally non-separable. In this paper, we focus on the problems having similar structures to Fig. 1(c), which are called overlapping problems. Borrowing the concept in community detection problem [51], we can define such a structure as an overlapping community structure that can be divided into several communities with overlap, each community corresponding to a subcomponent. Explicit interactions are dense within a community. Between communities, there will be a few shared variables that provide bridges for implicit interactions. Generally, the number of non-shared variables is larger than the number of shared variables in a community. Otherwise, it cannot be recognized as a community since most of its variables also belong to other communities. From Fig. 1(c)

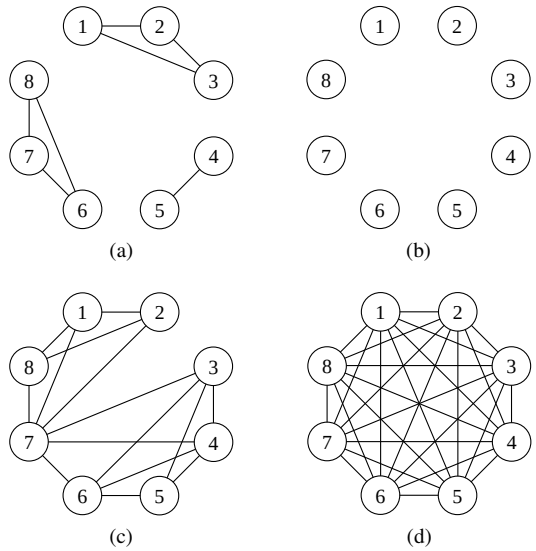

Fig. 1. Examples of different kinds of large-scale problems. (a) partially separable, (b) totally separable, (c) overlapping, (d) totally non-separable.

we can see that there are two communities $\{x_1, x_2, x_7, x_8\}$ and $\{x_3, x_4, x_5, x_6, x_7\}$ sharing a variable x_7. The overlapping community structure widely exists in real-world applications such as the water distribution network [37], [52] and the road network [53]. Thus, the overlapping problem optimization by CCEA has practical significance. In addition, without loss of generality, the overlapping problems considered in this paper are assumed to be minimization problems.

B. Cooperative Co-evolution

Usually, there are two procedures in a CCEA: decomposition and optimization.

1) Decomposition: The decomposition of a problem in a CCEA can be represented as follows:

$$\Theta = \Theta_1 \cup \Theta_2 \cup \cdots \cup \Theta_M \tag{2}$$

$$\text{s.t. } \forall i \in \{1, 2, \ldots, M\}, \Theta_i \neq \varnothing \tag{3}$$

$$\forall i, j \in \{1, 2, \ldots, M\} \wedge i \neq j, \Theta_i \cap \Theta_j = \varnothing \tag{4}$$

where M is the number of subcomponents. (3) shows that there is no empty subcomponent and (4) shows that a variable cannot be included in multiple subcomponents simultaneously.

DG [12] and its variants [13]–[15] are the state-of-the-art decomposition methods because of the high variable interdependence detecting accuracy they provide. Based on (1), DG checks the explicit interaction between two variables x_i and x_j by calculating the following three difference values:

$$\Delta_{x_i,x_j}[f] = f(x_i + \delta_i, x_j + \delta_j) - f(x_i, x_j) \tag{5}$$

$$\Delta_{x_i}[f] = f(x_i + \delta_i, x_j) - f(x_i, x_j) \tag{6}$$

$$\Delta_{x_j}[f] = f(x_i, x_j + \delta_j) - f(x_i, x_j). \tag{7}$$

x_i and x_j are thought to explicitly interact if:

$$|\Delta_{x_i,x_j}[f] - (\Delta_{x_i}[f] + \Delta_{x_j}[f])| > \epsilon \tag{8}$$

After obtaining the interdependence graph, DG decomposes the problem in a greedy way. Each time it selects a variable x_i and gathers the variables that interact with x_i into a group, and then removes them from the interdependence graph. RDG reduces the complexity of the interdependence detecting algorithm by using recursive detection, but the shared variables are still grouped randomly even in its latest version RDG3 [36]. GDG and DG2 use a depth-first search or a breadth-first search algorithm to decompose the problem. The difference between greedy decomposition and graphic search decomposition is that, in greedy decomposition, only explicit interaction is considered, while in graphic search decomposition, both explicit and implicit interactions are considered. Consequently, DG and RDG3 can decompose an overlapping problem into subcomponents, but GDG and DG2 cannot decompose the overlapping problems. Since the selection of the start variable in greedy decomposition is essentially a random process and shared variables are grouped with the earlier selected variable that interacts with them, actually, each shared variable is just randomly grouped into an interactive subcomponent. Although the greedy decomposition method can decompose an overlapping problem, randomly assigning the shared variables to interactive subcomponents may not generate the most effective grouping structure for optimization.

2) Optimization: In the optimization process, M optimizers $\{O_1, O_2, \ldots, O_M\}$ will be utilized corresponding to the M subcomponents. In a CCEA, each time only one subcomponent is optimized. The other subcomponents are held fixed. The best solution of the whole problem can be generated as the combination of the best sub-solutions:

$$\mathbf{x}^* = (\mathbf{x}_1^*, \mathbf{x}_2^*, \ldots, \mathbf{x}_M^*). \tag{9}$$

Conventionally, we call this solution "context vector". The fitness value of the ith sub-solution of the jth subcomponent is calculated as:

$$f(\mathbf{x}_{j,i}) = f(\mathbf{x}_1^*, \ldots, \mathbf{x}_{j,i}, \ldots, \mathbf{x}_M^*) \tag{10}$$

Each optimizer will update the context vector by its best sub-solution after optimization. The frequency of context vector updating can be taken as the cooperation/communication frequency of the optimizers.

Since each time only one one subcomponent is optimized, which subcomponent to be optimized next becomes an important issue. Different strategies lead to different CC frameworks as shown in Fig. 2. Traditional CC treats all subcomponents equally in a RR way.

In CBCC1 [44], besides the RR optimization process, the subcomponent that has the largest contribution in each generation, denoted as Θ_l, is selected to be awarded another generation. The contribution of a subcomponent accumulates in each generation:

$$\eta_i^{g+1} = \eta_i^g + (f^g - f^{g+1}), i \in \{1, 2, \ldots, M\}, \tag{11}$$

where g represents the number of generation.

CBCC2 [44] will give Θ_l enough generations instead of only one extra generation until the optimization cannot see improvement. Based on the analysis in [45], CBCC1 is more robust than CBCC2.

However, accumulating the contribution of each subcomponent may give the subcomponents that have larger historical

contributions awards rather than the subcomponents that have larger present contributions. Considering this drawback, two different frameworks, CBCC3 [46] and CCFR [48], are proposed. The structures of these two frameworks are similar as shown in Fig. 2(c).

In CBCC3, the contribution of a subcomponent is considered as the decline of the objective value in recent γ generations:

$$\eta_i^{g+\gamma} = f^g - f^{g+\gamma}, i \in \{1, 2, \ldots, M\}. \tag{12}$$

Information on historical contribution is abandoned in CBCC3. Regarding the subcomponent selection strategy, at first, an exploration phase is set that each subcomponent is given γ generations to be optimized. Then, Θ_l is chosen to be optimized until its contribution is no longer the largest. There will be a very small probability to go back to the exploration phase. Otherwise, a new Θ_l will be selected.

In CCFR, the contribution is calculated as:

$$\eta_i^{g+\gamma} = \frac{\eta_i^g + (f^g - f^{g+\gamma})}{2}, i \in \{1, 2, \ldots, M\}. \tag{13}$$

After the exploration phase, CCFR also continuously chooses Θ_l to optimize. When all subcomponents have the same contribution, it returns to the exploration phase. Besides, whenever an optimizer is stagnant, the contribution of its corresponding subcomponent will be set to 0. Due to the stagnancy detecting technique, usually, when all subcomponents have the same contribution, the optimizers are all stagnant.

DCC [50] has two stages: the random grouping stage and the dynamic grouping stage. Firstly, the random grouping method is applied to optimize the problem and the contribution of each variable accumulates for γ_1 generations. In the second stage, ranked by contribution, each time the top N_1 variables and the variables interacting with them are chosen to be optimized for γ_2 generations.

Comparing these CC frameworks, CBCC1 is the most moderate one that is built on the RR structure and each time only one subcomponent is awarded only one extra generation. However, the accumulation of contribution is a defect that cannot be ignored. CBCC3 and CCFR are more aggressive since each time γ generations are given to a subcomponent. They may be more suitable to the separable problems than non-separable problems because this strategy has abandoned RR [46]. As to the contribution calculation methods, abandoning the historical information in CBCC3 may also cause problems since most EAs are probabilistic algorithms that their performance in a short period is not very stable. DCC uses a different strategy that chooses variables rather than subcomponents during optimization. However, the accumulation of contribution may make its dynamic grouping method focus on a small group of variables.

III. Contribution-based Cooperative Co-evolution for Overlapping Problems

Considering the defects of the greedy decomposition and the aforementioned CC frameworks, we propose a new CCEA called CBCCO with a new contribution-based decomposition method and a new contribution-based optimization framework

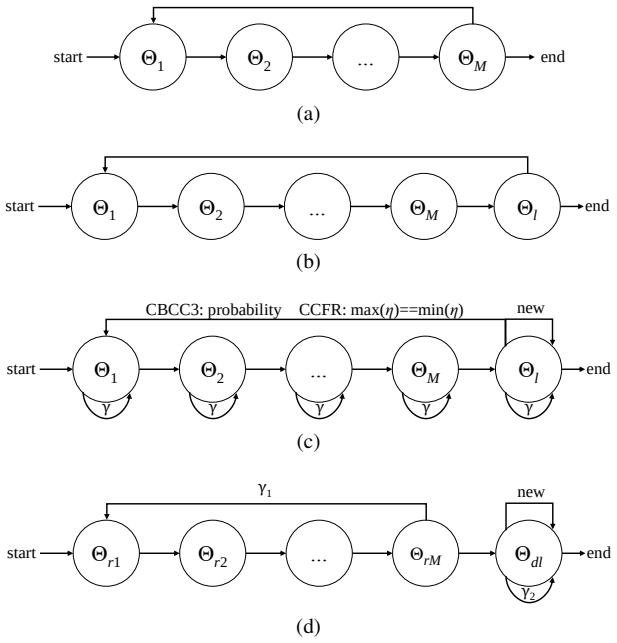

Fig. 2. Different optimization structures of different CCEAs. (a) traditional CC, (b) CBCC1, (c) CBCC3 and CCFR, (d) DCC.

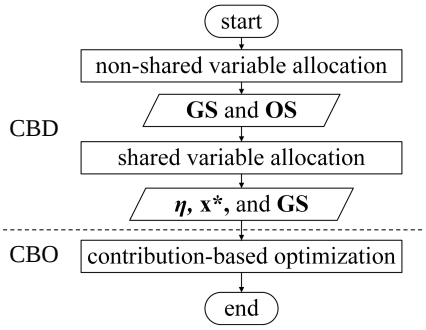

Fig. 3. Flowchart of CBCCO.

to specifically solve the large-scale overlapping problems. The overall process of CBCCO is shown in Fig. 3. It consists of two main stages: a CBD stage and a CBO stage. The CBD stage can be further divided into two sub-stages: non-shared variable allocation stage and shared variable allocation stage. In the non-shared variable allocation stage, the problem is first decomposed into two sets. $\mathbf{GS} = \{\Theta_1, \ldots, \Theta_M\}$ is the set of subcomponents consisting of non-shared variables, and $\mathbf{OS} = \{\Theta_{i \cap j} | 1 \leq i < j \leq M\}$ is the set of the shared variables. Here a graphic decomposition method is designed based on the decomposition method proposed in [40]. The shared variable allocation stage will optimize each subcomponent for N generations to set the initial contribution and to assign the shared variables to the existing subcomponents **GS**. Moreover, since the non-shared variables are already optimized in this stage, not only the context vector $\mathbf{x}^*$ and the contribution vector $\boldsymbol{\eta}$ but also the information of the optimizers will be saved, so that in the final optimization stage, the optimizers will not start from scratch. Finally, CBO is used to optimize the problem.

Algorithm 1 Non-shared Variable Allocation

Input: problem $f(\mathbf{x})$, variable set $\Theta = \{x_1, \ldots, x_n\}$, acceptable overlapping rate ζ.
Output: set of subcomponents consisting of non-shared variables **GS**, set of shared variables **OS**.

1: $\boldsymbol{G}$ = *interdependence-detect*$(f(\mathbf{x}), \Theta)$;
2: $M = 1$, $\bar{\Theta} = \Theta$;
3: **while** $\bar{\Theta}$ is not empty **do**
4: Find the variable with the smallest degree $\bar{x}$ in $\bar{\Theta}$;
5: Find $\{x_i | x_i \leftrightarrow \bar{x}\}$ by $\boldsymbol{G}$, add them into Θ_M with $\bar{x}$;
6: Remove Θ_M from $\bar{\Theta}$;
7: Insert Θ_M into **GS**;
8: $M = M + 1$;
9: **end while**
10: **for** $i = 2 \rightarrow M$ **do**
11: **for** $j = 1 \rightarrow i - 1$ **do**
12: Get the overlap $\Theta_{i \cap j}$;
13: **if** $|\Theta_{i\cap j}|/|\Theta_i| \geq \zeta$ or $|\Theta_{i\cap j}|/|\Theta_j| \geq \zeta$ **then**
14: Merge Θ_j into Θ_i;
15: Delete Θ_j from **GS**;
16: Delete overlaps related to Θ_j from **OS**;
17: $M = M - 1$, $j = 1$, $i = max(i - 1, 2)$;
18: **else**
19: Insert $\Theta_{i\cap j}$ into **OS**;
20: **end if**
21: **end for**
22: **end for**
23: **for all** $\Theta_{i\cap j}$ in **OS do**
24: $\Theta_i = \Theta_i - \Theta_{i\cap j}$, $\Theta_j = \Theta_j - \Theta_{i\cap j}$;
25: **end for**
26: **return GS** and **OS**;

Algorithm 2 Shared Variable Allocation

Input: problem $f(\mathbf{x})$, subcomponent set **GS**, shared variable set **OS**, number of test generation N.
Output: contribution vector $\boldsymbol{\eta}$, context vector $\mathbf{x}^*$, group set **GS**.

1: set M optimizers $\{O_1, \ldots, O_M\}$ corresponding to M subcomponents;
2: initialize the contribution vector $\boldsymbol{\eta} = (\eta_1, \ldots, \eta_M) = \mathbf{0}$;
3: initialize the context vector $\mathbf{x}^* = \mathbf{0}$;
4: **for** $i = 1 \rightarrow M$ **do**
5: $f^o = f(\mathbf{x}^*)$;
6: **for** $j = 1 \rightarrow N$ **do**
7: optimize $f(\mathbf{x}_i)$ by O_i;
8: update $\mathbf{x}^*$;
9: **end for**
10: $\eta_i = f^o - f(\mathbf{x}^*)$;
11: **end for**
12: **for all** $\Theta_{i\cap j}$ in **OS do**
13: **if** $\eta_i > \eta_j$ **then**
14: $\Theta_i = \Theta_i \cup \Theta_{i\cap j}$;
15: **else**
16: $\Theta_j = \Theta_j \cup \Theta_{i\cap j}$;
17: **end if**
18: **end for**
19: store the information of the optimizers;
20: **return** $(\boldsymbol{\eta}, \mathbf{x}^*, \mathbf{GS})$;

A. Contribution-based Decomposition

1) Non-shared Variable Allocation: Based on our previous work [40], a graphic decomposition method that generates subcomponent set of non-shared variables **GS** and overlap set of shared variables **OS** is devised.

This stage consists of four parts as shown in Algorithm 1. The first part is to detect the interdependence between variables and get the interdependence graph $\boldsymbol{G}$ (line 1). Due to the high detection accuracy, DG2 [14] is recommended to be used here. The second procedure generates subcomponents that each shared variable appears in every subcomponent that interacts with it (lines 2-9). Then, by checking every pair of subcomponents, we can get all shared variables in the third part (lines 10-22). If the shared variables occupy over ζ in a subcomponent, this pair of subcomponents will be merged (lines 13-17). In the fourth part, all shared variables are eliminated from the subcomponents (lines 23-25). Finally, we get the subcomponent set **GS** containing all non-shared variables and the overlap set **OS** containing all shared variables.

The acceptable overlapping rate ζ is used to measure how many shared variables between two subcomponents are acceptable. If there are excessive shared variables between two subcomponents, they will be merged. The rate can be adjusted according to the real situation of the problem. According to [40], it is set to 0.3 in Algorithm 1.

2) Shared Variable Allocation: In the shared variable allocation stage, we assign the shared variables by computing the contribution of each subcomponent in **GS**. After the non-shared variable allocation process, **GS** only contains non-shared variables. Since the non-shared variables are in the majority in a subcomponent, evaluating the contribution of a subcomponent to the whole problem by only non-shared variables on a large probability can still reflect the contribution difference between subcomponents correctly. The pseudo-code is shown in Algorithm 2.

In this stage, each subcomponent is optimized for N generations. Since shared variables are not considered, directly optimizing each subcomponent for N generations is identical to using the RR optimization method (lines 4-11). The contribution of each subcomponent accumulates in this stage (line 10):

$$\begin{aligned} \eta_i &= f^0 - f^N \\ &= (f^0 - f^1) + (f^1 - f^2) + \cdots + (f^{N-1} - f^N). \end{aligned} \tag{14}$$

Then, for each overlap, it is assigned to the subcomponent which has a larger contribution and interacts with it (lines 12-18). Finally, the information of optimizers is saved for the reset procedure in the optimization process and the algorithm returns the results (lines 19-20).

B. Contribution-based Optimization

Experimental studies in [45] show that when the subcomponents are not independent of each other, high cooperation

Algorithm 3 Contribution-based Optimization

Input: problem $f(\mathbf{x})$, contribution vector $\boldsymbol{\eta}$, context vector $\mathbf{x}^*$, group set **GS**, stop criterion *SC*.
Output: context vector $\mathbf{x}^*$.

1: reset optimizers $\{O_1, \ldots, O_M\}$;
2: **while** stop criterion is not met **do**
3: **for** $i = 1 \to M$ **do**
4: $f^o = f(\mathbf{x}^*)$;
5: optimize $f(\mathbf{x}_i)$ by O_i;
6: update $\mathbf{x}^*$;
7: $\eta_i = (\eta_i + (f^o - f(\mathbf{x}^*))/2.0$;
8: **end for**
9: $\eta_{max} = \max(\boldsymbol{\eta})$;
10: $awardList = \varnothing$;
11: **for** $i = 1 \to M$ **do**
12: **if** $\eta_{max}/\eta_i < 2$ **then**
13: add i into $awardList$;
14: **end if**
15: **end for**
16: **if** $|awardList| == M$ **then**
17: clear $awardList$;
18: **end if**
19: **for all** i in $awardList$ **do**
20: $f^o = f(\mathbf{x}^*)$;
21: optimize $f(\mathbf{x}_i)$ by O_i;
22: update $\mathbf{x}^*$;
23: $\eta_i = (\eta_i + (f^o - f(\mathbf{x}^*))/2.0$;
24: **end for**
25: **end while**
26: **return** $\mathbf{x}^*$;

frequency is helpful to maintain a steady optimizing speed of the whole algorithm. Since every subcomponent is explicitly or implicitly related to other subcomponents in an overlapping problem, high cooperation frequency is thus crucial to solving overlapping problems. The RR structure has the highest cooperation frequency as every subcomponent will update the context vector in every generation. Therefore, in CBO, the RR structure is reserved as the basis. Regarding the awarding scheme, the basic idea is to accelerate the optimizing speed of the important subcomponents so that they can reach the same contribution level with the less important subcomponents. When most optimizers are on the same level, use RR to optimize them. Based on this intuition, CBO is designed as Algorithm 3.

First, the optimizers should be reset to add the shared variables into optimization using the information stored in the shared variable allocation stage (line 1). For example, if a population-based EA like differential evolution is used, the individuals evolved in the previous stage can be inherited and new dimensions can be inserted into each individual to involve the shared variables. If a distribution-based EA like evolution strategy is used, the distribution models evolved in the previous stage can be inherited and new dimensions can be added to the models to involve the shared variables.

Then, the main loop (lines 2-25) of the algorithm can be divided into three parts. The first part is the RR optimization

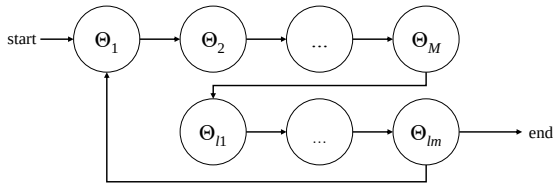

Fig. 4. Structure of CBO. $\{\Theta_{l1}, \ldots, \Theta_{lm}\}$ represents the awarded subcomponents in the first echelon.

(lines 3-8). All subcomponents are optimized in a RR manner, and the contribution of each subcomponent is updated:

$$\eta_i^{g+1} = \frac{\eta_i^g + (f^g - f^{g+1})}{2}. \tag{15}$$

According to (15), the historical contribution halves in each generation which follows an exponential speed. To make the historical contribution play a role in the optimization and meanwhile accommodate to the fast decay speed, a big base value should be established. That is the reason why the contribution accumulates in the shared variable allocation stage rather than updating by (15).

The second part is to choose the subcomponents that should be awarded (lines 9-18). First, the maximum contribution among subcomponents η_{max} is obtained (line 9). Then, for each subcomponent, if its contribution is bigger than half η_{max}, we put it into the award list (lines 12-14). We call these subcomponents "the first echelon". The third part is to award the first echelon (lines 19-24). Each subcomponent in the echelon gets an extra generation to evolve, and the contribution is also updated. The structure of CBO is shown in Fig. 4.

In Algorithm 3, half η_{max} is set as the standard to select the first echelon. The reason is due to the contribution calculating method and the awarding method. The historical contribution halves in each generation according to (15). Considering the worst situation that the subcomponent whose contribution is the largest does not improve the objective value at all in the awarded extra generation, its contribution will be directly updated to half its previous value. Thus, if the contribution of a subcomponent before awarding is just smaller than half η_{max}, definitely it does not belong to the first echelon.

C. Discussions

Here, we discuss the rationality of CBCCO. First, the difference between CBO and some other decomposition methods is shown and why CBD works is explained conceptually. Then, the applicability of CBO to overlapping problems is compared with the other contribution-based CC optimization frameworks.

1) Contribution-based Decomposition: First, CBO is compared with some other decomposition methods that generate fixed grouping results on how they allocate the shared variables when dealing with an overlapping problem. From Table I, we can see that the way CBD allocates shared variables is different from the others. Each shared variable is assigned to a related subcomponent that has larger contribution rather than random allocation or duplication.

TABLE I
DIFFERENCE BETWEEN DIFFERENT DECOMPOSITION METHODS WHEN HANDLING SHARED VARIABLES.

Methods	Shared Variable Allocation
DG2, GDG	will not divide the problem
DG, RDG3	randomly assign each shared variable to a related subcomponent
FEA	duplicate shared variables in related subcomponents and select their values through competition
CBD	assign each shared variable to a related subcomponent that has larger contribution

Except CBD, RDG3 is the most recent one that is used to decompose overlapping problems. Here, we use an example to show why assigning shared variables to the subcomponents that have larger contributions is better than random allocation. Consider an overlapping function with two subcomponents:

$$f(\mathbf{x}) = f_1(\mathbf{x}_1, \mathbf{y}) + f_2(\mathbf{x}_2, \mathbf{y}) \tag{16}$$

where $\mathbf{y}$ is the overlap between f_1 and f_2. Assume f_1 outweighs f_2, and due to the nature of the large-scale overlapping problem, we find no optimizer that can solve the problem as a whole, but only optimizers that can solve f_1 and f_2 separately. After the gth generation, we have context vector $\mathbf{x}^* = (\mathbf{x}_1^*, \mathbf{y}^*, \mathbf{x}_2^*)$. Then, in the ($g$+1)th generation, we consider two decomposition strategies: a) assigning $\mathbf{y}$ to the first subcomponent; b) assigning $\mathbf{y}$ to the second subcomponent.

If $\mathbf{y}$ is assigned to the first subcomponent, we can get two groups: $\{\mathbf{x}_1, \mathbf{y}\}$ and $\{\mathbf{x}_2\}$. In CC, the optimization process of the first group is:

$$\min f(\mathbf{x}|_{\mathbf{x}_2=\mathbf{x}_2^*}) = \min[f_1(\mathbf{x}_1, \mathbf{y}) + f_2(\mathbf{x}_2 = \mathbf{x}_2^*, \mathbf{y})] \tag{17}$$

Suppose we get $(\mathbf{x}_1 = \mathbf{x}_1^{**}, \mathbf{y} = \mathbf{y}^{**})$ after optimizing the first group. Then for the second group, the optimization process becomes:

$$\min f(\mathbf{x}|_{\mathbf{x}_1=\mathbf{x}_1^{**}, \mathbf{y}=\mathbf{y}^{**}}) = \min f_2(\mathbf{x}_2, \mathbf{y} = \mathbf{y}^{**}). \tag{18}$$

In this assigning strategy, when the first group is optimized, if f_1 outweighs f_2 greatly, $f_2(\mathbf{x}_2 = \mathbf{x}_2^*, \mathbf{y})$ is only a small disturbance to the optimization of $f_1(\mathbf{x}_1, \mathbf{y})$. If f_1 only outweighs f_2 slightly, optimizing the first group is actually optimizing the whole f_1 and a part of f_2 since $f_2(\mathbf{x}_2 = \mathbf{x}_2^*, \mathbf{y})$ is not greatly smaller than $f_1(\mathbf{x}_1, \mathbf{y})$. When optimizing the second group, the non-shared variables of f_2, i.e. $\mathbf{x}_2$, is optimized.

If $\mathbf{y}$ is assigned to the second subcomponent, we can get two groups: $\{\mathbf{x}_1\}$ and $\{\mathbf{x}_2, \mathbf{y}\}$. When the first group is optimized, we have:

$$\min f(\mathbf{x}|_{\mathbf{x}_2=\mathbf{x}_2^*, \mathbf{y}=\mathbf{y}^*}) = \min f_1(\mathbf{x}_1, \mathbf{y} = \mathbf{y}^*). \tag{19}$$

Suppose we get $\mathbf{x}_1 = \mathbf{x}_1^{**}$ after optimizing the first group. Then, for the second group, the optimization process is:

$$\min f(\mathbf{x}|_{\mathbf{x}_1=\mathbf{x}_1^{**}}) = \min[f_1(\mathbf{x}_1 = \mathbf{x}_1^{**}, \mathbf{y}) + f_2(\mathbf{x}_2, \mathbf{y})]. \tag{20}$$

This time, when optimizing the first group, the non-shared part of f_1, i.e. $\mathbf{x}_1$, is optimized. When optimizing the second group, if f_1 outweighs f_2 greatly, it is actually optimizing $f_1(\mathbf{x}_1 = \mathbf{x}_1^{**}, \mathbf{y})$ rather than f_2. Only if f_1 outweighs f_2 slightly, f_2 can be optimized. Thus, assigning the shared variables to an unimportant subcomponent can be considered as a decomposition of the important subcomponent, and the unimportant subcomponent itself in the worst situation will be ignored. If we already know that the applied optimizer can deal with the low-dimensional subcomponents efficiently as we have assumed before, it will be better to assign the shared variables to the important subcomponents that have larger contributions.

2) Contribution-based Optimization: The differences between CBO and several existing contribution-based CC optimization frameworks are explained to show why CBO is more suitable to overlapping problems than the others. Table II compares these frameworks in two aspects, contribution calculation method and computing resource allocation method.

Considering the contribution calculation method, CBO considers both the historical contributions and the current contributions of the subcomponents, and sets a fast attenuation speed to the historical contribution to balance the influence between these two parts. In terms of the computing resource allocation method, CBCC3, CCFR, and DCC have abandoned RR. CBCC1 and CBCC2 award only one subcomponent each time. Different from them, CBO reserves RR and awards multiple subcomponents in each generation. This difference can also be shown from the comparison of the structures shown in Fig. 2 and Fig. 4. Due to the utilization of the new contribution calculation method and the new resource allocation method, CBO can both quickly synchronize all subcomponents' contributions to a same level by awarding important subcomponents more computing resources and maintain a high cooperation frequency among optimizers which is critical to the optimization of overlapping problems. Due the page limit, the theoretical and empirical analyses of cooperation frequencies of different CC optimization methods are demonstrated in the supplementary material.

IV. EXPERIMENTAL SETUP

Before comparing CBCCO with other algorithms, in this section, the benchmark functions are first introduced. Then, the setup about CBCCO is given, and the only parameter of CBCCO, i.e. the generation number of shared variable allocation stage N, is selected.

A. Benchmark

CEC2013 LSGO benchmark functions [54] are widely-used to test the performance of EAs proposed for LSGO problems. Among the 15 functions, there are two overlapping functions, f_{13} and f_{14}. f_{13} is a conforming overlapping function in which the optimum values of shared variables are the same in different subcomponents. f_{14} is a conflicting overlapping function in which the optimum values of shared variables are different in different subcomponents. Thus, for a conforming overlapping function, the optimization of one subcomponent may benefit the other subcomponents that overlap with it. However, for a conflicting overlapping function, the optimization of one subcomponent may interfere with the optimization of other subcomponents. Both functions use the schwefel function as the elementary function. To specifically study the overlapping

TABLE II
DIFFERENCE BETWEEN DIFFERENT CONTRIBUTION-BASED COOPERATIVE CO-EVOLUTION OPTIMIZATION FRAMEWORKS.

Methods	Contribution Calculation	Resource Allocation
CBCC1	accumulation every generation	RR, rewarding one subcomponent one generation
CBCC2	accumulation every generation	RR, rewarding one subcomponent until it converges
CBCC3	objective improvement in recent γ generations	select one subcomponent evolving γ generations
CCFR	accumulation with attenuation every γ generations	selecct one subcomponent evolving γ generations
DCC	accumulation every γ_2 generation	select one subcomponent evolving γ_2 generations
CBO	accumulation with attenuation every generation	RR, rewarding multiple subcomponents one generation

TABLE III
LARGE-SCALE OVERLAPPING BENCHMARK CONSISTS OF 12 FUNCTIONS.

Func.	Character	Group Size	Range	Elementary Function
f_1	Conforming	$50\times5+25\times10+100\times5$	$x\in[-100,100]$	$f_{schwefel}(\mathbf{x})=\sum_{i=1}^{D}(\sum_{j=1}^{i}x_i)^2$
f_2	Conflicting	$50\times5+25\times10+100\times5$		
f_3	Conforming	50×20		
f_4	Conflicting	50×20		
f_5	Conforming	$50\times5+25\times10+100\times5$	$x\in[-100,100]$	$f_{elliptic}(\mathbf{x})=\sum_{i=1}^{D}10^{6\frac{i-1}{D-1}}x_i^2$
f_6	Conflicting	$50\times5+25\times10+100\times5$		
f_7	Conforming	50×20		
f_8	Conflicting	50×20		
f_9	Conforming	$50\times5+25\times10+100\times5$	$x\in[-5,5]$	$f_{rastrigin}(\mathbf{x})=\sum_{i=1}^{D}[x_i^2-10\cos(2\pi x_i)+10]$
f_{10}	Conflicting	$50\times5+25\times10+100\times5$		
f_{11}	Conforming	50×20		
f_{12}	Conflicting	50×20		

problem, there are three different benchmarks proposed [36], [40], [50]. Compared with the benchmark functions proposed in [36], [50], the benchmark of [40] is more comprehensive since not only different elementary functions but functions with both uniform and non-uniform subcomponent sizes are considered [1]. Thus, in the following experiments, this benchmark is used.

From Table III, we can see that besides the schwefel function, another two elementary functions: elliptic function and rastrigin function, are also used. Schwefel function and elliptic function are two unimodal functions; rastrigin function is a multimodal function. Thus, f_9-f_{12} are much harder to be optimized than the former 8 functions. Meanwhile, both uniform and non-uniform subcomponent sizes are considered in the benchmark. For functions with uniform subcomponent size, there are totally 20 subcomponents with 50 decision variables each. The imbalanced importance is realized by setting different weight factors to different subcomponents. For functions with non-uniform subcomponent sizes, there are 5 subcomponents with 50 decision variables each, 10 subcomponents with 25 variables each, and 5 subcomponents with 100 decision variables each. The imbalanced importance is realized by both different weight factors and different subcomponent sizes. For each function, there are 95 shared variables. Thus, the dimension of one function is $1000-95=905$. f_{13} and f_{14} of CEC2013 LSGO benchmark are exactly f_1 and f_2 in Table I, respectively.

B. Parameter Setting

In CBCCO, covariance matrix adaptation evolution strategy (CMA-ES) is chosen as the optimizer due to its excellent capability to solve low-dimensional continuous optimization problems [13], [36], [48], [55], [56]. The parameters of CMA-ES are set by following the commonly used settings [56].

Besides the parameters of CMA-ES, there is only one parameter in CBCCO, the number of generation N in the shared variable allocation stage. Theoretically, a larger N will bring more stable assignment. However, if N is too large, it will affect the optimization stage because of the limited computing resources. Before selecting a suitable value for N, an upper bound is set to 100 empirically just as same as γ in CBCC3 and CCFR. Setting the interval as 10, 10 candidates values $\{10, 20, \ldots, 100\}$ are selected within (0, 100]. Each value is tested on each function for 30 times. Before choosing a suitable value, a sufficiently large number (i.e. 200) is tested 30 times on each function to get a stable assignment as the standard. According to our experiment, the assignment is already stable when N is equal to 200 that on each function the 30 times assignments are all identical. The stability of an N value is defined as the ratio of the number of the assignments that are identical to the standard. Experimental results are shown in Fig. 5.

As we expected, generally a larger N brings a higher stability on all functions. For the first four functions, it is easy to get a stable assignment that even $N=20$ is sufficient to achieve 100% stability. For f_5-f_8, it is harder to get a stable assignment, but eventually when $N=100$, CBD gets stable assignments on these four functions. For the last four functions, $N=80$ is large enough. Generally, we can see that $N=100$ is sufficiently large for CBD to get stable assignments on all functions but does not lead to high computational cost. Thus, in the following experiment, N is set to 100. Since the essence of setting this parameter is to figure out the contribution difference between overlapping subcomponents, based on the results, we suggest that for a

[1]The benchmark is available on https://github.com/Flyki/Large-Scale-Overlapping-Optimization

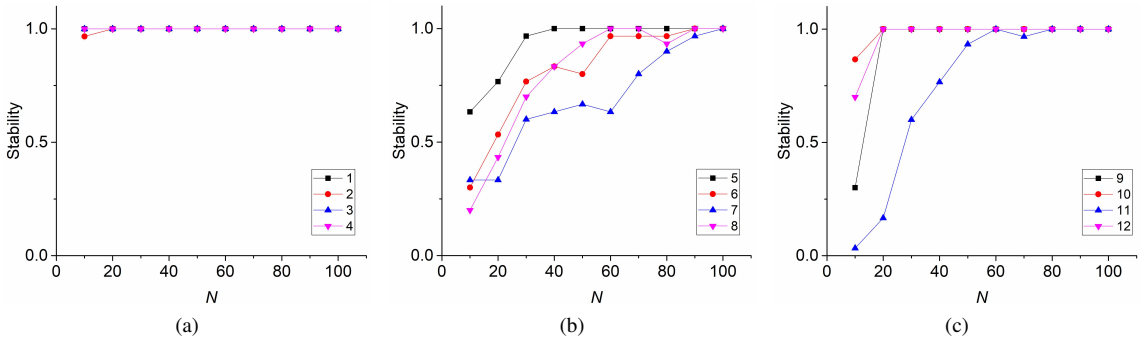

Fig. 5. Stability of shared variable assignment under four different N values. (a) f_1-f_4, (b) f_5-f_8, (c)f_9-f_{12}.

new problem, $N = 2 \cdot n/M$ would be a good starting point.

V. Comparisons and Analyses

First of all, CBCCO is compared with the state-of-the-art EAs that were proposed for LSGO problems, including some algorithms especially proposed for large-scale overlapping problems. Then, further analyses about CBD and CBO are made to find the reason why CBCCO is effective in optimizing overlapping problems.

Since the interdependence detecting method is not a focus in this paper, and DG variants can already detect the variable interdependence in a very high accuracy (DG2 [14] can already achieve 100% accuracy on all LSGO benchmark functions.), we adopt the experimental design method used in [45], [46], [57] that the ideal interdependence graph is utilized for all compared algorithms to avoid the influence caused by using different interdependence detecting methods. 30 independent runs are executed for each algorithm to get the statistic information including mean objective value and standard deviation. Meanwhile, the two-sided Wilcoxon rank-sum test is conducted between the proposed algorithm and the compared algorithms to observe whether CBCCO is significantly better than them. The significance level is set to 0.05. When there are multiple comparisons, the Bonferroni correction is used to adjust the significance level. To make fair comparisons, the maximum number of fitness evaluations (FEs) is used as the stopping criterion for all the compared algorithms. According to the suggestion of the benchmark [54], it is set to 3×10^6.

A. Comparisons with State-of-the-art Algorithms

To check the performance of CBCCO, we compared it with the start-of-the-art EAs, including competitive swarm optimizer (CSO) [5], level-based learning swarm optimizer (LLSO) [4], success-history-based adaptive differential evolution with iterative local search (SHADEILS) [6], RDG3 [36], DCC [50], FEA [41], and cooperative co-evolution with adaptive subcomponents (CCAS) [33]. CSO, LLSO, and SHADEILS are not CC-based algorithms. They focus on improving either the exploration or exploitation ability of the algorithm. RDG3, DCC, FEA, and CCAS are CCEAs. RDG3 and DCC are especially proposed for large-scale overlapping problems. FEA requires overlaps between different subcomponents, which is in line with the problem structure. CCAS

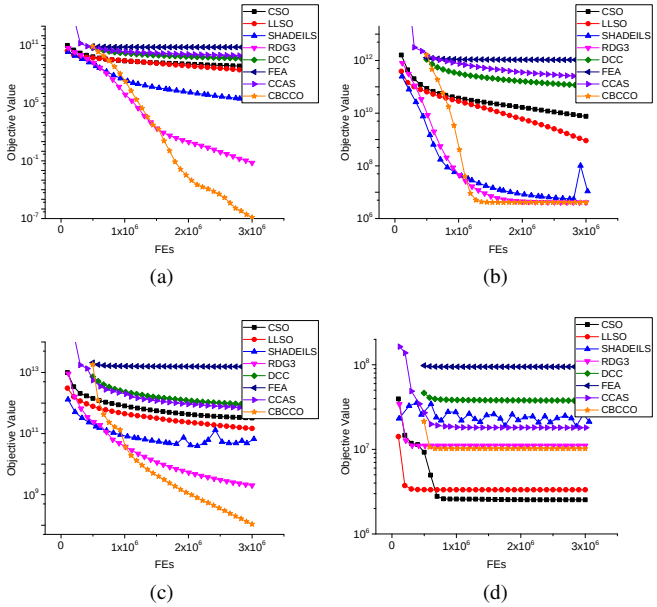

Fig. 6. Convergence curves of CSO, LLSO, SHADEILS, RDG3, DCC, and CBCCO. (a) f_3, (b) f_4, (c) f_5, (d) f_9.

is a representative CCEA using random grouping method. In addition, SHADEILS and RDG3 the winners of the CEC2018 and CEC2019 LSGO competitions, respectively.

Since there is no interaction detection process in CSO, LLSO, SHADEILS, and CCAS, all 3×10^6 FEs can be used for optimization. For RDG3, according to [36], it uses roughly 1.65×10^4 FEs to decompose a benchmark function. Thus, it can use 2.9835×10^6 FEs during optimization. For DCC, FEA, and CBCCO, since they both require a whole interdependence graph, according to DG2 [14], 4.1×10^5 FEs will be used in the decomposition process. Thus, there are 2.59×10^6 FEs can be used during optimization. Experimental results are shown in Table IV. Moreover, the convergence curve of each algorithm is drawn. Due to the page limitation, Fig. 6 only shows four representative cases on f_3, f_4, f_5, and f_9. The full set of figures can be found in the supplementary material.

The performance of CBCCO is checked by comparing with different kinds of EAs.

1) Compared with CSO and LLSO, CBCCO outperforms them on the first 8 functions and is inferior to them on

TABLE IV
COMPARISON OF OBJECTIVE VALUES ON 12 BENCHMARK FUNCTIONS BETWEEN CSO, LLSO, SHADEILS, RDG3, DCC, FEA, CCAS, AND CBCCO

Func	Stats	CSO	LLSO	SHADEILS	RDG3	DCC	FEA	CCAS	CBCCO
f_1	mean	7.50E+08(+)	2.83E+08(+)	9.31E+05(+)	5.85E+04(+)	2.38E+09(+)	3.09E+10(+)	5.20E+09(+)	1.21E+03
	std.	3.28E+08	1.44E+08	7.80E+05	8.85E+04	1.42E+09	1.11E+10	3.93E+09	1.21E+03
f_2	mean	4.86E+09(+)	1.14E+08(+)	7.98E+06(+)	4.42E+06(=)	2.79E+10(+)	5.12E+11(+)	2.49E+11(+)	4.43E+06
	std.	4.96E+09	6.01E+07	9.05E+05	6.88E+04	2.43E+10	1.70E+11	1.36E+11	5.09E+04
f_3	mean	6.40E+08(+)	2.29E+08(+)	2.22E+05(+)	6.26E-02(+)	4.16E+09(+)	6.50E+10(+)	8.28E+09(+)	1.32E-07
	std.	2.30E+08	9.32E+07	2.03E+05	1.42E-01	1.37E+09	1.62E+10	6.22E+09	2.34E-07
f_4	mean	7.58E+09(+)	9.11E+08(+)	5.23E+06(+)	4.11E+06(=)	1.09E+11(+)	1.05E+12(+)	2.41E+11(+)	4.09E+06
	std.	3.63E+09	1.13E+09	3.74E+05	8.30E+04	5.12E+10	4.09E+11	1.68E+11	8.54E+04
f_5	mean	3.25E+11(+)	1.48E+11(+)	2.82E+10(+)	2.01E+09(+)	8.88E+11(+)	1.56E+13(+)	6.74E+11(+)	1.09E+08
	std.	2.61E+10	2.41E+10	6.57E+09	5.06E+08	3.01E+11	5.65E+12	2.68E+11	6.48E+07
f_6	mean	3.10E+12(+)	1.59E+12(+)	1.78E+11(+)	9.36E+08(+)	1.20E+13(+)	1.77E+14(+)	6.20E+12(+)	8.28E+08
	std.	4.00E+11	2.72E+11	3.62E+10	1.53E+08	4.13E+12	6.15E+13	2.14E+12	1.56E+07
f_7	mean	6.91E+11(+)	4.21E+11(+)	5.68E+10(+)	5.70E+07(+)	2.11E+12(+)	2.43E+13(+)	1.11E+12(+)	1.42E+07
	std.	6.66E+10	5.68E+10	1.37E+10	6.14E+07	4.88E+11	8.31E+12	3.36E+11	3.54E+07
f_8	mean	1.06E+13(+)	5.56E+12(+)	4.45E+11(+)	2.90E+11(+)	4.96E+13(+)	3.38E+14(+)	1.85E+13(+)	2.83E+08
	std.	1.17E+12	6.27E+11	1.12E+11	1.14E+11	1.65E+13	7.77E+13	5.76E+12	6.69E+07
f_9	mean	2.53E+06(-)	3.33E+06(-)	1.16E+07(+)	1.11E+07(=)	3.78E+07(+)	9.47E+07(+)	1.81E+07(+)	1.02E+07
	std.	3.58E+05	4.99E+05	1.19E+06	1.86E+06	7.92E+06	1.24E+07	4.89E+06	1.78E+06
f_{10}	mean	5.97E+07(-)	6.53E+07(-)	1.40E+08(-)	1.75E+08(=)	8.59E+08(+)	1.47E+09(+)	2.67E+08(+)	1.87E+08
	std.	6.49E+06	9.02E+06	1.80E+07	3.04E+07	2.16E+08	2.92E+08	6.63E+07	4.20E+07
f_{11}	mean	5.36E+06(-)	6.93E+06(-)	1.62E+07(-)	1.87E+07(=)	7.13E+07(+)	1.79E+08(+)	2.47E+07(+)	1.94E+07
	std.	5.93E+05	7.68E+05	2.10E+06	2.81E+06	1.59E+07	2.92E+07	3.74E+06	2.44E+06
f_{12}	mean	9.29E+07(-)	1.14E+08(-)	2.64E+08(-)	4.17E+08(+)	1.35E+09(+)	3.47E+09(+)	4.19E+08(+)	3.30E+08
	std.	1.38E+07	1.84E+07	3.38E+07	7.07E+07	3.13E+08	4.38E+08	5.04E+07	6.26E+07
W/T/L		8/0/4	8/0/4	9/0/3	7/5/0	12/0/0	12/0/0	12/0/0	

[1] (+), (-), and (=) represent that CBCCO is significantly better than, significantly worse than, and equivalent to the compared algorithm according to the Wilcoxon rank-sum test.
[2] "W/T/L" represents on how many functions CBCCO wins/ties/loses the comparison.

the last 4 functions. CSO and LLSO have stronger exploration abilities than others, but their exploitation ability is weak. Thus, they do not have very fast optimizing speed on f_1-f_8, but on multimodal functions f_9-f_{12}, the great exploration ability helped them to achieve better objective values than CBCCO.

2) Compared with SHADEILS, CBCCO is significantly better on f_1-f_9, but is worse on f_{10}-f_{12}. The exploitation ability SHADEILS is better than CSO, but is not good enough to surpass CBCCO. Thus, on f_1-f_9, it is better than CSO but is worse than CBCCO. On f_{10}-f_{12}, the situation is just the opposite. Generally, the performance of SHADEILS is in the middle position between CSO and CBCCO.
3) Compared with RDG3, CBCCO outperforms it on 7 functions and has similar performance to it on 5 functions. RDG3 decomposes the overlapping problems into subcomponents and also uses the canonical CMA-ES to optimize them like CBCCO. Since the canonical CMA-ES is very good at optimizing low-dimensional unimodal functions, RDG3 obtains better results than CSO, LLSO, and SHADEILS on f_1-f_8. However, since the grouping structure generated by the greedy decomposition method cannot support contribution-based CC frameworks, RDG3 is less effective than CBCCO. Both RDG3 and CBCCO perform poorly on f_9-f_{12}, because of the applied optimizer, CMA-ES. An empirical study in [58] shows that the canonical CMA-ES usually cannot handle multimodal functions.
4) Compared with DCC, FEA, and CCAS, CBCCO is significantly better on all functions. This fact implies that random grouping or parallel CC may not very effective. Although DCC also uses the interaction relationship between variables in its optimization, the accumulated contribution still takes control of the algorithm that causes low cooperation frequency.
5) From Fig. 6, we can see that on f_3, f_4, and f_5, CBCCO has the fastest convergence speed. On the conforming functions f_3 and f_5, it maintains this speed from the beginning to the end. On the conflicting function f_4, although CBCCO, RDG3, and SHADEILS finally get similar results, the convergence speed of CBCCO is still the fastest. However, when the applied optimizer cannot handle the subcomponents effectively such as on f_9, all algorithms converge very early including CBCCO. In summary, the function figures show that the convergence speed of CBCCO is fast.

Generally, the results show that CBCCO is among the best of the compare algorithms. On the unimodal functions, its performance is significantly better than the compared algorithms. On the multimodal functions, its performance is not that satisfactory. However, this does not necessarily mean that CBCCO only works well on unimodal or some simple problems, since other EAs can also be applied in CBCCO. Many CCEAs have verified that using different optimizers in CC can bring different effects and which optimizer to use is problem-dependent [12], [48]. Actually, simply enlarging the population size of CMA-ES and setting a restart procedure can improve the performance of CBCCO on f_9-f_{12} greatly. The original population size is set as $\lambda_i = 4 + \lfloor 3 \ln n_i \rfloor$ in canonical

TABLE V
OBJECTIVE VALUES OBTAINED BY CBCCO2 ON MULTIMODAL FUNCTIONS WITH ADJUSTED CMA-ES.

Func.	Stats	CSO	LLSO	CBCCO2
f_9	mean	2.53E+06(-)	3.33E+06(+)	2.80E+06
	std.	3.58E+05	4.99E+05	4.03E+05
f_{10}	mean	5.97E+07(+)	6.53E+07(+)	4.74E+07
	std.	6.49E+06	9.02E+06	8.70E+06
f_{11}	mean	5.36E+06(+)	6.93E+06(+)	4.71E+06
	std.	5.93E+05	7.68E+05	5.47E+05
f_{12}	mean	9.29E+07(+)	1.14E+08(+)	8.43E+07
	std.	1.38E+07	1.84E+07	9.61E+06

CMA-ES, where λ_i is the population size of the ith optimizer and n_i is the number of the ith subcomponent's dimension. Here, we enlarge it to $\lambda_i = 2 \times n_i$. The restart condition is set to $\sigma < \max(x) \cdot 10^{-3}$. The new version of CBCCO is denoted as CBCCO2. CSO and LLSO are taken as the control group since they achieved the best results on these four functions. The performance of CBCCO2 on f_9-f_{12} is shown in Table V.

It is clear that when the optimizer is adjusted, the performance of CBCCO has the ability to surpass the other algorithms. This time, only on f_9, CBCCO is beaten by CSO. On the other three functions, it is significantly better than CSO and LLSO. Overall, the results of Table IV and Table V show that CBCCO can significantly outperform the other state-of-the-art EAs on large-scale overlapping problems.

B. Analysis of CBD

To gain a deep sight of CBCCO, we analyse CBD and CBO separately to check their advantages. First, to verify the effectiveness of CBD, it is compared with the greedy decomposition method based on the ground-truth that all subcomponents have been correctly recognized. Second, the condition under which CBD is effective is investigated.

1) Comparison between CBD and Greedy Decomposition: In the greedy decomposition method, shared variables are randomly assigned to an interactive subcomponent. To get a fixed decomposition results of the greedy decomposition method, for each shared variable subset $\Theta_{i \cap j}, i < j$, it is assigned to Θ_i. Besides, the reverse version of CBD, denoted as CBD-R, in which each shared variable is assigned to the subcomponent that has smaller contribution is also tested as a control group. A hypothesise is made that among the three decompositions, CBD is better than the greedy decomposition, and CBD-R is the worst. Moreover to avoid the influence of the optimization structure, RR is used. Experimental results are displayed in Table VI.

According to the results, our hypothesis is basically verified that CBD outperforms the greedy method and CBD-R on most functions, and CBD-R gets the worst performance. Specifically, according to the results of Wilcoxon rank-sum tests, CBD is significantly better than CBD-R on all 12 functions, and is significantly better than the greedy method on 6 functions $\{f_3, f_5\text{-}f_9, f_{12}\}$. There are no functions on which CBD is worse than the greedy method.

Moreover, the convergence curves are drawn. Fig. 7 shows four representative cases on f_3, f_4, f_5, and f_9. The full set

TABLE VI
COMPARISON OF OBJECTIVE VALUES BETWEEN CBD-R, GREEDY DECOMPOSITION, AND CBD.

Func.	Stats	CBD-R	Greedy	CBD
f_1	mean	2.41E+06(+)	2.88E+05(=)	3.55E+05
	std.	2.47E+06	3.04E+05	5.95E+05
f_2	mean	4.52E+06(+)	4.42E+06(=)	4.40E+06
	std.	1.83E+05	4.70E+04	4.51E+04
f_3	mean	3.66E+02(+)	8.99E+00(+)	2.03E-08
	std.	3.40E+02	8.18E+00	7.45E-08
f_4	mean	4.18E+06(+)	4.10E+06(=)	4.10E+06
	std.	1.50E+05	8.64E+04	8.67E+04
f_5	mean	1.92E+10(+)	5.65E+09(+)	7.32E+08
	std.	5.45E+10	3.38E+09	3.34E+08
f_6	mean	2.26E+11(+)	2.11E+11(+)	9.22E+08
	std.	5.30E+10	9.59E+10	2.04E+08
f_7	mean	1.09E+10(+)	1.61E+10(+)	7.64E+07
	std.	3.90E+09	8.02E+09	1.59E+08
f_8	mean	1.63E+11(+)	9.67E+10(+)	4.54E+09
	std.	5.83E+10	6.65E+10	1.07E+10
f_9	mean	1.68E+07(+)	1.36E+07(+)	9.83E+06
	std.	3.86E+06	2.39E+06	1.81E+06
f_{10}	mean	3.12E+08(+)	2.16E+08(=)	1.89E+08
	std.	9.58E+07	5.89E+07	3.91E+07
f_{11}	mean	2.26E+07(+)	2.07E+07(=)	1.92E+07
	std.	3.33E+06	3.05E+06	2.55E+06
f_{12}	mean	4.30E+08(+)	3.64E+08(=)	3.30E+08
	std.	8.56E+07	6.56E+07	7.02E+07
W/T/L		12/0/0	6/6/0	

of figures can be found in the supplementary material. First, Fig. 7(a) and Fig. 7(c) have verified our analysis that if an optimizer can solve the subcomponents efficiently, it will be less efficient to assign the shared variables to the unimportant subcomponents than to the important subcomponents. Fig. 7(b) shows that the three decomposition methods converge to the same level on this conflicting function, but we can still find that CBD is faster than the other two. Fig. 7(d) shows that they are all trapped into local optima. However, even under this circumstance, CBD still has slight advantage. Overall, the results have verified the analysis made in Section III.C and shown that CBD is effective.

2) Condition of the Effectiveness of CBD: According to the analysis and the experimental results, we have known that when there is contribution difference between subcomponents, CBD will be effective, but how large the contribution difference should be to exhibit the advantage of CBD is unknown. Here, we choose f_3 to test this problem under different levels of contribution difference. As shown in Table I, f_3 has uniform subcomponent size. Therefore, we can adjust each subcomponent's importance by setting different weight factors. Another reason to choose f_3 is that the uncertainties like premature and local optima can be avoided as shown in Fig. 7(a) that even when FEs $= 3 \times 10^6$, the three methods are still maintaining good optimizing speeds.

To set different contribution differences, first, the weight factors of all subcomponents with odd indices are set to $w_o = 1$. Then, the weight factors of all subcomponents with even indices are set to 6 different values $w_e \in \{1, 1.25, 1.5, 2, 3, 4\}$ each time. When $w_e = 1$, all subcomponents have the same importance to the problem. When $w_e = 4$, the contribution of even subcomponents are 4 times the contribution of odd

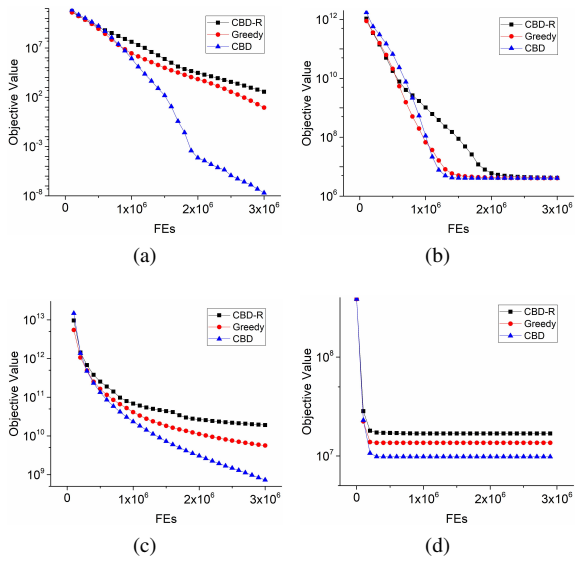

Fig. 7. Convergence curves of CBD-R, Greedy, and CBD. (a) f_3, (b) f_4, (c) f_5, (d) f_9.

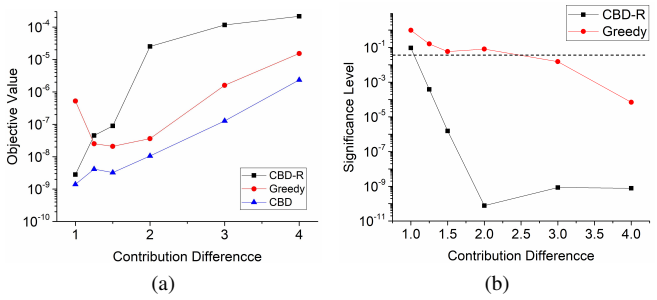

Fig. 8. Objective values and significance levels obtained by different decomposition methods under different levels of contribution difference. The dashed line in (b) represents the significance level 0.025 with Bonferroni correction. When the value is underneath the dashed line, CBD is significantly better than the other two methods.

subcomponents. The objective values are shown in Fig. 8(a). The results of the Wilcoxon rank-sum tests are shown in Fig. 8(b) to check when CBD becomes significantly better than the other two methods.

Fig. 8 basically matches our expectation that when the contribution difference increases, the advantage of CBD strengthens. From Fig. 8(a), we can find that the objective values obtained by the three decomposition methods all grow along with the contribution difference. Generally, CBD-R is the worst, the greed decomposition method is in the middle, and CBD is the best. Seeing Fig. 8(b), we can find that when $w_e \approx 2$, CBD starts to be significantly better than the greed decomposition method according to the significance level 0.025. When $w_e \geq 3$, CBD can certainly show its advantage. Actually, if the interdependence graph has been obtained, CBD is always a good choice worth considering since it at worst can turn into a greedy decomposition method when all subcomponents have the identical importance.

TABLE VII
COMPARISON OF OBJECTIVE VALUES BETWEEN RR, CBCC1, CCFR, AND CBO

Func.	Stats	RR	CBCC1	CCFR	CBO
f_1	mean	3.55E+05(+)	1.18E+03(+)	1.29E+09(+)	4.98E+01
	std.	5.95E+05	1.24E+03	3.16E+09	6.08E+01
f_2	mean	4.40E+06(-)	4.40E+06(-)	4.76E+10(+)	4.43E+06
	std.	4.51E+04	4.11E+04	7.79E+10	5.09E+04
f_3	mean	2.03E-08(+)	5.00E-06(+)	1.32E+09(+)	9.69E-10
	std.	7.45E-08	2.68E-05	3.72E+09	1.70E-09
f_4	mean	4.10E+06(=)	4.09E+06(=)	9.49E+09(+)	4.09E+06
	std.	8.67E+04	7.30E+04	3.47E+10	8.54E+04
f_5	mean	7.32E+08(+)	9.79E+08(+)	1.21E+12(+)	4.09E+07
	std.	3.34E+08	4.86E+08	3.32E+12	3.47E+07
f_6	mean	9.22E+08(+)	9.70E+08(+)	5.12E+13(+)	8.20E+08
	std.	2.04E+08	4.08E+08	9.89E+13	1.11E+07
f_7	mean	7.64E+07(+)	4.04E+07(+)	6.70E+11(+)	2.76E+06
	std.	1.59E+08	7.88E+07	1.70E+12	7.76E+06
f_8	mean	4.54E+09(+)	3.17E+08(+)	1.02E+14(+)	2.54E+08
	std.	1.07E+10	1.59E+08	2.17E+14	4.77E+07
f_9	mean	9.83E+06(=)	1.05E+07(=)	1.03E+07(=)	1.02E+07
	std.	1.81E+06	1.34E+06	1.51E+06	1.78E+06
f_{10}	mean	1.89E+08(=)	1.87E+08(=)	1.65E+08(-)	1.87E+08
	std.	3.91E+07	3.26E+07	4.16E+07	4.20E+07
f_{11}	mean	1.92E+07(=)	2.02E+07(=)	1.82E+07(=)	1.94E+07
	std.	2.55E+06	2.66E+06	2.76E+06	2.44E+06
f_{12}	mean	3.30E+08(=)	3.42E+08(=)	3.35E+08(=)	3.30E+08
	std.	7.02E+07	4.58E+07	4.27E+07	6.26E+07
W/T/L		6/5/1	6/5/1	8/3/1	

C. Analysis of CBO

In this subsection, CBO is compared with two other contribution-based CC frameworks, CBCC1 and CCFR. CBCC1 also uses the RR structure as their basis, and according to [45] it is more recommended than CBCC2. CCFR is a state-of-the-art framework that has a different structure from CBCC1 and CBCC2. CBD and CMA-ES are also used in CBCC1 and CCFR. Since the stagnancy detection method of CCFR is designed for EAs that maintain a fixed population [48], the stagnancy detection method for CMA-ES is set that if for a specific number of successive generations there is no improvement of the objective value, the optimizer is considered to be stagnant. Following the original setting, this number is set to the subcomponent size. RR is tested as the baseline. The results are shown in Table VII. Convergence curves are also drawn that can be found in the supplementary material Fig. C.

According to the mean values, CBO outperforms the other algorithms on 7 functions $\{f_1, f_3, f_5\text{-}f_8, f_{12}\}$. The results of Wilcoxon rank-sum tests tell us that only on f_2 and f_{10}, CBO is beaten by RR, CBCC1, and CCFR, respectively. On other functions, CBO either outperforms the other algorithms or has a similar performance to them. Generally, the results show that CBO is very efficient in accelerating the optimization speed.

Regarding the adaptive capacities of different CC frameworks to overlapping problems, generally, the results in Table VII show that CBO, CBCC1, and RR are better than CCFR. This fact indicates that the RR structure is indeed necessary when solving overlapping problems. CBO and CBCC1 seldom hinder the optimization compared with RR. Even when CBO gets worse results than RR on f_2, according to the mean objective values, the gap is not big. However, the results show

that on f_1-f_8, CCFR performs much worse than RR, CBCC1, and CBO. Thus, a verdict can be made that the contribution-based CC frameworks that conserve RR structure is more adaptive to the overlapping problems. Like the conclusion drawn in [45], if there is no domain knowledge available, the conservative method is safer than the aggressive method.

Although the empirical study has shown that high cooperation frequency is one of the key factors leading to the success of CBO on overlapping problems, so far, there has not been any quantitative measurement of cooperation frequency to systematically compare among these models. To fill this gap, a dynamic quantitative measurement and corresponding analysis are given in the supplementary material.

VI. Conclusions and Future Work

The goal of this paper was to solve the difficulties of optimizing large-scale overlapping problems by CCEA. This goal has been successfully achieved by proposing a new CCEA called CBCCO. Specifically, the two novel components of CBCCO, the decomposition method CBD and the optimization framework CBO, have shown great effectiveness and efficiency when decomposing and optimizing the problems respectively. Compared with the greedy decomposition method, using CBD can bring significant improvement when there is a considerable difference among the importance of subcomponents. CBO also shows the optimizing speed advantage compared with other CC optimization frameworks. Combining these two components together, the effectiveness and efficiency of CBCCO is further strengthened. The empirical comparisons with the state-of-the-art algorithms, including CSO, LLSO, SHADEILS, RDG3, DCC, FEA, and CCAS have showed that CBCCO can be much better than them for solving large-scale overlapping problems.

Although CBCCO has achieved very promising results in this paper, it still has potential to be better. The existing decomposition methods including CBD still encounter the interference problem that the optimization of one subcomponent would largely affect the other subcomponents. Since whether the problem is conforming or conflicting is usually unknown to us, developing new decomposition methods that can reduce the interference is a promising direction for future research. Catering to different decomposition methods, there should be more studies on the contribution-based optimization framework as well to further accelerate the optimizing speed.

References

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

[2] X. Ma, X. Li, Q. Zhang, K. Tang, Z. Liang, W. Xie, and Z. Zhu, "A survey on cooperative co-evolutionary algorithms," *IEEE Trans. Evol. Comput.*, vol. 23, no. 3, pp. 421–441, 2018.

[3] K. Tang, X. Yao, P. N. Suganthan, C. MacNish, Y.-P. Chen, C.-M. Chen, and Z. Yang, "Benchmark functions for the cec'2008 special session and competition on large scale global optimization," *Nature Inspired Comput. Appl. Lab., USTC, China*, vol. 24, 2007.

[4] Q. Yang, W.-N. Chen, J. Da Deng, Y. Li, T. Gu, and J. Zhang, "A level-based learning swarm optimizer for large-scale optimization," *IEEE Trans. Evol. Comput.*, vol. 22, no. 4, pp. 578–594, 2017.

[5] R. Cheng and Y. Jin, "A competitive swarm optimizer for large scale optimization," *IEEE Trans. Cybern.*, vol. 45, no. 2, pp. 191–204, 2014.

[6] D. Molina, A. LaTorre, and F. Herrera, "Shade with iterative local search for large-scale global optimization," in *Proc. CEC*. IEEE, 2018, pp. 1–8.

[7] D. Molina and F. Herrera, "Iterative hybridization of de with local search for the cec'2015 special session on large scale global optimization," in *Proc. CEC*. IEEE, 2015, pp. 1974–1978.

[8] M. A. Potter and K. A. D. Jong, "Cooperative coevolution: An architecture for evolving coadapted subcomponents," *Evol. Comput.*, vol. 8, no. 1, pp. 1–29, 2000.

[9] M. A. Potter, "The design and analysis of a computational model of cooperative coevolution," Ph.D. dissertation, Citeseer, 1997.

[10] K. A. De Jong and M. A. Potter, "Evolving complex structures via cooperative coevolution." in *Proc. Evol. Program.*, 1995, pp. 307–317.

[11] W. Chen, T. Weise, Z. Yang, and K. Tang, "Large-scale global optimization using cooperative coevolution with variable interaction learning," in *Proc. PPSN*. Springer, 2010, pp. 300–309.

[12] M. N. Omidvar, X. Li, Y. Mei, and X. Yao, "Cooperative co-evolution with differential grouping for large scale optimization," *IEEE Trans. Evol. Comput.*, vol. 18, no. 3, pp. 378–393, 2013.

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

[14] M. N. Omidvar, M. Yang, Y. Mei, X. Li, and X. Yao, "Dg2: A faster and more accurate differential grouping for large-scale black-box optimization," *IEEE Trans. Evol. Comput.*, vol. 21, no. 6, pp. 929–942, 2017.

[15] Y. Sun, M. Kirley, and S. K. Halgamuge, "A recursive decomposition method for large scale continuous optimization," *IEEE Trans. Evol. Comput.*, vol. 22, no. 5, pp. 647–661, 2017.

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

[17] M. N. Omidvar, X. Li, Z. Yang, and X. Yao, "Cooperative co-evolution for large scale optimization through more frequent random grouping," in *Proc. CEC*. IEEE, 2010, pp. 1–8.

[18] X. Li and X. Yao, "Cooperatively coevolving particle swarms for large scale optimization," *IEEE Trans. Evol. Comput.*, vol. 16, no. 2, pp. 210–224, 2011.

[19] Y.-H. Jia, W.-N. Chen, T. Gu, H. Zhang, H.-Q. Yuan, S. Kwong, and J. Zhang, "Distributed cooperative co-evolution with adaptive computing resource allocation for large scale optimization," *IEEE Trans. Evol. Comput.*, vol. 23, no. 2, pp. 188–202, 2018.

[20] X. Peng, Y. Jin, and H. Wang, "Multimodal optimization enhanced cooperative coevolution for large-scale optimization," *IEEE Trans. Cybern.*, vol. 49, no. 9, pp. 3507–3520, 2018.

[21] X. Ma, F. Liu, Y. Qi, X. Wang, L. Li, L. Jiao, M. Yin, and M. Gong, "A multiobjective evolutionary algorithm based on decision variable analyses for multiobjective optimization problems with large-scale variables," *IEEE Trans. Evol. Comput.*, vol. 20, no. 2, pp. 275–298, 2015.

[22] L. Sun, L. Lin, M. Gen, and H. Li, "A hybrid cooperative coevolution algorithm for fuzzy flexible job shop scheduling," *IEEE Trans. Fuzzy Syst.*, vol. 27, no. 5, pp. 1008–1022, 2019.

[23] A.-M. Farahmand, M. N. Ahmadabadi, C. Lucas, and B. N. Araabi, "Interaction of culture-based learning and cooperative co-evolution and its application to automatic behavior-based system design," *IEEE Trans. Evol. Comput.*, vol. 14, no. 1, pp. 23–57, 2009.

[24] N. R. Sabar, J. Abawajy, and J. Yearwood, "Heterogeneous cooperative co-evolution memetic differential evolution algorithm for big data optimization problems," *IEEE Trans. Evol. Comput.*, vol. 21, no. 2, pp. 315–327, 2016.

[25] Y. Mei, X. Li, and X. Yao, "Cooperative coevolution with route distance grouping for large-scale capacitated arc routing problems," *IEEE Trans. Evol. Comput.*, vol. 18, no. 3, pp. 435–449, 2013.

[26] C. Liang, C. Chung, K. Wong, and X. Duan, "Parallel optimal reactive power flow based on cooperative co-evolutionary differential evolution and power system decomposition," *IEEE Trans. Power Syst.*, vol. 22, no. 1, pp. 249–257, 2007.

[27] N. K. Adhikari, S. J. Louis, and S. Liu, "Multi-objective cooperative co-evolution of micro for rts games," in *Proc. CEC*. IEEE, 2019, pp. 482–489.

[28] L. Sun, L. Lin, H. Li, and M. Gen, "Cooperative co-evolution algorithm with an mrf-based decomposition strategy for stochastic flexible job shop scheduling," *Mathematics*, vol. 7, no. 4, p. 318, 2019.

[29] M. A. Potter and K. A. De Jong, "A cooperative coevolutionary approach to function optimization," in *Proc. PPSN*. Springer, 1994, pp. 249–257.

[30] 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, 2004.
[31] Z. Yang, K. Tang, and X. Yao, "Large scale evolutionary optimization using cooperative coevolution," *Inf. Sci.*, vol. 178, no. 15, pp. 2985–2999, 2008.
[32] ——, "Differential evolution for high-dimensional function optimization," in *Proc. CEC*. IEEE, 2007, pp. 3523–3530.
[33] G. A. Trunfio, P. Topa, and J. Was, "A new algorithm for adapting the configuration of subcomponents in large-scale optimization with cooperative coevolution," *Inf. Sci.*, vol. 372, pp. 773–795, 2016.
[34] Y. Sun, M. Kirley, and S. K. Halgamuge, "Extended differential grouping for large scale global optimization with direct and indirect variable interactions," in *Proc. GECCO*. ACM, 2015, pp. 313–320.
[35] Y. Sun, M. N. Omidvar, M. Kirley, and X. Li, "Adaptive threshold parameter estimation with recursive differential grouping for problem decomposition," in *Proc. GECCO*. ACM, 2018, pp. 889–896.
[36] Y. Sun, X. Li, A. Ernst, and M. N. Omidvar, "Decomposition for large-scale optimization problems with overlapping components," in *Proc. CEC*. IEEE, 2019, pp. 326–333.
[37] W. Zhao, T. H. Beach, and Y. Rezgui, "Optimization of potable water distribution and wastewater collection networks: A systematic review and future research directions," *IEEE Trans. Syst., Man, Cybern., Syst.*, vol. 46, no. 5, pp. 659–681, 2015.
[38] L. Tong, X. Zhou, and H. J. Miller, "Transportation network design for maximizing space–time accessibility," *Transport. Res. B-Meth.*, vol. 81, pp. 555–576, 2015.
[39] R. Moeini and M. Afshar, "Arc based ant colony optimization algorithm for optimal design of gravitational sewer networks," *Ain Shams Eng. J.*, vol. 8, no. 2, pp. 207–223, 2017.
[40] Y.-H. Jia, Y.-R. Zhou, Y. Lin, W.-J. Yu, Y. Gao, and L. Lu, "A distributed cooperative co-evolutionary cma evolution strategy for global optimization of large-scale overlapping problems," *IEEE Access*, vol. 7, pp. 19 821–19 834, 2019.
[41] S. Strasser, J. Sheppard, N. Fortier, and R. Goodman, "Factored evolutionary algorithms," *IEEE Trans. Evol. Comput.*, vol. 21, no. 2, pp. 281–293, 2016.
[42] C. K. Goh, K. C. Tan, D. Liu, and S. C. Chiam, "A competitive and cooperative co-evolutionary approach to multi-objective particle swarm optimization algorithm design," *Eur. J Oper. Res.*, vol. 202, no. 1, pp. 42–54, 2010.
[43] C.-K. Goh and K. C. Tan, "A competitive-cooperative coevolutionary paradigm for dynamic multiobjective optimization," *IEEE Trans. Evol. Comput.*, vol. 13, no. 1, pp. 103–127, 2008.
[44] M. N. Omidvar, X. Li, and X. Yao, "Smart use of computational resources based on contribution for cooperative co-evolutionary algorithms," in *Proc. GECCO*. ACM, 2011, pp. 1115–1122.
[45] B. Kazimipour, M. N. Omidvar, X. Li, and A. K. Qin, "A sensitivity analysis of contribution-based cooperative co-evolutionary algorithms," in *Proc. CEC*. IEEE, 2015, pp. 417–424.
[46] M. N. Omidvar, B. Kazimipour, X. Li, and X. Yao, "Cbcc3—a contribution-based cooperative co-evolutionary algorithm with improved exploration/exploitation balance," in *Proc. CEC*. IEEE, 2016, pp. 3541–3548.
[47] G. A. Trunfio, "Adaptation in cooperative coevolutionary optimization," in *Adaptation and Hybridization in Computational Intelligence*. Springer, 2015, pp. 91–109.
[48] M. Yang, M. N. Omidvar, C. Li, X. Li, Z. Cai, B. Kazimipour, and X. Yao, "Efficient resource allocation in cooperative co-evolution for large-scale global optimization," *IEEE Trans. Evol. Comput.*, vol. 21, no. 4, pp. 493–505, 2016.
[49] F.-M. De Rainville, M. Sebag, C. Gagné, M. Schoenauer, and D. Laurendeau, "Sustainable cooperative coevolution with a multi-armed bandit," in *Proc. GECCO*, 2013, pp. 1517–1524.
[50] X. Y. Zhang, Y. J. Gong, Y. Lin, J. Zhang, S. Kwong, and J. Zhang, "Dynamic cooperative coevolution for large scale optimization," *IEEE Trans. Evol. Comput.*, 2019.
[51] X. Wen, W.-N. Chen, Y. Lin, T. Gu, H. Zhang, Y. Li, Y. Yin, and J. Zhang, "A maximal clique based multiobjective evolutionary algorithm for overlapping community detection," *IEEE Trans. Evol. Comput.*, vol. 21, no. 3, pp. 363–377, 2016.
[52] W.-N. Chen, Y.-H. Jia, F. Zhao, X.-N. Luo, X.-D. Jia, and J. Zhang, "A cooperative co-evolutionary approach to large-scale multisource water distribution network optimization," *IEEE Trans. Evol. Comput.*, 2019.
[53] E. Walraven, M. T. Spaan, and B. Bakker, "Traffic flow optimization: A reinforcement learning approach," *Eng. Appl. Artif. Intel.*, vol. 52, pp. 203–212, 2016.
[54] X. Li, K. Tang, M. N. Omidvar, Z. Yang, K. Qin, and H. China, "Benchmark functions for the cec 2013 special session and competition on large-scale global optimization," *gene*, vol. 7, no. 33, p. 8, 2013.
[55] N. Hansen and A. Ostermeier, "Adapting arbitrary normal mutation distributions in evolution strategies: The covariance matrix adaptation," in *Proc. IEEE intl. conf. evol. comput.* IEEE, 1996, pp. 312–317.
[56] N. Hansen, "The cma evolution strategy: a comparing review," in *Towards a new evol. comput.* Springer, 2006, pp. 75–102.
[57] W. Chen and K. Tang, "Impact of problem decomposition on cooperative coevolution," in *Proc. CEC*. IEEE, 2013, pp. 733–740.
[58] N. Hansen and S. Kern, "Evaluating the cma evolution strategy on multimodal test functions," in *Proc. PPSN*. Springer, 2004, pp. 282–291.