Decomposition for Large-scale Optimization Problems with Overlapping Components

Yuan Sun
School of Science
RMIT University
Melbourne, Australia
yuan.sun@rmit.edu.au

Xiaodong Li
School of Science
RMIT University
Melbourne, Australia
xiaodong.li@rmit.edu.au

Andreas Ernst
School of Mathematical Sciences
Monash University
Clayton, Australia
andreas.ernst@monash.edu

Mohammad Nabi Omidvar
School of Computer Science
The University of Birmingham
Birmingham, United Kingdom
m.omidvar@cs.bham.ac.uk

***Abstract*—In this paper we use a divide-and-conquer approach to tackle large-scale optimization problems with overlapping components. Decomposition for an overlapping problem is challenging as its components depend on one another. The existing decomposition methods typically assign all the linked decision variables into one group, thus cannot reduce the original problem size. To address this issue we modify the *Recursive Differential Grouping* (RDG) method to decompose overlapping problems, by breaking the linkage at variables shared by multiple components. To evaluate the efficacy of our method, we extend two existing overlapping benchmark problems considering various level of overlap. Experimental results show that our method can greatly improve the search ability of an optimization algorithm via divide-and-conquer, and outperforms RDG, random decomposition as well as other state-of-the-art methods. We test an adaptive allocation of computational resources to components based on a measure of contribution, but this does not facilitate solving overlapping problems. We further evaluate our method using the CEC'2013 benchmark problems and show that our method is very competitive when equipped with a component optimizer.**

***Index Terms*—Cooperative co-evolution, large-scale continuous optimization, overlapping problem, variable interaction, problem decomposition.**

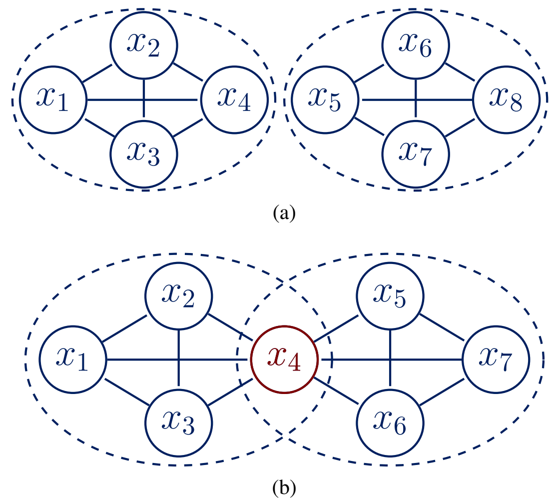


Fig. 1: An example of the problems with (a) separable and (b) overlapping components. In (a) two components are completely separable from each other; while in (b) two components share a decision variable x_4.

I. Introduction

Many real-world optimization problems are composed of several sub-problems that possibly depend on each other [1]–[4]. Exploiting this module structure can greatly facilitate the problem solving process [5], [6]. This is particularly useful when tackling a large-scale problem, where the search space is very large and the computational resource is limited. The variable interaction structure can be used to decompose a large-scale problem into sub-problems that are solved individually. This divide-and-conquer approach is known as cooperative co-evolution (CC) and has achieved many successes when used to solve large-scale optimization problems [6]–[10].

If a problem consists of separable components, e.g., Fig. 1a, it is possible to generate an optimal solution by solving each component independently. However in many real-world applications, e.g., optimizing the wine supply chain [3] and the transportation of water tanks [4], the problem components usually interact with each other. What is the best strategy to decompose these problems? In this paper we try to answer this question using a particular type of problems in which the components share some decision variables, e.g., Fig. 1b.

In the CC literature numerous methods have been proposed to decompose a black-box optimization problem. However they are typically ineffective when tackling problems with overlapping components. The random grouping (RG) [11] and delta grouping [12] do not explicitly consider the underlying interaction structure of decision variables. The intelligent decomposition methods including extended differential grouping (XDG) [13], global differential grouping (GDG) [14], recursive differential grouping (RDG) [10] and differential grouping 2 (DG2) [15] assign all the linked variables (i.e., variables that interact both directly and indirectly [16]) into one group, thus in many cases can not reduce the problem size.

The above methods partition the decision variables of a given problem into mutually exclusive subsets. Differently there are other methods that assign some decision variables to more than one subset [17]–[22]. For example the overlapped CC [21] allocates "influential" variables to multiple groups and the statistical variable interdependence learning [18] identities a linkage group for each decision variable based on non-monotonicity detection. However it is non-trivial to design a good communication strategy for the components that share some common variables. Thus we will focus on the mutually exclusive decomposition in this paper.

In the genetic algorithm literature, there have been some works that construct overlapping building blocks to handle overlapping problems [5], [23]–[26]. For example in [23], the linkage groups are identified by non-monotonicity detection and loosely linked variables are removed from the linkage groups. In [24], [25] a Bayesian network is built based on promising candidate solutions that implicitly captures the problem structure. In [5] the pairwise mutual information between decision variables is calculated based on promising candidate solutions, and a clustering algorithm is then used to group variables into overlapping linkage groups. However these methods are computationally expensive and are not directly applicable to CC.

In this paper we tackle large-scale overlapping problems using CC. To this end, we modify the RDG method to effectively decompose an overlapping problem. RDG is chosen due to its decomposition efficiency; it can decompose an n-dimensional problem using $O\big(n\log(n)\big)$ function evaluations (FEs). The idea of our modification is to break the linkage at variables shared by multiple components (see Fig. 3 in Section III for an example). In our modification we recursively identify the decision variables that directly interact with a given variable under consideration and place them into a group. If the current group size is less than a given threshold, we will further examine the interaction between the current group and remaining variables. Otherwise we treat the current group as a component without further identifying any indirect interaction. The threshold is used to control the group size.

To evaluate the efficacy of our proposed method, we extend two overlapping problems in the CEC'2013 benchmark suite [27], by varying the number of shared variables between components. Experimental results show that our method significantly improves over RDG, and outperforms other methods when embedded into a CC framework to solve the extended overlapping problems. We then try to boost the performance of CC via adaptively allocating computational resources to components based on their contributions to the overall fitness improvement. However the solution quality generated by a typical contribution-based CC model [28] is worse than that of standard CC. We infer the reason may partially attribute to the dependence of components in overlapping problems. Finally we show that our method equipped with covariance matrix adaptation – evolutionary strategy (CMA-ES) [29] produces overall the best solution quality when compared against 9 other state-of-the-arts on the CEC'2013 benchmark suite.

The remainder of this paper is organized as follows. In the next section, we describe CC and briefly review the related methods. Our modified RDG is described in Section III, and evaluated using numerical experiments in Section IV. In the last section, we conclude the paper and suggest possible directions for future work.

II. Background and Related Work

In this section, we describe CC [7] that tackles a large-scale optimization problem via a divide-and-conquer strategy. CC (Algorithm 1) typically consists of two stages: 1) decomposition: dividing a given high-dimensional problem into a number of low-dimensional sub-problems; and 2) optimization: solving each sub-problem cooperatively using an optimizer.

A. Decomposition Stage

The efficacy of CC heavily relies on a proper problem decomposition, that is to decompose a problem based on its underlying variable interaction structure. Two variables interact if they influence each other in the optimization process. A decomposition is considered as "good" if it minimizes the inter-group and maximizes the intra-group variable interactions [5], [6]. Generally, there are two different approaches that can be used to identify variable interactions based on perturbation: 1) non-monotonicity detection [23], and 2) non-linearity detection [30].

The non-monotonicity detection method identifies variable interactions by detecting non-monotonicity in fitness function when perturbing decision variables. If the monotonicity of fitness function with respect to variable x_i does not change for different values of x_j, x_i and x_j are independent; otherwise they interact. Decomposition methods in this category include variable interaction learning [9], statistical variable interdependence learning [18] and fast variable interdependence searching [31]. These methods may require more samples to identify a non-monotonicity relationship, thus are typically more computationally expensive than non-linearity detection.

The non-linearity detection method identifies variable interactions by detecting the non-linearity in fitness changes when perturbing decision variables. If the fitness change induced by perturbing decision variable x_i varies for different values of x_j, x_i and x_j interact. The decomposition methods in this line include differential grouping [6], XDG, GDG, DG2 and fast interdependency identification [32]. These methods typically require $\mathcal{O}(n^2)$ FEs for decomposing an n-dimensional problem. The RDG method has reduced the decomposition cost to $\mathcal{O}(n\log(n))$. We will further describe RDG in Section II-C, as our proposition in Section III is closely related to it.

B. Optimization Stage

In the optimization stage, the sub-problems are optimized iteratively using an optimizer in a cooperative manner. When optimizing the i_{th} sub-problem, a context vector is used to assist the evaluation of the individuals in the sub-problem. The context vector is a complete candidate solution, typically consisting of the best sub-solutions from each sub-problem. The context vector (excluding the i_{th} sub-solution) is used to combine with an individual in the i_{th} sub-problem, so a complete candidate solution can be formed and evaluated. The context vector will be updated if a better sub-solution is found for the i_{th} sub-problem.

Algorithm 1 Cooperative Co-evolution

```
Divide decision variables X into components X_i, 1 ≤ i ≤ m
Initialize a context vector x* (a complete candidate solution)
for j from 1 to max_cycles do
    for i from 1 to m do
        Sample sub-solutions x_i s for X_i using an optimizer
        Evaluate the fitness of each x_i, combined with x*
        Update x* if a better sub-solution x_i is found
    end for
end for
return the best solution found x*
```

The original CC [7] optimizes sub-problems in a round-robin fashion, thus computational resources are evenly distributed to each sub-problem. However if the sub-problems contribute very differently to the overall fitness value, such an allocation policy may be inefficient. Thus there is a recent trend to adaptively allocate computational resources to sub-problems based on their contribution to the overall fitness improvement [28], [33]–[38]. In Section IV-D, we will empirically investigate the efficacy of contribution-based CC on overlapping problems.

C. Recursive Differential Grouping

In this sub-section, we describe the RDG method in detail and discuss the issues of RDG when dealing with overlapping problems. The RDG method identifies the interaction between two subsets of variables X_1 and X_2 based on a measure of non-linearity detection (see Fig. 2 for an example):

Theorem 1. *(Sun et al. [10]) Let* $f\colon \mathbb{R}^n \to \bar{\mathbb{R}}$ *be an objective function;* $X_1 \subset X$ *and* $X_2 \subset X$ *be two mutually exclusive subsets of decision variables:* $X_1 \cap X_2 = \emptyset$*.* X_1 *and* X_2 *interact, if there exist a candidate solution* $\mathbf{x}^*$ *and sub-vectors* $\mathbf{a}_1$, $\mathbf{a}_2$, $\mathbf{b}_1$, $\mathbf{b}_2$*, such that the non-linearity term* λ *is non-zero:*

$$\lambda(\mathbf{x}, \mathbf{x}_1, \mathbf{x}_2) := |\Delta_1 - \Delta_2| \neq 0, \tag{1}$$

where

$$\Delta_1 := f(\mathbf{x}^*)|_{\mathbf{x}_1=\mathbf{a}_1, \mathbf{x}_2=\mathbf{b}_1} - f(\mathbf{x}^*)|_{\mathbf{x}_1=\mathbf{a}_2, \mathbf{x}_2=\mathbf{b}_1}, \tag{2}$$

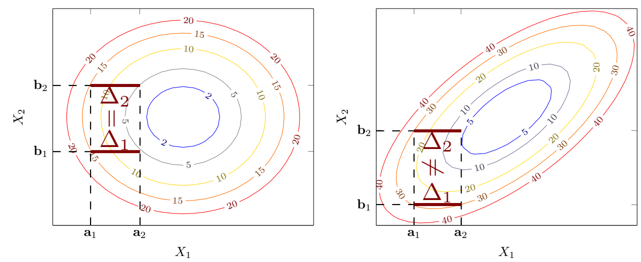
$$\Delta_2 := f(\mathbf{x}^*)|_{\mathbf{x}_1=\mathbf{a}_1, \mathbf{x}_2=\mathbf{b}_2} - f(\mathbf{x}^*)|_{\mathbf{x}_1=\mathbf{a}_2, \mathbf{x}_2=\mathbf{b}_2}. \tag{3}$$

Here, $f(\mathbf{x}^*)|_{\mathbf{x}_1=\mathbf{a}_i, \mathbf{x}_2=\mathbf{b}_j}$ *calculates the objective value of* $\mathbf{x}^*$ *when replacing* X_1 *with* $\mathbf{a}_i$*, and* X_2 *with* $\mathbf{b}_j$*.*

In theory, any positive value of the non-linearity term λ implies an interaction between the subsets of decision variables under examination. However in practice, the value of λ for separable decision variables may be non-zero, due to the computational round-off errors incurred by the floating-point operations [15]. In [39] we applied the technique suggested by DG2 [15] to estimate an upper bound on the round-off errors associated with the calculation of the non-linearity term λ:

$$\epsilon := \gamma_{\sqrt{n}+2}\big(|f(\mathbf{x}^*_{1,1})| + |f(\mathbf{x}^*_{2,1})| + |f(\mathbf{x}^*_{1,2})| + |f(\mathbf{x}^*_{2,2})|\big). \tag{4}$$

Here $f(\mathbf{x}^*_{i,j})$ stands for $f(\mathbf{x}^*)|_{\mathbf{x}_1=\mathbf{a}_i, \mathbf{x}_2=\mathbf{b}_j}$; n is the dimensionality; and $\gamma_k := k\mu_{\mathrm{M}}/(1-k\mu_{\mathrm{M}})$, where μ_{M} is a machine dependent constant. The upper bound is then used as the threshold value to distinguish between separable and non-separable variables for RDG2 [39].



(a) (b)

Fig. 2: The rationale behind the non-linearity detection method when identifying (a) separable and (b) non-separable subsets of decision variables. In the separable contour plot (a), the fitness change induced by perturbing the decision variable subset X_1 is the same for different values of X_2. However in the non-separable contour plot (b), the fitness change induced by perturbing X_1 varies for different values of X_2.

With Theorem 1, the interaction between two subsets of decision variables (X_1 and X_2) can be identified by the following procedure:

1) Set all the decision variables to the lower bounds ($\mathbf{lb}$) of the search space ($\mathbf{x}_{l,l}$);
2) Perturb the decision variables X_1 of $\mathbf{x}_{l,l}$ from the lower bounds to the upper bounds ($\mathbf{ub}$), denoted as $\mathbf{x}_{u,l}$;
3) Calculate the fitness change Δ_1 between $\mathbf{x}_{l,l}$ and $\mathbf{x}_{u,l}$;
4) Perturb decision variables X_2 of $\mathbf{x}_{l,l}$ ($\mathbf{x}_{u,l}$) from $\mathbf{lb}$ to the middle of the search space, denoted as $\mathbf{x}_{l,m}$ ($\mathbf{x}_{u,m}$);
5) Calculate the fitness change Δ_2 between $\mathbf{x}_{l,m}$ and $\mathbf{x}_{u,m}$;
6) If the difference (λ) between Δ_1 and Δ_2 is greater than the threshold ϵ, X_1 and X_2 interact.

The two subscripts of $\mathbf{x}$ denote the values of X_1 and X_2 respectively: 'l' is lower bound; 'u' is upper bound; and 'm' is the mean of lower and upper bounds.

The decomposition procedure of RDG can be briefly summarized into three steps: 1) identifying the decision variables that interact with a selected variable x_i, and placing them into a subset X_1; 2) recursively identifying and grouping the decision variables that interact with any variable in X_1, until X_1 is independent of the remaining variables; and 3) repeating step 1) and 2) until all variables have been grouped. Thus in an overlapping problem, all decision variables will be assigned into one group as they are all linked (either directly or indirectly). In the next section we will modify the RDG method to effectively decompose an overlapping problem.

III. Decomposition for Overlapping Problems

In this section we modify the RDG method to effectively decompose an overlapping problem. The basic idea is to break the linkage at shared variables, by placing shared variables into either of the overlapping components. Considering an example in Fig. 1b, our desired decomposition is to assign x_4 to either

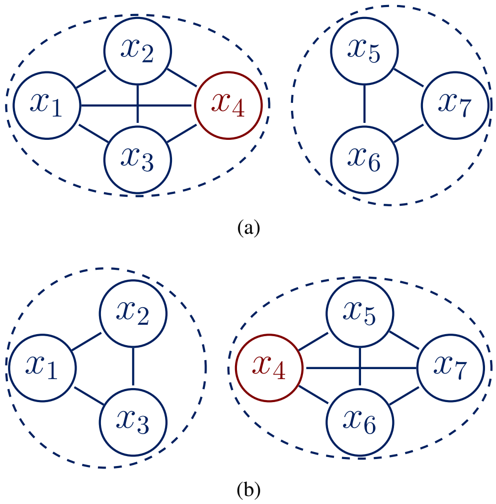


Fig. 3: The desired decompositions (a) or (b) for the overlapping problem in Fig 1b. The idea is to break the linkage at shared variables, such that the level of interaction between components is low.

of the two components as shown in Fig. 3. For simplicity, we refer to our modification as RDG3, to distinguish it from the previous versions proposed in [10], [39].

The same as its predecessors, RDG3 begins by identifying the interaction between the first decision variable x_1 and the remaining decision variables. If no interaction is detected, x_1 will be placed in the separable decision variable set S, and the algorithm will move on to the next decision variable x_2. If any interaction is detected, the remaining decision variables will be randomly divided into two (nearly) equally-sized groups G_1 and G_2. Then the interaction between x_1 and G_1, x_1 and G_2 will be identified respectively. This process is recursively conducted until all the individual decision variables that interact with x_1 are identified and placed in the decision variable subset X_1 with x_1.

In the next step, a threshold ϵ_n is imposed on the size of X_1 to handle overlapping problems. If the size of X_1 is less than ϵ_n ($|X_1| < \epsilon_n$), RDG3 further examines the interaction between X_1 and the remaining variables (excluding X_1) to identify the variables that indirectly interact with x_1 (linked by other variables). If any interaction is identified, the interacting decision variables will be placed into X_1. This process is repeated until $|X_1| \geq \epsilon_n$ or no interaction can be further detected between X_1 and the remaining variables (line 9). The variables in X_1 will be treated as a non-separable group.

The RDG3 method moves on to the next decision variable that has not been grouped (x_i), and the above process is repeated until all decision variables have been grouped. Different from its predecessors, RDG3 further divides the separable variables (stored in S) into small groups with an interval ϵ_s (line 23 to 30). That is to break the set S into subsets at the ϵ_s, $2\epsilon_s$, $\cdots$ $k\epsilon_s$ elements, where $k = \lfloor |S|/\epsilon_s \rfloor$. Finally RDG3 returns the identified separable variable groups (*seps*) and non-separable variable groups (*nonseps*) as the outputs.

Algorithm 2 RDG3 for Overlapping Problems

Require: f, **ub**, **lb**, ϵ_n, ϵ_s, n
1: Initialize *seps* and *nonseps* as *empty* groups
2: Initialize S as empty (to store separable variables)
3: Set all decision variables to the lower bounds: $\mathbf{x}_{l,l} \leftarrow \mathbf{lb}$
4: Calculate the fitness: $y_{l,l} \leftarrow f(\mathbf{x}_{l,l})$
5: Assign the first variable x_1 to the variable subset X_1
6: Assign the rest of variables to the variable subset X_2
7: **while** X_2 is not *empty* **do**
8: $[X_1^*] \leftarrow$ INTERACT(X_1, X_2, $\mathbf{x}_{l,l}$, $y_{l,l}$, n)
9: **if** $|X_1^*| \geq \epsilon_n$ or $|X_1^*| = |X_1|$ **then**
10: **if** X_1 contains one decision variable **then**
11: Add X_1 to S for further decomposition
12: **else**
13: Add X_1 to *nonseps* as a component
14: **end if**
15: Empty X_1 and X_1^*
16: Assign the first variable of X_2 to X_1
17: Delete the first variable in X_2
18: **else**
19: $X_1 \leftarrow X_1^*$
20: Delete the variables of X_1 from X_2
21: **end if**
22: **end while**
23: **while** S is not empty **do**
24: **if** $|S| < \epsilon_s$ **then**
25: Add S as a group to *seps*, and empty S
26: **else**
27: Add the first ϵ_s variables in S as a group to *seps*
28: Delete the first ϵ_s variables from S
29: **end if**
30: **end while**
31: **return** *seps* and *nonseps*

1: **function** INTERACT(X_1, X_2, $\mathbf{x}_{l,l}$, $y_{l,l}$, n)
2: $\mathbf{x}_{u,l} \leftarrow \mathbf{x}_{l,l}$; $\mathbf{x}_{u,l}(X_1) \leftarrow \mathbf{ub}(X_1)$ //Set X_1 to the **ub**
3: Calculate the fitness of $\mathbf{x}_{u,l}$: $y_{u,l} \leftarrow f(\mathbf{x}_{u,l})$
4: Calculate the fitness change: $\delta_1 \leftarrow y_{l,l} - y_{u,l}$
5: $\mathbf{x}_{l,m} \leftarrow \mathbf{x}_{l,l}$; $\mathbf{x}_{l,m}(X_2) \leftarrow \big(\mathbf{lb}(X_2) + \mathbf{ub}(X_2)\big)/2$
6: $\mathbf{x}_{u,m} \leftarrow \mathbf{x}_{u,l}$; $\mathbf{x}_{u,m}(X_2) \leftarrow \big(\mathbf{lb}(X_2) + \mathbf{ub}(X_2)\big)/2$
7: Calculate the fitness: $y_{l,m} \leftarrow f(\mathbf{x}_{l,m})$; $y_{u,m} \leftarrow f(\mathbf{x}_{u,m})$
8: Calculate the fitness change: $\delta_2 \leftarrow y_{l,m} - y_{u,m}$
9: Estimate $\epsilon \leftarrow \gamma_{\sqrt{n}+2}\big(|y_{l,l}| + |y_{u,l}| + |y_{l,m}| + |y_{u,m}|\big)$
10: **if** $|\delta_1 - \delta_2| > \epsilon$ **then**
11: **if** X_2 contains one variable **then**
12: $X_1 \leftarrow X_1 \cup X_2$
13: **else**
14: Divide X_2 into equally-sized groups G_1, G_2
15: $[X_1^1] \leftarrow$ INTERACT(X_1, G_1, $\mathbf{x}_{l,l}$, $y_{l,l}$, ϵ)
16: $[X_1^2] \leftarrow$ INTERACT(X_1, G_2, $\mathbf{x}_{l,l}$, $y_{l,l}$, ϵ)
17: $[X_1] \leftarrow X_1^1 \cup X_1^2$
18: **end if**
19: **end if**
20: **return** X_1
21: **end function**

The main difference between RDG3 and its predecessors is highlighted in red in Algorithm 2.

We introduce the threshold ϵ_n and ϵ_s in the hope that a large-scale problem can be decomposed into reasonably-sized components. On one hand it is a waste of computational resources to optimize a very small-sized component. On the other hand a large-sized component is typically not manageable by optimization algorithms. More importantly by tuning the threshold ϵ_n, it is possible to break the linkage at shared variables for an overlapping problem. Again consider the example in Fig. 1b and $\epsilon_n = 4$. If searching from x_1, the variables $\{x_1, x_2, x_3, x_4\}$ will be placed in a subset X_1 after the first step. As $|X_1| \geq \epsilon_n$, X_1 will be treated as a component. The remaining variables $\{x_5, x_6, x_7\}$ will be identified as another component. The decomposition in this case is identical to the one shown in Fig. 3a. Similarly if starting from x_7, the decomposition is identical to Fig. 3b. Note that the decomposition of RDG3 is dependent on the order in which variables are explored. Designing a sophisticated exploration order for a black-box optimization problem is non-trivial and requires additional computational effort, thus we simply use variable index as the exploration order in this paper.

IV. Experiments

In this section we use simulation experiments to evaluate the efficacy of RDG3. All experiments were performed in MATLAB. The source codes of RDG3 and the benchmark problems used in our experiments are available online.[1]

A. Benchmarking Overlapping Problems

To systemically evaluate the efficacy of RDG3, we extend two CEC'2013 overlapping problems (f_{13} and f_{14}) [27], considering various level of overlap between components.

The CEC'2013 f_{13} and f_{14} consist of 20 components, and the adjacent components are designed to share m ($m = 5$) common decision variables to impose overlap. The overlapping effects in f_{13} and f_{14} are very different; the former is conforming and the latter is conflicting [27], [40]. In a problem with conforming overlapping components, a shared decision variable has the same optimal value across overlapping components. For example, if decision variable x_i is in component C_1 and C_2, the optimal value of x_i in C_1 is also optimal for C_2. However in a problem with conflicting overlapping components, the optimal value of a shared decision variable may not be the same in different components.

We extend the CEC'2013 f_{13} and f_{14}, by varying the parameter m from 1 to 10, resulting in 10 benchmark problems for each of the conforming and conflicting categories. We denote the conforming and conflicting problems as $f_{o,m}$ and $f_{l,m}$ respectively, where $m = 1, 2 \cdots 10$. Therefore, a suite of 20 overlapping benchmark problems are created in total. Each problem is designed to have 20 components with 1000 decision variables in total. As adjacent components share m decision variables, the problem dimension is thus $n = 1000 - 19m$. The global optimum for a conforming problem is 0, as all components can be minimized to 0 simultaneously. However the optimal value for a conflicting problem is unknown. As a shared decision variable may have different optimal values in overlapping components, it is not possible to simultaneously solve each component to optimality 0. In this case, the global optimum of the whole problem is thus greater than 0.

[1] https://bitbucket.org/yuans/rdg3

B. Decomposition Effects on Overlapping Problems

Methodology: The RDG3 method is used to decompose the 20 overlapping benchmark problems designed in Section IV-A. Different threshold values $\epsilon_n = 0, 50, 100, 1000$ are tested. As the dimensions of all the benchmark problems are less than 1000, RDG3 with $\epsilon_n = 1000$ is expected to group all variables into a component. It has been found that neither a very large nor small value of ϵ_s is beneficial to the performance of CC [41]. Thus we simply set ϵ_s to 100 in this paper. The number of components generated (n_c), the average component size ($\bar{s}$) and the number of FEs used are reported in Table I. We then use CMA-ES [29] to solve the components in a round-robin fashion. The parameter setting for CMA-ES is consistent with the original paper. The computational budget for the decomposition and optimization stages is set to 3×10^6 FEs in total. The mean of best solutions ($\bar{y}$) generated from 40 independent runs is reported in Table I; the best results are determined using Wilcoxon rank-sum test (significance level = 0.05) with Holm p-value correction [42].

Results: We observe in Table I that as ϵ_n increases, the number of components (n_c) generated by RDG3 decreases; the average component size $\bar{s}$ increases; and the number of FEs used in decomposition is roughly the same. RDG3 with $\epsilon_n = 1000$ is significantly outperformed by the ones with other parameter settings. In fact when $\epsilon_n = 1000$, RDG3 is equivalent to the RDG (or RDG2) method, that groups all linked variables into one component. The results suggest that overlapping problems can benefit from a divide-and-conquer strategy, and can potentially be solved in a more effective way. Further $\epsilon_n = 50$ is a robust parameter setting for RDG3. It significantly outperforms the other parameter settings on conforming problems ($f_{o,1}$ to $f_{o,10}$); and generates comparable solution quality with $\epsilon_n = 0$ on conflicting problems ($f_{l,0}$ to $f_{l,10}$). Thus we will use $\epsilon_n = 50$ in the following experiments. Last we note the threshold value used to identify variable interactions (Eq. 4) is very conservative, resulting in some non-separable variables being classified as separable. This is why RDG3 with $\epsilon_n = 1000$ generates more than one component for some benchmark problems.

C. Comparison on Overlapping Problems

Methodology: We compare the performance of RDG3 against DG2, RG, and delta grouping when incorporated with CMA-ES to solve the overlapping benchmark problems. DG2 is a state-of-the-art method, however similar to RDG it cannot effectively decompose overlapping problems. The RG method groups decision variables randomly in each evolutionary cycle, while delta grouping groups variables based on a measure of averaged variable differences. As a baseline, we also compare RDG3 to a variant of RG, denoted as RG2, that randomly groups decision variables in the first iteration, and remains unchanged until the end of an optimization run. For RDG3, ϵ_n is set to 50 and ϵ_s is 100; for RG, RG2, and delta grouping the maximal component size is set to 100. The mean and standard deviation of the best solutions generated in 40 runs

TABLE I: Decomposition and optimization results of RDG3 with different ϵ_n values on the overlapping benchmark problems. n_c is the number of components generated; $\bar{s}$ is the average component size; FEs is the number of function evaluations used in decomposition; and $\bar{y}$ is the mean of best solution quality generated by a CC algorithm from 40 independent runs. The best solution quality is in bold, according to the Wilcoxon rank-sum tests (significance level = 0.05) with Holm p-value correction.

Fun	$\epsilon_n = 0$				$\epsilon_n = 50$				$\epsilon_n = 100$				$\epsilon_n = 1000$			
	n_c	$\bar{s}$	FEs	$\bar{y}$	n_c	$\bar{s}$	FEs	$\bar{y}$	n_c	$\bar{s}$	FEs	$\bar{y}$	n_c	$\bar{s}$	FEs	$\bar{y}$
$f_{o,1}$	28	35	18778	5.30e+05	18	54	19108	**1.31e+04**	17	57	19111	2.16e+04	8	122	18037	8.12e+06
$f_{o,2}$	21	45	18220	3.78e+05	14	68	18112	**1.46e+04**	9	106	17482	7.20e+04	3	320	16930	5.32e+06
$f_{o,3}$	28	33	17965	9.90e+05	13	72	17431	**3.65e+03**	11	85	16633	1.20e+04	1	943	15454	1.91e+06
$f_{o,4}$	24	38	17644	1.40e+05	13	71	17293	**6.45e+03**	10	92	17044	**5.84e+03**	3	308	15937	2.59e+06
$f_{o,5}$	18	50	16339	**1.27e+04**	14	64	15988	**8.27e+03**	8	113	15913	7.98e+04	2	452	15187	9.24e+05
$f_{o,6}$	26	34	16546	**1.02e+04**	16	55	16876	**3.17e+03**	14	63	16972	7.49e+04	3	295	14602	1.48e+06
$f_{o,7}$	20	43	15532	4.07e+05	15	57	15622	**5.03e+04**	10	86	15025	9.28e+05	1	867	14554	1.78e+06
$f_{o,8}$	21	40	15814	2.91e+06	14	60	14893	4.34e+05	8	106	14296	**5.42e+04**	1	848	13192	1.20e+06
$f_{o,9}$	18	46	14464	**1.77e+03**	13	63	14476	1.90e+03	9	92	14218	2.24e+04	1	829	14218	1.25e+06
$f_{o,10}$	27	30	13969	1.24e+06	15	54	14503	**1.29e+05**	12	67	14185	2.47e+05	4	202	12979	1.35e+06
$f_{l,1}$	21	46	18793	**7.84e+05**	15	65	17971	8.57e+05	12	81	17632	4.12e+06	2	490	17035	1.50e+07
$f_{l,2}$	21	45	18682	**1.08e+07**	12	80	17854	1.14e+07	8	120	16978	2.70e+07	1	962	17026	4.00e+07
$f_{l,3}$	21	44	17872	**1.03e+07**	13	72	17605	1.18e+07	10	94	17482	1.10e+07	2	471	16951	5.75e+07
$f_{l,4}$	19	48	17047	**1.11e+07**	14	66	16273	1.15e+07	10	92	15799	3.02e+07	1	924	15010	3.46e+07
$f_{l,5}$	21	43	16669	**4.45e+06**	13	69	16288	5.56e+06	9	100	16438	4.94e+06	1	905	16150	2.74e+07
$f_{l,6}$	17	52	14932	1.14e+08	13	68	14902	1.13e+08	11	80	14848	**1.11e+08**	1	886	16216	1.44e+08
$f_{l,7}$	23	37	16324	**1.58e+09**	12	72	16198	**1.60e+09**	8	108	16123	**1.62e+09**	1	867	16582	1.76e+09
$f_{l,8}$	17	49	14848	**2.11e+07**	14	60	14887	2.22e+07	8	106	14812	2.51e+07	1	848	14614	4.81e+07
$f_{l,9}$	18	46	14701	**1.06e+08**	13	63	14962	**1.05e+08**	9	92	14926	1.12e+08	1	829	14647	1.64e+08
$f_{l,10}$	21	38	14698	**8.03e+07**	10	81	14863	8.26e+07	8	101	14881	8.16e+07	1	810	13381	1.02e+08

TABLE II: Optimization results of CC-DG2, RG, RG2, Delta and RDG3, as well as CBCC-RDG3 when used to solve the 20 overlapping benchmark problems. CC-RDG3 significantly outperforms the other algorithms across the benchmark suite.

Fun	CC-DG2		CC-RG		CC-RG2		CC-Delta		CC-RDG3		CBCC-RDG3	
	mean	std	mean	std	mean	std	mean	std	mean	std	mean	std
$f_{o,1}$	3.00e+06	4.74e+05	1.94e+11	2.68e+11	4.59e+06	1.64e+07	2.99e+11	1.67e+11	**1.31e+04**	4.56e+03	8.39e+06	1.52e+07
$f_{o,2}$	3.20e+06	3.22e+05	7.27e+10	2.77e+10	2.84e+06	7.74e+06	8.71e+10	1.64e+10	**1.46e+04**	8.61e+03	3.46e+07	2.67e+07
$f_{o,3}$	3.18e+06	3.80e+05	8.46e+10	2.03e+10	1.16e+06	1.40e+06	5.67e+10	8.03e+09	**3.83e+03**	3.41e+03	1.07e+06	2.78e+05
$f_{o,4}$	4.22e+06	4.54e+05	6.78e+10	1.62e+10	8.59e+05	3.44e+05	5.56e+10	1.10e+10	**6.72e+03**	5.24e+03	2.48e+05	1.53e+05
$f_{o,5}$	2.39e+06	2.36e+05	6.81e+10	1.83e+10	3.15e+08	7.94e+08	8.19e+10	2.01e+10	**8.24e+03**	3.09e+03	6.28e+04	2.83e+04
$f_{o,6}$	4.05e+06	4.75e+05	7.83e+10	4.09e+10	7.95e+07	3.63e+08	4.91e+10	1.29e+10	**2.92e+03**	2.76e+03	2.76e+05	1.45e+05
$f_{o,7}$	1.93e+06	2.52e+05	9.91e+10	9.81e+10	1.89e+06	5.81e+06	7.45e+10	1.56e+10	**5.17e+04**	2.99e+04	2.87e+08	1.49e+08
$f_{o,8}$	1.93e+06	2.92e+05	7.12e+10	2.32e+10	**3.25e+08**	1.49e+09	1.15e+11	2.15e+10	**4.84e+05**	4.76e+05	9.42e+08	4.66e+08
$f_{o,9}$	1.81e+06	2.70e+05	7.97e+10	2.79e+10	7.38e+05	4.94e+05	6.07e+10	1.42e+10	**2.01e+03**	1.08e+03	6.74e+07	2.55e+07
$f_{o,10}$	3.51e+06	5.13e+05	1.11e+11	4.28e+10	2.11e+07	5.64e+07	2.42e+11	1.24e+11	**1.23e+05**	8.28e+04	8.63e+07	5.83e+07
$f_{l,1}$	4.40e+07	3.90e+06	9.69e+11	3.20e+11	4.15e+06	9.66e+05	9.04e+11	2.28e+11	**8.64e+05**	5.13e+04	4.32e+10	1.49e+09
$f_{l,2}$	5.12e+07	4.31e+06	1.36e+12	4.57e+11	1.55e+07	1.97e+06	1.09e+12	2.26e+11	**1.14e+07**	5.30e+05	6.55e+08	4.79e+08
$f_{l,3}$	4.57e+07	2.52e+06	3.90e+12	4.95e+12	1.44e+07	2.17e+06	2.39e+12	5.03e+11	**1.17e+07**	4.51e+05	1.76e+10	4.30e+09
$f_{l,4}$	7.92e+07	7.68e+06	1.11e+12	3.55e+11	1.47e+07	1.58e+06	1.18e+12	2.31e+11	**1.15e+07**	4.27e+05	1.23e+07	6.49e+05
$f_{l,5}$	3.58e+07	2.49e+06	8.43e+11	2.49e+11	1.53e+09	6.81e+09	7.85e+11	1.98e+11	**5.57e+06**	2.83e+05	1.62e+09	2.06e+09
$f_{l,6}$	1.51e+08	3.09e+06	9.92e+11	3.64e+11	1.18e+08	2.26e+06	7.71e+11	1.69e+11	**1.13e+08**	2.25e+06	6.80e+09	1.22e+10
$f_{l,7}$	1.76e+09	1.21e+08	1.24e+12	5.15e+11	1.67e+09	1.08e+08	9.84e+11	2.30e+11	**1.59e+09**	7.11e+07	1.75e+09	1.38e+08
$f_{l,8}$	5.64e+07	2.73e+06	1.76e+12	8.62e+11	2.52e+07	1.36e+06	3.38e+12	1.50e+12	**2.22e+07**	1.48e+06	1.88e+08	1.32e+08
$f_{l,9}$	1.69e+08	1.55e+07	1.16e+12	3.32e+11	1.11e+08	4.10e+06	1.33e+12	2.19e+11	**1.05e+08**	5.30e+06	3.26e+08	2.60e+08
$f_{l,10}$	1.07e+08	3.10e+06	1.45e+12	4.33e+11	2.01e+09	6.03e+09	1.80e+12	3.31e+11	**8.27e+07**	2.49e+06	2.27e+09	2.53e+09

are reported in Table II. The same statistical tests are used as before to identify the best results.

Results: RDG3 significantly outperforms the DG2 method across the benchmark suite. DG2 aims at grouping all linked variables into one component, thus all decision variables result in one group for overlapping problems. By decomposing overlapping problems into components that are optimized cooperatively, RDG3 is able to greatly improve the solution quality. However a "blind" decomposition, i.e., not explicitly considering variable interaction structure, is detrimental to optimization for overlapping problems. This can be inferred from the results generated by RG, RG2 and delta grouping.

D. Contribution-Based CC on Overlapping Problems

Methodology: A contribution-based CC (CBCC) allocates computational resources to components based on their contribution to the overall fitness improvement. A number of studies has reported that CBCC is more effective than CC when used to solve problems with separable components [28], [33]–[38]. Here we evaluate the efficacy of a CBCC algorithm on overlapping problems. In each evolutionary cycle, the component that contributes the most to overall fitness improvement is selected and evolved. We use the exponential smoothing method to measure the contribution of a component [28]:

$$U = \alpha \hat{U} + (1-\alpha)(\hat{y}_b - y_b)/\hat{y}_b, \quad (5)$$

TABLE III: Optimization results of CC-GDG, DG2, RDG, RDG2, RDG3 as well as CBCC-RDG3 when used to solve the CEC'2013 benchmark problems. The best solution quality is in bold, determined by Wilcoxon rank-sum tests (significance level = 0.05) with Holm p-value correction.

Fun	CC-GDG		CC-DG2		CC-RDG		CC-RDG2		CC-RDG3		CBCC-RDG3	
	mean	std	mean	std	mean	std	mean	std	mean	std	mean	std
f_1	**1.04e-20**	9.90e-22	5.52e+05	5.88e+04	2.90e+05	3.28e+04	2.78e+05	3.17e+04	9.67e-19	1.23e-19	1.14e-18	1.27e-19
f_2	**1.54e+03**	7.52e+01	4.69e+03	1.81e+02	4.69e+03	1.78e+02	4.71e+03	2.05e+02	2.36e+03	1.11e+02	2.31e+03	1.06e+02
f_3	2.04e+01	4.28e-02	2.04e+01	5.21e-02	2.04e+01	4.96e-02	2.04e+01	4.35e-02	2.04e+01	6.21e-02	2.04e+01	5.95e-02
f_4	7.31e+04	3.72e+04	8.52e+06	8.54e+05	5.83e+06	6.32e+05	5.83e+06	6.32e+05	**1.61e+04**	9.06e+03	**4.29e+04**	7.21e+04
f_5	2.23e+06	4.24e+05	2.19e+06	3.51e+05	2.40e+06	4.36e+05	2.23e+06	3.23e+05	2.27e+06	3.02e+05	2.04e+06	3.13e+05
f_6	9.96e+05	1.70e+03	9.96e+05	3.31e+02	9.96e+05	1.48e+02	9.96e+05	6.55e+01	9.96e+05	4.71e+02	1.00e+06	2.48e+04
f_7	3.73e+07	1.30e+07	1.05e+03	2.79e+02	8.12e-17	2.17e-16	4.05e-16	1.49e-15	1.01e-03	3.26e-03	**1.71e-21**	2.39e-22
f_8	1.28e+08	3.52e+07	3.85e+07	1.09e+07	8.51e+06	2.92e+06	8.70e+06	3.61e+06	1.24e+07	5.01e+06	**7.11e+03**	2.30e+03
f_9	1.67e+08	3.88e+07	1.51e+08	2.87e+07	1.65e+08	4.16e+07	1.67e+08	2.66e+07	1.45e+08	3.15e+07	1.57e+08	2.90e+07
f_{10}	9.11e+07	1.20e+06	9.13e+07	1.51e+06	9.10e+07	1.29e+06	9.11e+07	1.31e+06	9.11e+07	1.43e+06	9.16e+07	2.18e+06
f_{11}	2.53e+07	2.69e+06	2.47e+05	2.37e+05	1.67e+07	1.62e+06	8.69e+03	1.24e+04	9.71e+03	1.46e+04	**2.18e-13**	1.02e-12
f_{12}	1.00e+03	3.91e+01	1.01e+03	5.81e+01	9.81e+02	7.30e+01	9.81e+02	7.30e+01	9.88e+02	9.31e+00	**7.00e+02**	1.46e+02
f_{13}	2.36e+06	3.38e+05	2.43e+06	3.70e+05	2.47e+06	3.83e+05	9.31e+05	1.60e+05	**8.24e+03**	3.09e+03	6.43e+04	4.40e+04
f_{14}	3.63e+07	3.18e+06	3.59e+07	2.85e+06	2.77e+07	1.80e+06	2.68e+07	1.89e+06	**5.57e+06**	2.83e+05	1.65e+09	1.33e+09
f_{15}	3.05e+06	3.35e+05	3.02e+06	3.30e+05	**2.19e+06**	2.28e+05	**2.26e+06**	2.45e+05	**2.37e+06**	6.94e+05	**2.30e+06**	2.17e+05

where $\hat{y}_b$ and y_b are the best fitness values found before and after evolving a component; $\hat{U}$ is the previous contribution of the component; and α is the smoothing factor set to 0.5. The calculation of U considers all fitness improvements in previous cycles, with the weight decaying exponentially. We set the number of FEs in each cycle to 1000. The decomposition method used is RDG3 with $\epsilon_n = 50$ and $\epsilon_s = 100$, and the component solver is CMA-ES. CBCC-RDG3 is compared against CC-RDG3, and the results are reported in Table II.

Results: The CBCC model used in the paper is consistently outperformed by the conventional CC across the benchmark problems. This may indicate that adaptively allocating computational resources is inefficient when tackling overlapping problems. We infer the reason is rooted in the interaction between components. The optimization state of a component, i.e., how close to optimality, is highly dependent on others, making the contribution of a component unpredictable especially for conflicting problems. However more research needs to be done, e.g., testing other CBCC models on overlapping problems, before any conclusion can be drawn.

E. Comparison on CEC'2013 Benchmark Problems

Methodology: In this sub-section, we perform three sets of comparisons on the CEC'2013 benchmark problems: 1) RDG3 ($\epsilon_n = 50$ and $\epsilon_s = 100$) versus RDG, RDG2, DG2 and GDG; 2) CC versus CBCC (used in Section IV-D) with RDG3 as the decomposition method; and 4) CC-RDG3 versus 9 state-of-the-arts listed in the TACO website.[2]

Results: We observe in Table III that RDG3 significantly outperforms the other four decomposition methods on overlapping problems f_{13} and f_{14}. RDG3 can generate significantly better solution quality than RDG2 for problems with separable variables e.g., f_1, f_2 and f_4, suggesting it is useful to further decompose separable variables into small components. CBCC-RDG3 significantly improves over CC-RDG3 on problems with separable components, e.g., f_7, f_8 and f_{11}; however it is outperformed on problems with overlapping components f_{13} and f_{14}. It confirms our previous observation that adaptive allocation of computational resources is not helpful when dealing with overlapping problems. Finally our algorithm CC-RDG3 generates the best solution quality for 7 out of 15 benchmark problems, when compared against the results of 9 other algorithms available on the TACO website.[2]

[2]https://tacolab.org

V. Conclusion

We tackled large-scale optimization problems with overlapping components using a divide-and-conquer approach. We modified the RDG method, denoted as RDG3, such that it can effectively decompose overlapping problems by breaking the linkage at shared (overlapped) variables. To systemically evaluate the efficacy of RDG3, we extended two CEC'2013 overlapping problems by considering various level of overlap. Experimental results showed our decomposition method facilitated problem solving, and outperformed random decomposition as well as other methods on overlapping problems. We also observed that a CBCC algorithm, that adaptively allocates computational resources to components, is ineffective when used to solve overlapping problems. Finally we showed RDG3, when equipped with CMA-ES, is one of the most competitive solvers for the CEC'2013 benchmark problems.

We suggest three potential research directions for future work. First in the existing overlapping benchmark problems, the overlapping effect is generated by adjacent components sharing some decision variables. Designing benchmark problems with more flexible variable interaction structure and richer source of overlap is desired. Second we presented some preliminary results showing that overlapping problems are challenging for a CBCC algorithm to solve. It would be useful to test more CBCC models on overlapping problems. Last the strength of variable interactions may be very different in a given overlapping problem. Breaking weak linkage may be an alternative approach to decompose overlapping problems.

Acknowledgment

This research was supported by an ARC (Australian Research Council) Discovery Grant (DP190101271).

References

[1] M. R. Bonyadi, Z. Michalewicz, M. Wagner, and F. Neumann, "Evolutionary computation for multicomponent problems: opportunities and future directions," in *Optimization in Industry*. Springer, 2019, pp. 13–30.

[2] Z. Michalewicz, "Quo vadis, evolutionary computation?" in *Advances in Computational Intelligence*. Springer, 2012, pp. 98–121.

[3] M. Michalewicz, Z. Michalewicz, and R. Spitty, "Optimising the wine supply chain," in *Proceedings of the Fourteen Australian Wine Industry Technical Conference, Adelaide, Australia. Citeseer*, 2010.

[4] J. Stolk, I. Mann, A. Mohais, and Z. Michalewicz, "Combining vehicle routing and packing for optimal delivery schedules of water tanks," *OR Insight*, vol. 26, no. 3, pp. 167–190, 2013.

[5] T.-L. Yu, D. E. Goldberg, K. Sastry, C. F. Lima, and M. Pelikan, "Dependency structure matrix, genetic algorithms, and effective recombination," *Evolutionary Computation*, vol. 17, no. 4, pp. 595–626, 2009.

[6] M. N. Omidvar, X. Li, Y. Mei, and X. Yao, "Cooperative co-evolution with differential grouping for large scale optimization," *IEEE Transactions on Evolutionary Computation*, vol. 18, no. 3, pp. 378–393, 2014.

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

[8] X. Ma, X. Li, Q. Zhang, K. Tang, Z. Liang, W. Xie, and Z. Zhu, "A survey on cooperative co-evolutionary algorithms," *IEEE Transactions on Evolutionary Computation*, accepted in September 2018.

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

[10] Y. Sun, M. Kirley, and S. K. Halgamuge, "A recursive decomposition method for large scale continuous optimization," *IEEE Transactions on Evolutionary Computation*, vol. 22, no. 5, pp. 647–661, 2018.

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

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

[13] Y. Sun, M. Kirley, and S. K. Halgamuge, "Extended differential grouping for large scale global optimization with direct and indirect variable interactions," in *Proceedings of the 2015 Annual Conference on Genetic and Evolutionary Computation*. ACM, 2015, pp. 313–320.

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

[15] 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 Transactions on Evolutionary Computation*, vol. 21, no. 6, pp. 929–942, 2017.

[16] Y. Sun, M. Kirley, and S. K. Halgamuge, "Quantifying variable interactions in continuous optimization problems," *IEEE Transactions on Evolutionary Computation*, vol. 21, no. 2, pp. 249–264, 2017.

[17] E. Sayed, D. Essam, and R. Sarker, "Dependency identification technique for large scale optimization problems," in *IEEE Congress on Evolutionary Computation*. IEEE, 2012, pp. 1–8.

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

[19] E. Sayed, D. Essam, R. Sarker, and S. Elsayed, "Decomposition-based evolutionary algorithm for large scale constrained problems," *Information Sciences*, vol. 316, pp. 457–486, 2015.

[20] S. Strasser, J. Sheppard, N. Fortier, and R. Goodman, "Factored evolutionary algorithms," *IEEE Transactions on Evolutionary Computation*, vol. 21, no. 2, pp. 281–293, 2017.

[21] A. Song, W.-N. Chen, P.-T. Luo, Y.-J. Gong, and J. Zhang, "Overlapped cooperative co-evolution for large scale optimization," in *IEEE International Conference on Systems, Man, and Cybernetics*. IEEE, 2017, pp. 3689–3694.

[22] B. Werth, E. Pitzer, and M. Affenzeller, "Enabling high-dimensional surrogate-assisted optimization by using sliding windows," in *Proceedings of the Genetic and Evolutionary Computation Conference Companion*. ACM, 2017, pp. 1630–1637.

[23] M. Munetomo and D. E. Goldberg, "Linkage identification by non-monotonicity detection for overlapping functions," *Evolutionary Computation*, vol. 7, no. 4, pp. 377–398, 1999.

[24] M. Pelikan, D. E. Goldberg, and E. Cantú-Paz, "BOA: The Bayesian optimization algorithm," in *Proceedings of the 1st Annual Conference on Genetic and Evolutionary Computation*. Morgan Kaufmann Publishers Inc., 1999, pp. 525–532.

[25] M. Pelikan, D. E. Goldberg, and E. Cantu-Paz, "Linkage problem, distribution estimation, and bayesian networks," *Evolutionary Computation*, vol. 8, no. 3, pp. 311–340, 2000.

[26] T.-L. Yu, K. Sastry, and D. E. Goldberg, "Linkage learning, overlapping building blocks, and systematic strategy for scalable recombination," in *Proceedings of the 7th Annual Conference on Genetic and Evolutionary Computation*. ACM, 2005, pp. 1217–1224.

[27] X. Li, K. Tang, M. N. Omidvar, Z. Yang, and K. Qin, "Benchmark functions for the CEC 2013 special session and competition on large-scale global optimization," *Technical Report, RMIT University*, 2013.

[28] Y. Sun, M. Kirley, and X. Li, "Cooperative co-evolution with online optimizer selection for large-scale optimization," in *Proceedings of the Genetic and Evolutionary Computation Conference*. ACM, 2018, pp. 1079–1086.

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

[30] M. Tezuka, M. Munetomo, and K. Akama, "Linkage identification by nonlinearity check for real-coded genetic algorithms," in *Genetic and Evolutionary Computation Conference*. Springer, 2004, pp. 222–233.

[31] H. Ge, L. Sun, X. Yang, S. Yoshida, and Y. Liang, "Cooperative differential evolution with fast variable interdependence learning and cross-cluster mutation," *Applied Soft Computing*, vol. 36, pp. 300–314, 2015.

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

[33] M. N. Omidvar, X. Li, and X. Yao, "Smart use of computational resources based on contribution for cooperative co-evolutionary algorithms," in *Proceedings of the 13th Annual Conference on Genetic and Evolutionary Computation*. ACM, 2011, pp. 1115–1122.

[34] M. N. Omidvar, B. Kazimipour, X. Li, and X. Yao, "CBCC3-A contribution-based cooperative co-evolutionary algorithm with improved exploration/exploitation balance." in *IEEE Congress on Evolutionary Computation*, 2016, pp. 3541–3548.

[35] 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 Transactions on Evolutionary Computation*, vol. 21, no. 4, pp. 493–505, 2017.

[36] Y.-H. Jia, W.-N. Chen, T. Gu, H. Zhang, H. Yuan, S. Kwong, and J. Zhang, "Distributed cooperative co-evolution with adaptive computing resource allocation for large scale optimization," *IEEE Transactions on Evolutionary Computation*, accepted in March 2018.

[37] Z. Ren, Y. Liang, A. Zhang, Y. Yang, Z. Feng, and L. Wang, "Boosting cooperative coevolution for large scale optimization with a fine-grained computation resource allocation strategy," *IEEE Transactions on Cybernetics*, accepted in August 2018.

[38] "Bandit-based cooperative coevolution for tackling contribution imbalance in large-scale optimization problems," *Applied Soft Computing*, vol. 76, pp. 265 – 281, 2019.

[39] Y. Sun, M. N. Omidvar, M. Kirley, and X. Li, "Adaptive threshold parameter estimation with recursive differential grouping for problem decomposition," in *Proceedings of the Genetic and Evolutionary Computation Conference*. ACM, 2018, pp. 889–896.

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

[41] M. N. Omidvar, Y. Mei, and X. Li, "Effective decomposition of large-scale separable continuous functions for cooperative co-evolutionary algorithms," in *IEEE Congress on Evolutionary Computation*. IEEE, 2014, pp. 1305–1312.

[42] D. J. Sheskin, *Handbook of parametric and nonparametric statistical procedures*. CRC Press, 2003.