Article:
Cheng, R, Omidvar, MN, Gandomi, AH et al. (2019) Solving Incremental Optimization Problems via Cooperative Coevolution. IEEE Transactions on Evolutionary Computation, 23 (5). pp. 762-775. ISSN: 1089-778X

https://doi.org/10.1109/TEVC.2018.2883599

Solving Incremental Optimization Problems via Cooperative Coevolution

Ran Cheng, Mohammad Nabi Omidvar, Amir H. Gandomi, Bernhard Sendhoff, Stefan Menzel and Xin Yao, *Fellow, IEEE*

***Abstract*—Engineering designs can involve multiple stages, where at each stage, the design models are incrementally modified and optimized. In contrast to traditional dynamic optimization problems where the changes are caused by some objective factors, the changes in such incremental optimization problems are usually caused by the modifications made by the decision makers during the design process. While existing work in the literature is mainly focused on traditional dynamic optimization, little research has been dedicated to solving such incremental optimization problems. In this work, we study how to adopt cooperative coevolution to efficiently solve a specific type of incremental optimization problems, namely, those with increasing decision variables. First, we present a benchmark function generator on the basis of some basic formulations of incremental optimization problems with increasing decision variables and exploitable modular structure. Then, we propose a contribution based cooperative coevolutionary framework coupled with an incremental grouping method for dealing with them. On one hand, the benchmark function generator is capable of generating various benchmark functions with various characteristics. On the other hand, the proposed framework is promising in solving such problems in terms of both optimization accuracy and computational efficiency. In addition, the proposed method is further assessed using a real-world application, i.e., the design optimization of a stepped cantilever beam.**

***Index Terms*—incremental optimization problem, cooperative coevolution, variable grouping, experience based optimization**

I. Introduction

OPTIMIZATION problems are widely seen in various areas of science and engineering. Some of the optimization problems have static objective functions, whereas the others have dynamic objective functions changing over time, known as the dynamic optimization problems (DOPs) [1]. Generally speaking, the changes in DOPs may affect the objective functions, the decision variables, or the constraints, where the reasons causing such changes can be attributed to the variance of available resources, the arrival of new jobs, the environmental changes, etc [2]. For such DOPs, a widely accepted assumption is that they must be solved online as time goes by [3]. In other words, since the changes are caused by some objective factors, there are often some hard constraints in computational time for each optimization period.

While the changes in traditional DOPs are caused by some objective factors, there exist another kind of optimization problems with dynamic changes made by the decision maker via incremental modifications. In general, the incremental modifications can be the addition/removal of decision variables from the objective function or the available constraints. For example, in truss topology optimization, the truss structures can be incrementally optimized by adding nodes and bars into the truss structure [4]; in aerodynamic shape optimization, the aerodynamic shape can be incrementally optimized by adding design parameters into the simulation system [5], [6]. Such problems are incrementally modified by adding decision variables. For simplicity, we denote such problems as the incremental optimization problems (IOPs for short) hereafter.

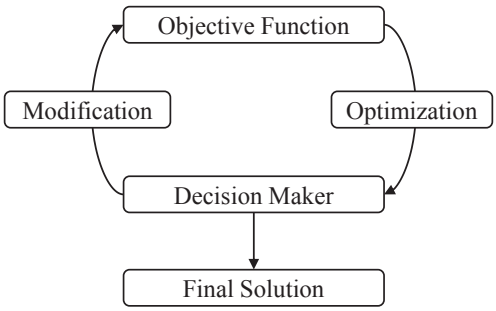


Fig. 1. Illustration to the optimization process of an incremental optimization problem with increasing decision variables (IOP).

As illustrated in Fig. 1, solving an IOP often involves two iterative steps: optimization and modification, where the decision maker incrementally modifies

Manuscript received –. This work was supported by two EPSRC grants (Nos. EP/P005578/1 and No. EP/J017515/1). Xin Yao was supported by a Royal Society Wolfson Research Merit Award and Honda Research Institute Europe. (*The first two authors contributed equally to this work.*)

Ran Cheng, Mohammad Nabi Omidvar and Xin Yao are with the Center of Excellence for Research in Computational Intelligence and Applications (CERCIA), School of Computer Science, University of Birmingham, Birmingham B15 2TT, U.K. (e-mail: ranchengcn@gmail.com, m.omidvar@cs.bham.ac.uk, x.yao@cs.bham.ac.uk).

Amir H. Gandomi is with Analytics & Information Systems at School of Business, Stevens Institute of Technology, Hoboken, NJ 07030, USA (email: a.h.gandomi@stevens.edu).

Bernhard Sendhoff and Stefan Menzel are with the Honda Research Institute Europe GmbH, Offenbach 63073, Germany. (e-mail: bernhard.sendhoff@honda-ri.de, stefan.menzel@honda-ri.de)

Xin Yao (*the corresponding author*) is also with the Department of Computer Science and Engineering, Southern University of Science and Technology, Shenzhen 518055, China.

the objective function by adding more decision variables. At each design stage, the objective function is optimized and its result is given back to the decision maker. Subsequently, the decision maker may change the design by modifying the objective function on the basis of the optimization results. This incremental optimization process continues until the decision maker is satisfied with the final solution.

Broadly speaking, the IOPs can be seen as a special type of the time-linkage problems (DTPs), where the future behavior of a system is influenced by the decision made at each time stage [7], [8]. When solving DTPs, a problem solver is expected to take the problem changes into account and make corresponding predictions [9]. In practice, however, it is difficult to make such predictions due to the black-box nature of the problem. Even worse, an algorithm can be deceived to make wrong predictions causing it to perform worse than the case where no predictor is used [10]. Particularly, the modifications in IOPs are not likely to be predictable as they are made by the decision makers according to personal preferences, which makes existing prediction-based approaches ineffective in solving IOPs.

To solve an IOP, a naive approach is to iteratively rerun the optimizer once the objective function is modified. However, performing such iterative re-optimizations can be time-consuming, especially when the function evaluations are computationally expensive [11]. Therefore, it is of particular interest to investigate ways of optimizing IOPs efficiently by saving redundant re-optimizations. Since the objective function of an IOP is incrementally developed over several stages, intuitively, part of the results obtained at each stage should be reusable in future optimization attempts. This is due to the fact that incremental modifications may not completely change the original objective function, and such a special property is worth taking advantage of when solving IOPs.

From the experience-based optimization point of view [12], the historical optimization results can be considered as useful experience to guide the optimization of IOPs at a later design stage. The key issue in experience-based optimization is how to make use of historical information to guide future optimization, either implicitly or explicitly. For example, Sushil Louis borrowed the idea from the case-based reasoning, where the previously obtained solutions are injected into the search process to improve the performance on syntactically similar problems [13]. This can be seen as an implicit way to use historical information.

In IOPs, the dependence between the newly added decision variables and the previous ones, namely, the variable interactions, can be important information to guide future optimization. Despite that information such as variables' interaction pattern is not known *a priori*, fortunately, the variable interaction techniques allow us to explicitly extract valuable structural information about a black-box problem and turn it into a gray-box one [14], [15]. Therefore, in this work, we propose to solve the IOPs via cooperative coevolution (CC) [16], where the variable interaction information is considered as the experience acquired during the incremental optimization process.

The CC framework was initially proposed to decompose the decision vector of an optimization problem into a group of smaller components, thus breaking the original optimization problem into a set of simpler ones. Since the decision vector of a given problem can be partially nonseparable due to variable interactions, a major challenge of using the CC framework is to find a proper decomposition. One classic method is known as the random grouping [17]–[19], where the decision variables are randomly grouped into different groups, and thus the variable interactions are taken into consideration implicitly. By contrast, the other way is to detect the variable interactions in an explicit manner and form the groups accordingly. As a representative method of such, differential grouping (DG) (as well as its variants) can identify the nonseparable (i.e. interacting) components with high accuracy [20], [21]. As reported in the recent literature [22], such CC algorithms are promising in solving partially separable problems.

By adopting the CC framework coupled with variable grouping, we propose to incrementally detect the interactions between the incremental modifications and the objective function of an IOP, such that the decision variables of an IOP can be incrementally grouped and optimized. To be specific, the main contributions of this work are as follows.

1) Basic formulations of generic IOPs are presented. A generic IOP is defined as a single-objective optimization problem (SOP) with several design stages, where at each stage, the objective function is incrementally modified by introducing additional decision variables into the objective function. To describe the incremental modifications in IOPs, we present three different modification types by considering the interactions between added decision variables and those in the original problem.
2) A benchmark generator for IOPs is developed. In order to represent different modification types in IOPs using benchmark functions, we propose a method based on the Givens rotation for controlling the variable interactions. On the basis of the proposed benchmark generator, we further instantiate seven representative benchmark functions, which not only cover different modification types but also have various properties in terms of separability and modality.
3) A contribution-based cooperative coevolution (CBCC) [23] framework is proposed for solving IOPs. The proposed CBCC framework automatically detects the interactions between the newly added decision variables and those in the original problem, thus determining which decision variables of the modified problem should remain unchanged and which should be re-

optimized. By considering the contributions of the modified parts and the unchanged parts, the CBCC framework adaptively allocates the computational resources (i.e. fitness evaluations) to improve the optimization efficiency.

The remainder of the paper is organized as follows. Section II covers the required background, including the fundamental concepts relating to variable interaction, and brief introductions to cooperative coevolution and variable grouping. Section III presents the basic formulation of IOPs together with a benchmark function generator. Section IV elaborates a contribution-based cooperative coevolutionary (CBCC) framework for solving IOPs, and an incremental grouping (IG) algorithm for variable interaction analysis and problem decomposition. Section V presents the experimental results, where several representative benchmark functions are generated using the proposed generator, and the proposed CBCC method is assessed and compared with other representative algorithms on the benchmark functions as well as a real-world engineering problem. Finally, the paper is concluded in Section VI.

II. Background

An incrementally modified optimization problem (IOP) can be referred to as a single-objective optimization problem (SOP) involving multiple design stages, where there are incremental decision variables added to the objective function at each stage. Without loss of generality, an SOP[1] can be defined as:

$$\begin{aligned} \arg\min_{\mathbf{x}} \quad & f(\mathbf{x}), \\ \text{s.t.} \quad & \mathbf{x} \in \mathcal{X}, \end{aligned} \tag{1}$$

where $\mathcal{X} \subset \mathbb{R}^D$ is the decision space and $\mathbf{x} = (x_1, x_2, ..., x_D)^\top \in \mathcal{X}$ is the decision vector, and D is the number of decision variables.

In traditional single-objective optimization, a decision vector can be optimized as a whole if the dimension is not large, but when it comes to large-scale optimization which can involve hundreds or even thousands of decision variables, the decision vector is usually decomposed into a set of components to break a large-scale problem into a set of simpler subproblems [24], [25]. Since such variable grouping techniques can be also applied to solving IOPs, this section will present some related background knowledge, including the variable separability as well as cooperative coevolution.

A. Variable Interaction

A decision variable x_i is known as *separable* iff:

$$\arg\min_{\mathbf{x}} f(\mathbf{x}) = \Big(\arg\min_{x_i} f(\mathbf{x}), \arg\min_{\forall x_j, j \neq i} f(\mathbf{x})\Big), \tag{2}$$

which means that there does not exist any other decision variable interacting with x_i.

Based on the definition of separability, a problem $f(\mathbf{x})$ is known as *fully separable* iff:

$$\arg\min_{\mathbf{x}} f(\mathbf{x}) = \Big(\arg\min_{x_1} f(x_1, \dots), \dots, \arg\min_{x_D} f(\dots, x_D)\Big), \tag{3}$$

where there does not exist any interaction between any pair of decision variables in $\mathbf{x}$. By contrast, a problem $f(\mathbf{x})$ is known as *fully nonseparable* if every pair of decision variables interact with each other.

However, if only part of the decision variables are separable while the others are nonseparable, the problem is known as *partially nonseparable*:

$$\arg\min_{\mathbf{x}} f(\mathbf{x}) = \Big(\arg\min_{\mathbf{x}_1} f(\mathbf{x}_1, \dots), \dots, \arg\min_{\mathbf{x}_m} f(\dots, \mathbf{x}_m)\Big), \tag{4}$$

where $\mathbf{x}_1, \dots, \mathbf{x}_m$ are disjoint sub-vectors of $\mathbf{x}$, and $2 \leq m \leq D$. A problem is *partially additively separable* if:

$$f(\mathbf{x}) = \sum_{i=1}^{m} f_i(\mathbf{x}_i)\ , \tag{5}$$

where $\mathbf{x}_i$ are mutually exclusive decision vectors of f_i, and m is the number of independent components.

Algorithm 1: $(x^\star, f^\star)$ = CC(f)

```
/*Main Framework of CC*/
P ← randomized initial population;
c ← randomized initial context vector;
//grouping stage
G = Grouping(f);
//optimization stage
while Termination Condition is Not Satisfied do
    for κ = 1 to |G| do
        (P, c) = Optimizer(P, c, G_κ);
x* = c ; f* = f(x*) ;
return (x*, f*);
```

B. Cooperative Coevolution

As shown in Algorithm 1, the main framework of cooperative coevolution (CC) consists of two main stages: the grouping stage and the optimization stage. In the grouping stage, the whole decision vector is decomposed into several components. At the grouping stage, the decision variables are divided into a number of variable groups, where the grouping information is stored in $\mathcal{G}$. Ideally, each decision variable should only interact with other decision variables inside the same group, but not with those in any other group. Such groups are also known as the nonseparable (or interacting) groups. In order to determine the nonseparable groups, Omidvar *et al.* proposed the differential grouping (DG) method on the basis of the following theorem [20]:

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

$$\Delta_{\delta, x_p}[f](\mathbf{x})|_{x_p=a, x_q=b_1} \neq \Delta_{\delta, x_p}[f](\mathbf{x})|_{x_p=a, x_q=b_2}, \tag{6}$$

[1]This work only considers minimization problems with box constraints.

then x_p and x_q are nonseparable, where

$$\Delta_{\delta,x_p}[f](\mathbf{x}) = f(\dots, x_p + \delta, \dots) - f(\dots, x_p, \dots), \quad (7)$$

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

The quantities in (6) are real-valued numbers; therefore, the equality check cannot be evaluated exactly over the floating-point number field on computer systems. Consequently, the equality check needs to be converted to an inequality check by introducing a sensitivity parameter: $|\Delta^{(1)} - \Delta^{(2)}| > \epsilon$, to check if two variables are interacting. Here, $\Delta^{(1)}$ and $\Delta^{(2)}$ denote the left and right hand side of (6) respectively. In the absence of representation and roundoff errors, ϵ can be theoretically set to zero; however, this is not usually the case and the optimal value of ϵ is often a nonzero positive number. This parameterization makes DG sensitive to choices of ϵ whose optimal value may vary from function to function and difficult to tune by practitioners. To alleviate this problem, Omidvar *et al.* proposed DG2 [21], a parameter-free version of version of DG, which automatically sets ϵ by estimating the bounds on the computational roundoff errors to maximize the accuracy of variable interaction detection.

Using the variable grouping information, at the optimization stage, the decision variables in each nonseparable group (i.e. $\mathcal{G}_\kappa$) are iteratively optimized in a coevolutionary manner, where the optimizer can be any derivative-free single-objective optimization algorithm. Here the assumption is that the objective function is fixed, and the decomposition is valid throughout the optimization process. However, this is not the case with IOPs, which results in a total failure of classic CC with the existing decomposition methods. In this paper, we propose a modified version of differential grouping, termed the incremental grouping (IG), which checks the variable interaction pattern of the decision variables added at two neighboring stages during an incremental design and optimization process. This allows for a more informed optimization of the modified problem based on the solutions obtained prior to any modification of the objective function.

III. Problem Definition

A. Basic Formulations

An IOP can be formulated as a special SOP with T number of design stages:

$$F = (f^1(\mathbf{x}^1), ..., f^t(\mathbf{x}^t), ..., f^T(\mathbf{x}^T)), \quad (8)$$

where $t = 1, ..., T$, and $\mathbf{x}^t = (x_1, x_2, ..., x_{d^t})^\top \in \mathbb{R}^{d^t}$ is the decision vector having a size of d^t at design stage t. At each stage $t \geq 2$, since the objective function is modified by adding more decision variables, the size of $\mathbf{x}^t$ is increased as follows:

$$d^t = d^{t-1} + \Delta d^t, \quad (9)$$

where Δd^t is the number of added decision variables at design stage t.

As $\mathbf{x}^t$ is generated by adding more decision variables into $\mathbf{x}^{t-1}$, the objective function f^t is also modified on the basis of f^{t-1}. In practice, since different modifications can result in different interactions between the added decision variables (denoted as $\Delta\mathbf{x}^t = (x_{d^{t-1}+1}, x_{d^{t-1}+2}, ..., x_{d^t})^\top$) and the original decision variables $\mathbf{x}^{t-1}$, we consider the following three scenarios as possible modifications.

- *Modification Type I*: $\Delta\mathbf{x}^t$ do not interact with $\mathbf{x}^{t-1}$. In this scenario, since none of the decision variables in $\Delta\mathbf{x}^t$ interacts with those in $\mathbf{x}^{t-1}$, the incremental decision vector $\Delta\mathbf{x}^t$ can be optimized independently.
- *Modification Type II*: $\Delta\mathbf{x}^t$ partially interacts with $\mathbf{x}^{t-1}$. In this scenario, only some of the decision variables in $\Delta\mathbf{x}^t$ interact with those in $\mathbf{x}^{t-1}$ and the rest are independent.
- *Modification Type II*: $\Delta\mathbf{x}^t$ fully interact with $\mathbf{x}^{t-1}$. In this scenario, since each decision variable in $\Delta\mathbf{x}^t$ interacts with at least one decision variable in $\mathbf{x}^{t-1}$, $\Delta\mathbf{x}^t$ has to be optimized together with part of $\mathbf{x}^{t-1}$ inside the same variable groups.

Based on the above formulations, we further present how to design benchmark functions for IOPs in the next section.

B. Benchmark Function Generator

In benchmark function designs of SOPs, variable interaction (refer to Section II-A) is one of the most important characteristics to be taken into consideration. As suggested by the benchmark design principles in [25], while the separable functions can be simply generated by weighted aggregations of separable base functions (e.g. the Sphere function), the nonseparable functions should be generated by rotating the fitness landscapes of some special base functions (e.g. the Elliptic function).

Moreover, since the separability properties are also closely related to variable interactions, in order to cover the three different modification types presented above, we adopt the following benchmark function generator:

$$F = \begin{cases} f^1(\mathbf{x}^1) & = g(\mathbf{R}^1(\mathbf{x}^1 - \mathbf{o}^1)) \\ & \vdots \\ f^t(\mathbf{x}^t) & = g(\mathbf{R}^t(\mathbf{x}^t - \mathbf{o}^t)) \\ & \vdots \\ f^T(\mathbf{x}^T) & = g(\mathbf{R}^T(\mathbf{x}^T - \mathbf{o}^T)) \end{cases}, \quad (10)$$

where g is the base function scalable to the number of decision variables, $\mathbf{o}^t$ is the shift vector determining the location of the global optimum of the variables added at stage t, and $\mathbf{R}^t$ is a rotation matrix that causes variable interaction.

As an important property of IOPs, the objective function f^t ($t \geq 2$) at each design stage is modified on the basis of its previous version f^{t-1}. To reflect such a property, both $\mathbf{o}^t$ and $\mathbf{R}^t$ can be incrementally modified

using iterative functions. These modifications are such that the location of the current optimal solution stays the same while the interaction pattern of the variables may change. Therefore, $\mathbf{o}^t$ can be iteratively modified as:

$$\mathbf{o}^t = \begin{bmatrix} \mathbf{o}^{t-1} \\ \Delta\mathbf{o}^t \end{bmatrix}, \tag{11}$$

where $t = 1, ..., T$, $\Delta\mathbf{o}^t \in \mathbb{R}^{d^t}$ is the optimal position of the added decision variables. It is worth noting that in this study we do not change the position of the global optimum over time. Neither the benchmark nor the proposed framework limit the study of moving optima within an incremental optimization context. It is indeed possible to use the proposed framework for such a purpose; however, tracking optima is a matter of component optimizer rather than the framework itself. Therefore, to keep the focus of this study on the effect of adding variables on problem structure, we limit our study to problems with fixed optima. It should be noted that this does not necessarily mean a static landscape. Indeed the rotation of the landscape and the partial interaction of the newly added decision variables with the previous ones change the fitness of the previously obtained solutions. The focus of this study is to detect and respond to such changes rather than tracking optima.

In order to generate the rotation matrix $\mathbf{R}^t$ which determines the variable interactions, there are several basic requirements to be satisfied. First, as a rotation matrix, $\mathbf{R}^t$ must always be an orthogonal matrix with d_t column vectors. Second, as the number of d_t increases (refer to (9)), the size of $\mathbf{R}^t$ should also increase correspondingly. Third, considering the three modification types presented in Section III-A, $\mathbf{R}^t$ must be generated by considering the variable interaction between any pair of decision variables. To this end, we propose a method based on the *Givens rotation* [26] for the incremental modifications of $\mathbf{R}^t$.

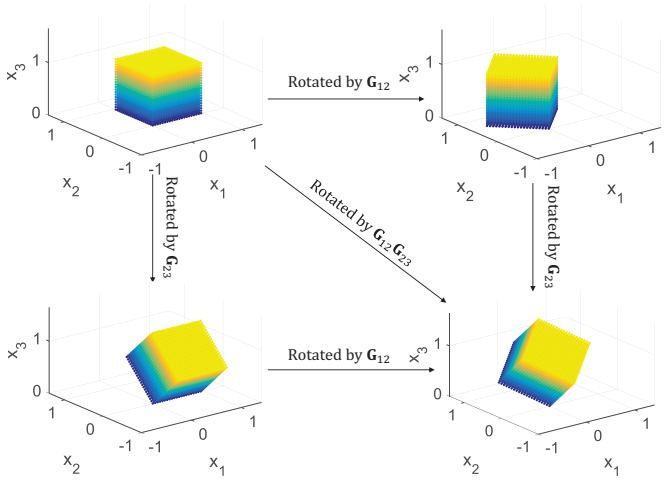


Fig. 2. An illustrative example of the Givens rotation, where $\mathbf{G}_{12} = G(1, 2, \frac{\pi}{4})$ and $\mathbf{G}_{23} = G(2, 3, \frac{\pi}{4})$ are two rotation matrices generated using Eq. (12) with $D = 3$.

Given two decision variables x_i and x_j, the x_i-x_j plane can be rotated by an angle of θ using the rotation matrix generated as:

$$G(i, j, \theta) = \begin{bmatrix} 1 & \cdots & 0 & \cdots & 0 & \cdots & 0 \\ \vdots & \ddots & \vdots & & \vdots & & \vdots \\ 0 & \cdots & \cos\theta & \cdots & -\sin\theta & \cdots & 0 \\ \vdots & & \vdots & \ddots & \vdots & & \vdots \\ 0 & \cdots & \sin\theta & \cdots & \cos\theta & \cdots & 0 \\ \vdots & & \vdots & & \vdots & \ddots & \vdots \\ 0 & \cdots & 0 & \cdots & 0 & \cdots & 1 \end{bmatrix}_{D\times D}, \tag{12}$$

where $\cos\theta$ and $\sin\theta$ are at the intersections of the ith and jth rows and columns of an D-dimensional identity matrix. As illustrated in Fig. 2, the Givens rotation is used to rotate the plane containing any two coordinate axes (i.e. decision variables).

Now that the Givens rotation allows us to rotate the plane containing any pair of coordinate axes, we can control the variable interactions by generating the rotation matrix correspondingly. Since the definition of IOPs mandates an increase in the dimensionality of the decision vector $\mathbf{x}^t$, the rotation matrix $\mathbf{R}^{t-1}$ must be extended by adding Δd^t additional dimensions so that its dimension conforms with that of $\mathbf{x}^t$:

$$\mathbf{R}_0^t = \begin{bmatrix} \mathbf{R}^{t-1} & \mathbf{0} \\ \mathbf{0} & \Delta\mathbf{I}^t \end{bmatrix}, \tag{13}$$

where $\Delta\mathbf{I}^t$ is a Δd^t-dimensional identity matrix, such that the original variable interactions reflected by $\mathbf{R}^{t-1}$ stay unchanged. Then, we perform a series of Givens rotations on $\mathbf{R}_0^t$ to generate the new rotation matrix $\mathbf{R}^t$:

$$\mathbf{R}^t = \prod_{p\in\mathcal{P}^t, q\in\mathcal{Q}^t} G(p, q, \theta_{p,q}) \times \mathbf{R}_0^t, \tag{14}$$

where $\mathcal{P}^t$ and $\mathcal{Q}^t$ contain the indices of each pair of interacting decision variables, and the interaction degree is controlled by the rotation angle $\theta_{p,q}$. In this way, $\mathbf{R}^t$ can be incrementally modified by specifying $\mathcal{P}^t$, $\mathcal{Q}^t$ and $\theta_{p,q}$ at each design stage t. To cover the three different modification types mentioned before, and to control the extent to which the new decision variables and the previous ones interact, the sets $\mathcal{P}^t$ and $\mathcal{Q}^t$ should be specified as follows:

$$\mathcal{P}^t \subset \{d^{t-1} + 1, d^{t-1} + 2, ..., d^t\}, \tag{15}$$

and

$$\mathcal{Q}^t \subset \{1, 2, ..., d^{t-1}\}, \tag{16}$$

with

$$|\mathcal{P}^t| = |\mathcal{Q}^t| = r^t \times \Delta d^t, \tag{17}$$

where $r \in [0, 1]$ is a parameter controlling the ratio of the new decision variables added at stage t that interact with the previous ones.

Using the generators above, the benchmark functions can be instantiated by specifying the following tuple:

$$(T, d^1, \Delta d^t, g, \theta_{p,q}, r^t), \tag{18}$$

where d^1 and Δd^t are the initial dimensionality and incremental dimensionality as defined in (9), g is the base function as defined in (10), $\theta_{p,q}$ is the rotation angle as defined in (14), and r^t is the interaction ratio as defined in (17). In Section V, we will instantiate some benchmark functions using the proposed generator.

IV. Method

The prime challenge of solving IOPs is the efficient handling of incremental modifications. A naive approach to dealing with such incremental modifications is to treat the modified problem at each design stage as a completely new one and opt for its re-optimization. However, this can discard any useful information contained within the previously obtained solutions to a previous design formulation. It is clear that the extent to which the previous solutions can be useful for the next formulation is dependent on the type of incremental modifications it entails. As was mentioned in the previous section, variable interaction is an important factor determining how the optimal value of each decision variable may change through the addition of new variables. Indeed, in this context, it is precisely the variable interaction pattern of the newly added design variables with the previous ones that cause the change. The aim of this section is to propose a method for exploiting variable interaction information for an efficient handling of incremental modifications.

In the absence of variable interaction information, it is difficult to devise an efficient strategy to deal with incremental modifications. The new variables are either completely independent of the previous ones, or interact with the previous ones in a partial or full manner (see Sec III). In a white-box problem full variable interaction pattern might be known; however, the complexity of the problem or lack of an algebraic form for the objective function can turn the problem into a black-box one. Variable interaction analysis allows us to extract valuable structural information about the problem and turn it into gray-box optimization [14], [15]. For this purpose, we propose a modified DG2 [21], termed the incremental grouping (IG) method, to suit IOPs. Further details about variable interaction analysis are given later in this section.

Once the variable interaction structure is inferred, it is natural to opt for a framework that facilitates the incorporation of such structural information. One such framework is CC [16] which allows an optimization problem to be broken down into its constituent components and be optimized cooperatively in a round-robin fashion. This property is of great importance for solving IOPs. CC not only allows us to preserve previous solutions obtained for an earlier design, but also allows for a seamless transition to the next design stage by treating new design variables as a new component. The round-robin coordination policy of CC allows the new and old components to be optimized collaboratively.

A major drawback of traditional CC for solving IOPs, however, is its suboptimal component selection policy (i.e., the round-robin strategy). This policy distributes the computational resources equally among components wasting a considerable amount of resources when components have nonuniform contributions towards improving the overall solution quality [27]. Although the issue of imbalance has been studied in the general context of single objective optimization [28]–[31], its effect is more pronounced when dealing with IOPs due to their unique features:

- Non-uniform dimensionality of components: In IOPs, the decision variables added at each design stage can be treated as a separate component. It is not uncommon to have a different number of decision variables added at each design stage. The variable interaction analysis may also result in components with different sizes depending on the underlying interaction pattern.
- Dynamics of the optimization process, and discrepancy among convergence behavior of components: First, due to the incremental nature of the design process in IOPs, the decision variables added at earlier design stages are often optimized longer. Depending on their variable interaction pattern, some of the decision variables belonging to earlier design stages may converge to their optima not requiring further optimization, thus leading to marginal contribution to the convergence of the objective function at later design stages. Second, the newly added decision variables often have the highest contribution, especially at the beginning of a design stage. Third, previously converged variables may interact with the ones added upon reformulation, which requires further optimization, thus leading to a transition from low to high contribution.

Contribution-based cooperative coevolution (CBCC) [23], [32] is an improved CC framework whose component selection policy is based on the contribution of components towards improving the overall solution quality, which makes CBCC a good fit for solving IOPs. It is clear that CBCC requires an estimation for the contribution of each component, but due to the nature of the problem, the actual contribution of each component is not directly observable. For a partially separable problem, it is possible to obtain a reliable estimation for the contribution of a component by recording its improvement after a round of optimization while keeping all other components constant. As was mentioned previously, in this paper we convert a black-box problem into a gray-box problem by means of variable interaction analysis, which allows us to minimize the inter-component dependence to obtain an accurate estimation of contributions.

The decomposition strategy that we devise in this paper is to analyze the interaction of the decision variables added at design stage t with respect to the previous components formed by gradual analysis of the objective

Algorithm 2: $\mathcal{G}^t = \texttt{IG}(f^t, \mathcal{G}^{t-1}, d^{t-1}, d^t)$

```
1 /*Pseudo Code of Incremental Grouping*/
2 Θ ← an uninitialized d^t × d^t interaction matrix;
3 initialize Θ^t_ij, ∀i, j ∈ {1, ..., d^{t-1}} using G^{t-1};
4 Θ_ij = 1, ∀i, j ∈ {d^{t-1} + 1, ..., d^t};
5 for i = d^{t-1} + 1 to d^t do
6     for j = 1 to d^{t-1} do
7         Θ^t_ij = InteractionDetection(f^t, i, j);
8 G^t = ConnectedComponents(Θ) ;
9 return G^t;
```

Algorithm 3: $(x^\star, f^\star) = \texttt{CBCC}(F, p_e)$

```
1  /*Main Framework of the proposed CBCC*/
2  t = 1;
3  G^t = {{1, ..., d^t}};
4  P^t ← randomized initial population;
5  c^t ← randomized initial context vector;
6  Δ^t ← component contributions initialized to ∞;
7  f^t_c = f^t(c^t);
8  while the termination criteria is not reached do
9      κ = ComponentSelector(Δ^t);
10     while κ is the best component according to Δ^t do
11         //optimization
12         (P^t, c^t) = Optimizer(P^t, c^t, G^t_κ);
13         //contribution calculation
14         f^t_p = f^t_c; f^t_c = f^t(c^t); Δ_κ = f^t_p − f^t_c;
15         if rand() < p_e then
16             Δ_i = ∞, ∀i ∈ {1, ..., |G|} ;
17         //new design stage
18         if F has switched to a new design stage then
19             t = t + 1;
20             G^t = IG(f^t, G^{t-1}, d^{t-1}, d^t);
21             P^t ← extending P^{t-1} to match d^t;
22             c^t ← extending c^{t-1} to match d^t;
23             Δ^t ← extending Δ^{t-1} to match |G^t|;
24 x^⋆ ← c^t ; f^⋆ ← f^t(x^⋆) ;
25 return (x^⋆, f^⋆);
```

function at all previous design stages, i.e., $1, \ldots, t-1$.

Algorithm 2 contains the details of the proposed incremental grouping (IG) method used in this paper to decompose IOPs. The algorithm takes as input f^t which is the objective function at design stage t, $\mathcal{G}^{t-1}$ the previous decomposition used at stage $t-1$, and the dimensionality of the objective function at design stages $t-1$ and t (i.e., d^{t-1} and d^t respectively). In other words, the purpose of this procedure is to analyze the variable interaction relationship of the decision variables ($\{d^{t-1}+1, \ldots, d^t\}$) with respect to all previous decision variables ($\{1, \ldots, d^{t-1}\}$) belonging to f^{t-1} and change the problem decomposition accordingly.

To be specific, Algorithm 2 works by constructing a binary variable interaction matrix $\mathbf{\Theta}$ which is then processed as an adjacency matrix to form all the independent variable components. First, $\mathbf{\Theta}$ is initialized to represent $\mathcal{G}^{t-1}$ which contains the decomposition prior to the design change. More specifically, Θ_{ij} is set to one if the ith and the jth decision variables belong to the same component in $\mathcal{G}^{t-1}$, and zero otherwise. If $\mathcal{G}^{t-1}$ is an empty set, then all entries of $\mathbf{\Theta}$ will be set to zero. On line 4, the entries of $\mathbf{\Theta}$ that belong to the newly added decision variables are all set to one due to the nonseparability assumption mentioned previously. Then, in the nested loops, the interaction of all newly added decision variables ($\{d^{t-1}+1, \ldots, d^t\}$) is checked against all previous decision variables ($\{1, \ldots, d^{t-1}\}$). This is done using the `InteractionDetection` function, which returns one if an interaction is detected and zero otherwise. The mechanism used in `InteractionDetection` is directly borrowed from the DG2 method [21], which is based on the differential grouping theorem (Theorem 1). Finally, the connected components are detected and returned by taking $\mathbf{\Theta}$ as an adjacency matrix and processed using `ConnectedComponents` which is a classic graph partitioning algorithm [33].

Algorithm 2 differs from its predecessor, namely, the DG2 method, in two major ways. First, it works progressively during the course of optimization and is invoked every time the problem formulation is changed by the inclusion of new decision variables to the objective function. Second, instead of performing a full pair-wise analysis of the decision variables, it only analyzes the new decision variables with respect to the previous ones, which makes it faster than DG2. To be exact, for transitions $t \in \{2, \cdots, T\}$, Algorithm 2 requires $\Delta d^t(d^{t-1}+1)$ evaluations whenever the design is changed by adding Δd^t new decision variables to the objective function. Additionally, it requires d^1 extra evaluations for the first transition ($t = 1 \rightarrow t = 2$).

As mentioned previously, the incremental grouping method denoted by Algorithm 2 is invoked repeatedly within the CBCC framework at the beginning of each new design stage to find how the newly added decision variables interact with the previous ones, such that the results obtained from the previous stages can be reused in the new stage. In this paper, we modify the framework in [32] as our proposed CBCC framework for solving IOPs, where the details are given in Algorithm 3. The proposed CBCC framework differs from its original version in the following ways. First, all its components (i.e. $\mathcal{G}^t$, $\mathbf{P}^t$, $\mathbf{c}^t$, $\mathbf{\Delta}^t$) are modified to be extendable to suit IOPs. Second, the component selection policy of the proposed algorithm is improved to maintain a better exploration/exploitation balance. Third, the use of p_e is modified such that when it is set to 1 it acts like canonical CC and when set to 0 it becomes a greedy algorithm. Finally, the IG method is iteratively invoked during the incremental optimization process. To be specific, the proposed CBCC framework takes as input the objective function F following the notation used in (8) and the parameter p_e which determines the extent to which the algorithm optimizes the most contributing component in a greedy manner (exploitation) or allows all components to get a chance to update their contribution (exploration).

On lines 2-7 of Algorithm 3, the framework is initialized. Similar to Algorithm 2, $\mathcal{G}^t$ on line 3 is a set of sets defining how the problem should be decomposed. In the first design stage ($t = 1$), the objective function (f^1) is assumed to be fully nonseparable; therefore, the only component that the algorithm begins with is $\{1, \ldots, d^1\}$, where d^1 is the dimensionality of f^1. The vector $\mathbf{\Delta}^t$ records the latest contribution of all components to the convergence of f^t at each design stage. Therefore, the size of $\mathbf{\Delta}^t$ is equal to the cardinality of $\mathcal{G}^t$. As can be seen, the contributions are initialized to infinity to guarantee that all components are optimized at least once. It is clear that the contributions will be updated in the main loop (lines 8-23) whenever a component is selected for optimization.

The CBCC framework works by finding the component with the largest contribution (κ). This is done using the `ComponentSelector` function which simply returns the index of the component with maximum contribution value to the convergence of f^t according to $\mathbf{\Delta}^t$. The selected component is optimized while its contribution is larger than all other components (line 10). The `Optimizer` function is any optimizer of choice which optimizes the component κ for a certain number of iterations. Once the component κ is optimized, its contribution is updated and set to the magnitude of improvement before and after optimization (lines 12-14). This exploitation strategy is broken with probability p_e to give all components a chance to be optimized, which is done by setting the contribution of all components to ∞.

Whenever the IOP is switched to a new design stage, the `IG` function is invoked to analyze the relationship between the new decision variables and the previous ones. Correspondingly, the dimensionality of $\mathbf{P}^t$, $\mathbf{c}^t$, and $\mathbf{\Delta}^t$ are updated to match the dimensionality of f^t and the most recent number of components. It is clear that the new decision variables added to each candidate solution are initialized within the respective bounds for each dimension, and the new decision variables on the context vector are also initialized in a similar manner. Finally, the dimensionality of $\mathbf{\Delta}^t$ is changed to match the new cardinality of $\mathcal{G}^t$ while preserving the contribution of previous components and initializing the new ones to ∞. The above procedure is repeated until the design process of the given IOP is finalized and the termination criteria are reached.

V. Experimental Study

In this section, we will conduct a series of experiments to assess the performance of the proposed CBCC framework for solving IOPs. First, we will instantiate some test functions using the benchmark function generator proposed in Section III-B. Second, we will perform some empirical comparisons using the instances of the benchmark functions. Finally, the proposed CBCC framework is further assessed using a real-world application, i.e., the design optimization of a stepped cantilever beam.

A. Benchmark Functions

As summarized in Table S-I in the supplementary document, we have instantiated seven benchmark functions using the benchmark generator as given in Section III-B. The four base functions are selected among the most commonly used ones in benchmark function designs [25], [34], [35]. More importantly, all of the four functions are originally separable but become nonseparable after landscape rotations [36], which meets the requirement as mentioned in Section III-B. For simplicity, the rotation angles $\theta_{p,q}$ are all randomized within $(0, \frac{\pi}{2})$, and the number of design phases T is set to 3 for all instances. The benchmark functions presented in Table S-I in the supplementary document cover the three modification types described in Section III-A, each exhibiting various modality and separability features. In what follows, we explain how these functions fit into the three modification types.

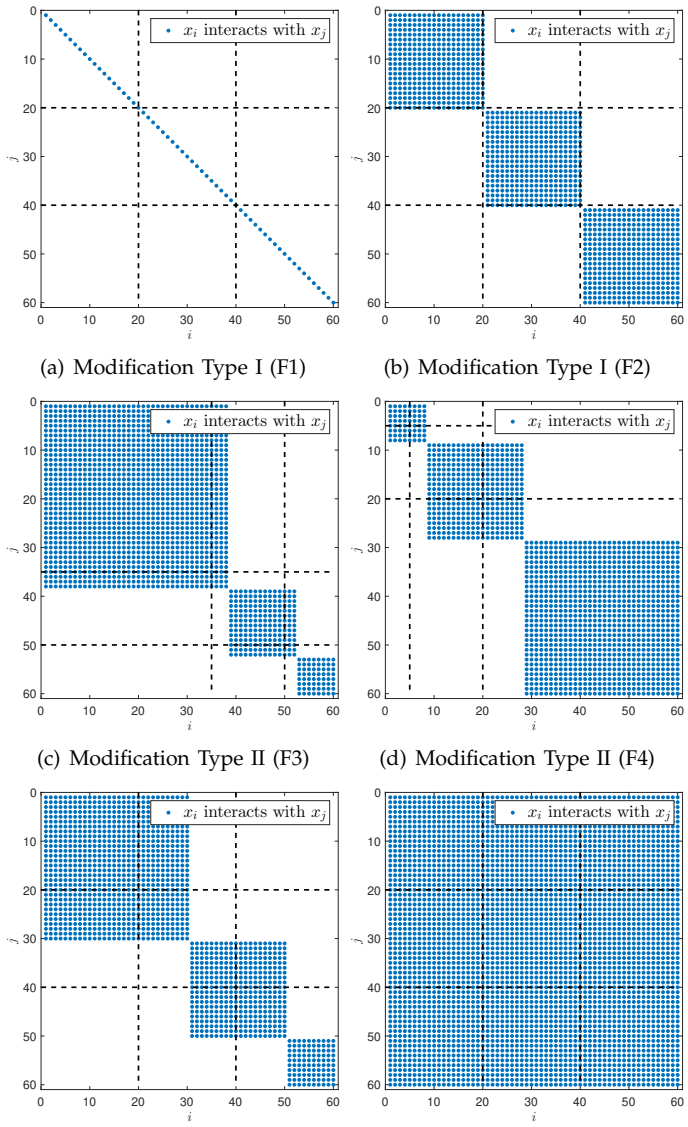
1) Modification Type I: For IOPs with Modification Type I (e.g. F1 and F2), the incremental modifications will not affect the variable interactions in the original problem. In this case, the optimum of the original problem is fully reusable at the new design stage. To be specific, we consider the following two scenarios:

- Fig. 3(a) shows a case where the new decision variables added at the latest design stage do not interact with each other or with any of the variables from previous stages. This means that each variable can be optimized independently;
- Fig. 3(b) shows a case where the decision variables added at the latest stage are fully nonseparable, while having no interaction with any of the variables from previous stages. This means that the new variables can be treated as a nonseparable component and be optimized independently of the other variables.

2) Modification Type II: For IOPs with Modification Type II (e.g. F1 to F6), the incremental modifications will partially affect the variable interactions in the original problem. In this case, the optimum of the original problem is only partially reusable at the new design stage. To be specific, we consider the following three scenarios:

- As shown in Fig. 3(c), the problem is initialized with a relatively large number of decision variables but modified with small increments;
- As shown in Fig. 3(d), the problem is initialized with a relatively small number of decision variables, but modified with big increments;
- As shown in Fig. 3(e), the problem is initialized and modified with even scales.

3) Modification Type III: As shown in Fig. 3(f), for IOPs with Modification Type III (e.g. F7), the incremental modifications will completely change the variable interactions in the original problem. In this case, the problem is fully changed and thus has to be completely re-optimized.



(a) Modification Type I (F1) (b) Modification Type I (F2)

(c) Modification Type II (F3) (d) Modification Type II (F4)

(e) Modification Type II (F5 or F6) (f) Modification Type III (F7)

Fig. 3. Variable interactions of IOPs with different modification types. Different design stages are marked by the dashed lines.

B. Empirical Comparisons

Using the benchmark functions as given in the previous subsection, this subsection presents some empirical comparisons to demonstrate the effectiveness of the proposed CBCC method. To begin with, we first introduce the methods in comparison. Then, the empirical comparisons mainly consist of two parts. First, we assess the general performance of CBCC on the benchmark functions given a specific number of fitness evaluations as the termination condition. Second, we further investigate the convergence speed of CBCC by setting specific accuracy levels.

1) Methods in comparison: In addition to the proposed CBCC method, we use three other methods to conduct the empirical comparisons, including the original CC method, an incremental method, and a naive method:

- The CC method: The classic CC framework as given by Algorithm 1 is adopted, where each incremental component is considered as an independent variable group. Here all components are optimized equally in a round-robin fashion.
- The incremental (INC) method: Each incremental component is optimized completely independently. Once the problem is modified, the method automatically switches to the new optimization stage and merely focuses on the newly added component.
- The naive method: As indicated by its name, this method simply treats the modified problem as a new one and performs a complete re-optimization of the modified problem.

In order to deal with the incrementally increased scales of IOPs, the decision vectors in each method are also incrementally increased. The component optimizer used by all methods mentioned above is SaNSDE [37]. For fair comparisons, the random seed of the random generator in SaNDE is set to the same value in each method, such that SaNDE performs the same stochastic behaviors. The population size of SaNDE is set to 50, and it is run for 50 iterations on any component upon selection. In practice, any other problem-specific optimizer is also applicable. For example, if the problem exhibits other forms of dynamism such as moving optima, dynamic optimization algorithms capable of tracking optima, such as multi-population methods [1], can be used as the component optimizer. For the proposed CBCC method, the parameter p_e is set to 0.2 in all the experiments, and the sensitivity analysis of p_e can be found in Section S-II in the supplementary document.

2) General Performance: To assess the general performance of the proposed CBCC method, we conduct some empirical comparisons with the CC method, the INC method, and the Naive method on the seven benchmark functions (F1 to F7). To obtain statistical results, each method is run for 31 independent times on each test function. For each run, we use a maximum number of $5000D$ fitness evaluations (FEs) as the final termination condition and assume that the decision-maker will make modifications on the given IOP once a number of $5000d_t$ FEs is reached at each stage. In this way, a fixed number of FEs is used by each method at each stage, and thus guaranteeing the fairness of the comparisons.

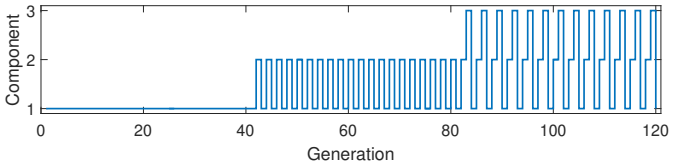
Table I[2] contains the experimental results for assessing the performance of CBCC in comparison with the three methods mentioned above. At the initial stage (i.e. Stage 1), the four methods have obtained exactly the same results on each test function. This is due to the fact that we have set the random seed of the random generator to the same value for the SaNSDE optimizer adopted in each method. However, as the test functions are incrementally modified at later stages (i.e. Stage 2 and Stage 3), the proposed CBCC method shows significantly better general performance than the other three methods.

[2]A complete table containing other descriptive statistics is given in Table S-III in the supplementary document.

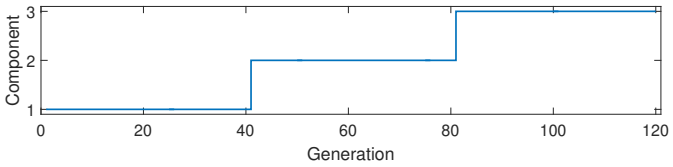
TABLE I
THE STATISTICAL RESULTS OF OPTIMIZATION ERRORS OBTAINED BY THE CBCC, CC, INCREMENTAL (INC) AND NAIVE METHODS ON THE BENCHMARK TEST FUNCTIONS. MEDIANS BASED ON 31 INDEPENDENT RUNS. BEST MEDIAN RESULTS ARE HIGHLIGHTED.

	Stage 1	Stage 2				Stage 3			
Fun.	All Algs.	CBCC	CC	INC	Naive	CBCC	CC	INC	Naive
F1	0.00e0	0.00e0	1.66e-18$^{\uparrow}$	0.00e0$^{\approx}$	6.98e-26$^{\uparrow}$	4.45e-28	3.51e-11$^{\uparrow}$	0.00e0$^{\downarrow}$	5.99e-15$^{\uparrow}$
F2	2.48e1	5.39e1	5.94e1$^{\uparrow}$	4.81e1$^{\downarrow}$	1.70e2$^{\uparrow}$	8.48e1	1.07e2$^{\uparrow}$	7.27e1$^{\downarrow}$	3.60e2$^{\uparrow}$
F3	1.69e5	2.25e4	6.26e4$^{\uparrow}$	7.11e4$^{\uparrow}$	6.68e4$^{\uparrow}$	7.38e3	2.65e4$^{\uparrow}$	5.61e4$^{\uparrow}$	8.26e4$^{\uparrow}$
F4	1.30e0	1.76e1	1.66e2$^{\uparrow}$	2.03e2$^{\uparrow}$	3.14e1$^{\uparrow}$	1.25e2	1.04e3$^{\uparrow}$	1.20e3$^{\uparrow}$	2.71e2$^{\uparrow}$
F5	3.49e4	5.42e3	7.17e4$^{\uparrow}$	2.09e4$^{\uparrow}$	1.94e4$^{\uparrow}$	1.77e0	3.01e4$^{\uparrow}$	1.86e4$^{\uparrow}$	1.45e3$^{\uparrow}$
F6	2.48e1	1.10e2	1.08e2$^{\approx}$	9.15e1$^{\downarrow}$	1.71e2$^{\uparrow}$	1.56e2	1.73e2$^{\uparrow}$	1.58e2$^{\approx}$	3.43e2$^{\uparrow}$
F7	1.37e0	3.14e1	3.14e1$^{\approx}$	4.13e3$^{\uparrow}$	3.14e1$^{\approx}$	5.12e1	5.22e1$^{\uparrow}$	3.03e3$^{\uparrow}$	5.58e1$^{\uparrow}$
w/t/l		—	5/2/0	4/1/2	6/1/0	—	7/0/0	4/1/2	7/0/0

The Wilcoxon signed-rank test at significance level of 5% is performed to compare the results obtained by the CBCC method and the others.
$^{\uparrow}$, $^{\approx}$and $^{\downarrow}$: CBCC performs statistically significantly better, equal and worse.
w/t/l: The number of results where CBCC statistically wins/ties/loses in the comparisons with other methods.
Hilighted entires are significantly better than the rest based on multiple comparison with Holm p-value correction.



(a) CC



(b) INC

Fig. 4. Components selected by CC and INC to undergo optimization at different stages of the incremental design process.

For F1 and F2, the proposed CBCC outperforms the CC and Naive methods, but is outperformed by the INC method. In this scenario, the decision variables in each incremental component can be naturally optimized independently, while the incremental grouping in CBCC will spend some extra function evaluations on variable interaction detections, thus reducing the total FEs used for optimization. By contrast, the INC method directly considers each incremental component to be independent, which is coincidentally consistent with the ideal grouping.

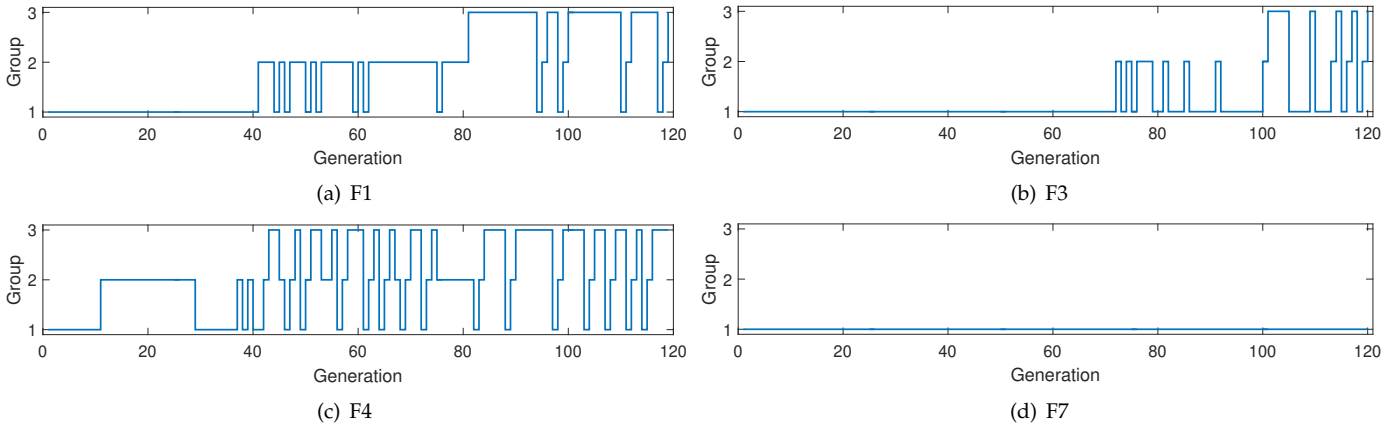
Given the fact that both CC and INC methods adopt the same grouping strategy, an interesting observation is that the INC method still significantly outperforms the CC method. On problems with Modification Type I (F1 and F2), the incremental components are of equal importance; therefore, the strategy adopted by INC is the most efficient due to an overall even allocation of FEs upon termination of the algorithm. CC, however, gives an equal chance to all components available at stage t, which results in an uneven resource allocation with a bias towards giving more resources to components belonging to earlier design stages. This observation can be further confirmed by Fig. 4, where the CC method iteratively switches between the three incremental components, but the INC method merely focuses on the incremental component at each stage.

For F3 to F6, the proposed CBCC method significantly outperforms the other three methods. The advantages of CBCC mainly lie in two aspects. First, CBCC performs incremental variable grouping to determine if the newly added decision variables interact with the previous ones, such that the nonseparable variable groups can be optimized in a cooperative manner. Second, CBCC performs adaptive allocation of the FEs to different nonseparable variable groups, such that the decision variables making more contributions to the convergence of the objective function (e.g. the newly added decision variables at each stage) will have a higher priority to be optimized. By contrast, none of the other three methods is able to handle the incremental modifications in F3 to F6 properly.

Finally, for the fully nonseparable function F7, the INC method is significantly outperformed by the other three methods. It is interesting to see that the performance of the INC method on F7 is exactly opposite to its performance on F1 and F2. This is due to the fact that the INC method consistently considers each incremental component to be independent, while F7 happens to be a fully nonseparable function. In fact, regardless of the variable interactions between the incremental components, the CC and INC methods always adopt the same grouping strategies on all of the seven test functions as demonstrated in Fig. 4. This indicates that the CC and INC methods lack robustness or generality when applied to different problems.

By contrast, as evidenced by Fig. 5, the proposed CBCC method shows robust performance on various types of problems by adaptively grouping the decision variables and allocating the FEs. To be specific, for fully separable problems, the CBCC method uniformly allocates the FEs to the optimization of three variable groups (Fig. 5(a)); for problems with a large initial component and small incremental components, the CBCC method mainly focuses on the first variable group (Fig. 5(b)); for problems with a small initial component and large incremental components, the CBCC method mainly focuses on the second and third variable groups (Fig. 5(c)); and for fully nonseparable problems (Fig. 5(d)), CBCC's performance is similar to that of the Naive method. It should be noted that CBCC has an advantage over the Naive method even on a fully nonseparable problem because it does not completely re-initialize the solutions obtained prior to a change, while the Naive method completely re-optimizes the problem after a change. This is also evident from Table I. Finally, the convergence profiles in Fig. S-1 in the supplementary document also indicate the robust performance of CBCC on various test functions.

It should be noted that CBCC uses the incremental



(a) F1 (b) F3 (c) F4 (d) F7

Fig. 5. Components selected by CBCC to undergo optimization at different stages of the incremental design process.

TABLE II
THE ACCURACY LEVELS FOR TEST FUNCTIONS F1 TO F7.

Function	F1	F2	F3	F4	F5	F6	F7
Accuracy Level	10^{-10}	10^{2}	10^{4}	10^{2}	10^{4}	10^{4}	10^{4}

grouping (IG) method presented in the previous section. Since the focus of this work is on analyzing the effect of an incremental design process, we limited our study to IG which analyzes the interaction of the newly added decision variables with respect to the previous ones from an earlier stage. One could easily opt to analyze the internal structure of a component at the expense of some additional computational cost. However, finding the optimal decomposition of a component is beyond the scope of this work. For the sake of completeness, we included some preliminary results in the supplementary material to compare the two strategies. The results in Table S-IV suggest that the two strategies perform statistically similar on small- to medium-scale problems with IG having a slight advantage. Given the computational advantage of IG over DG, we expect the gap to become wider on large-scale problems. In what follows, we will investigate the convergence speed of CBCC by conducting additional experiments.

3) Convergence Speed: In practice, the decision-maker may not make incremental modifications to an IOP until the optimization results are satisfactory at a certain design stage. Therefore, the convergence speed (in terms of FEs) can be an important criterion when solving IOPs. In order to assess the convergence speed of the proposed CBCC method, we conduct additional experiments as given next.

First, we define some accuracy levels for each test function as listed in Table II. We assume that, during the optimization process, it is automatically switched to the next stage once the accuracy level is reached. We still run each method for 31 independent times and record the success rate (SR) and computational cost with respect to the accuracy levels. In order to guarantee that there is sufficient FEs for each method to reach the accuracy levels, we set the maximum number of FEs to $50000d_t$ (10 times as large as what was used in the previous experiment) for each stage. If the maximum number of FEs is exceeded, the optimization process will automatically switch to the next stage.

TABLE III
THE MEAN RESULTS OF THE SUCCESS RATE (SR) AND COMPUTATIONAL COST OF THE CBCC, CC, INCREMENTAL (INC) AND NAIVE METHODS ON THE BENCHMARK TEST FUNCTIONS. BEST RESULTS ARE HIGHLIGHTED.

		Stage 1	Stage 2				Stage 3			
Fun.	Stats.	All Algs.	CBCC	CC	INC	Naive	CBCC	CC	INC	Naive
F1	Cost	3.31e+4	7.24e+4	9.75e+4	6.76e+4	8.35e+4	1.19e+5	1.94e+5	1.03e+5	1.62e+5
	SR	100%	100%	100%	100%	100%	100%	100%	100%	100%
F2	Cost	2.07e+4	9.28e+4	9.50e+4	1.60e+6	5.80e+5	2.01e+5	2.29e+5	3.00e+6	3.00e+6
	SR	100%	100%	100%	22%	83%	100%	100%	0%	0%
F3	Cost	7.15e+5	7.41e+5	7.56e+5	7.51e+5	8.91e+5	7.59e+5	7.92e+5	7.71e+5	1.01e+6
	SR	100%	100%	100%	100%	100%	100%	100%	100%	100%
F4	Cost	2.55e+3	1.02e+4	8.73e+5	1.00e+6	1.85e+4	2.53e+5	3.00e+6	3.00e+6	2.75e+6
	SR	100%	100%	12%	0%	100%	100%	0%	0%	12%
F5	Cost	2.84e+5	3.11e+5	4.00e+5	3.03e+5	4.06e+5	3.42e+5	4.99e+5	3.21e+5	4.80e+5
	SR	100%	100%	100%	100%	100%	100%	100%	100%	100%
F6	Cost	2.07e+4	1.40e+5	1.24e+5	1.82e+6	3.73e+5	4.56e+5	5.45e+5	3.00e+6	3.00e+6
	SR	100%	100%	100%	9%	96%	100%	93%	0%	0%
F7	Cost	7.39e+3	1.74e+4	2.14e+4	4.00e+5	2.13e+4	3.02e+4	4.61e+4	1.66e+6	4.12e+5
	SR	100%	100%	100%	80%	100%	100%	100%	45%	90%

SR: The success rate calculated by $SR = \frac{NSR}{NR}$, where NR denotes the total number of runs and NRS denotes the number successful runs having reached the predefined accuracy levels.
Cost: The minimum number of FEs cost by each method to reach the predefined accuracy levels.
Highlighted entires are significantly better based on a series of Wilcoxon signed-rank tests with Holm p-value adjustment.

As demonstrated by the results summarized in Table III, CBCC generally achieves the best overall performance in terms of convergence speed. At Stage 1, all four methods obtained the same results due to the same random seed used in the SaNSDE optimizer. At Stage 2 and Stage 3, the advantage of CBCC start to emerge, resulting in a high success rate and a lower computational cost. An interesting observation is that INC is also outperformed by CBCC on the multimodal function F2 in terms of the success rate. This is due to the fact that CBCC has a good chance to escape from local optima when switching between different

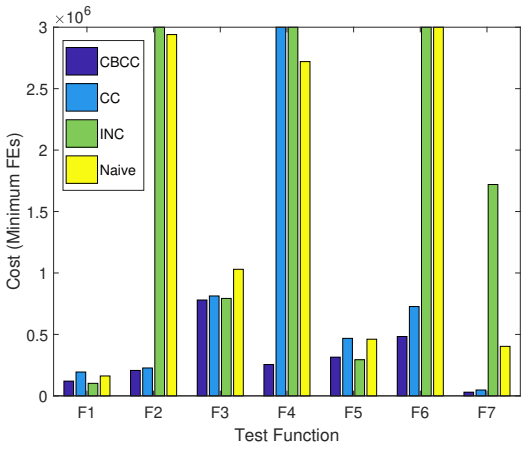


Fig. 6. Average computational costs needed by each method to reach the predefined accuracy levels at the end of the final optimization stage (Stage 3).

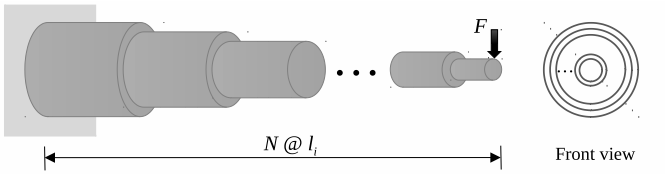
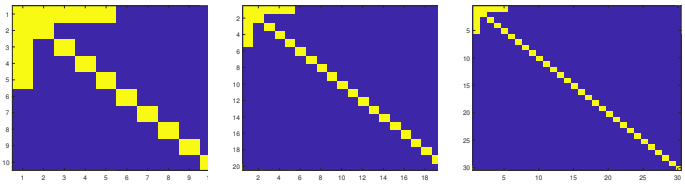


Fig. 7. Schematic of the stepped beam design problem with circular sections.



(a) 10D (b) 20D (c) 30D

Fig. 8. Variable interaction structures of the 10, 20 and 30-dimensional cantilever beam problems.

variable groups, while INC is unable to improve the solution quality once trapped in local optima, even if a large number of FEs are given. As further evidenced by Fig. 6, CBCC shows clear advantages over the other three methods on the difficult multimodal functions F2, F4, F6, and F7 by reaching the designated accuracy levels with fewer evaluations. By contrast, the four methods show similar convergence speed on unimodal functions such as F1, F3, and F5. This indicates the better potential of CBCC in dealing with complicated problems with limited computational costs.

C. Design Optimization of a Stepped Cantilever Beam

In this section, we investigate the efficacy of the proposed algorithm on a structural engineering problem. The case study is about the design of a stepped cantilever beam [38], which has applications in aeronautical engineering for rotor shaft design, in mechanical engineering for designing robot arms, and in civil engineering for beam design. The objective is to minimize the weight of the beam such that it is capable of bearing a concentrated load at its end while satisfying a set of mechanical constraints. The cantilever beam comprises a set of segments each having a variable cross-section area defined by a radius (r_i) and its length (l_i) (Fig. 7). This is a scalable design problem whose granularity can be adjusted by allowing the total number of segments to change in an iterative design process. The objective function of this problem is defined as follows:

$$\min f(\mathbf{r},\mathbf{l}) = \rho \sum_{i=1}^{n} l_i \pi r_i^2, \tag{19}$$

where ρ is the density, l_i is the length of the ith segment, and r_i is the radius of the ith cross-section area.

This problem must satisfy a series of stress constraints such that the bending stress in each beam segment (σ_i) must be less than a predefined allowable stress (σ_a). Therefore, the solutions should also satisfy the following n nonlinear constraints:

$$g_i(r_i) = \frac{\sigma_i}{\sigma_a} - 1 \leq 0, \tag{20}$$

where,

$$\sigma_i = \frac{M_i r_i}{I_i}, \tag{21}$$

$$M_i = F(L + (1-i)l_i), \tag{22}$$

$$I_i = \pi \frac{r_i^4}{4}, \tag{23}$$

and L is the total length of the beam, F the concentrated force at its end, and I_i the moment of inertia of the ith segment. To satisfy the stress constraints, we used a penalty function approach to penalize the infeasible solutions.

In addition to the above, the physics of the problem also mandates that the segments closer to the fixed end have a larger cross-section, i.e., $r_1 \geq r_2 \geq \cdots \geq r_n$ (see Fig. 7). This is due to the fabrication condition and the fact that a more distant load causes a larger bending stress. To satisfy this condition, the objective is reformulated such that the radius of each segment, with the exception of the first segment, is defined with respect to its adjacent segment closer to the restraining wall.

$$r_i = r_1 \prod_{j=1}^{i-1} p_j, \text{ for } i \in \{2,\ldots,n\}, \tag{24}$$

where $0 < r_1 \leq 30$, and $0 < p_j \leq 1$. For this particular experiment, we set $L = 500$ (cm), $\sigma_a = 14000$ (N/cm^2), and the concentrated force $F = 50000$ (N). It is also customary in structural design to assume $l_1 = \cdots = l_n = \frac{L}{n}$. Therefore, the final objective function will have the following form $f(r_1, p_1, \ldots, p_{n-1})$.

To compare the performance of the algorithms, the cantilever beam problem is solved incrementally starting with 10 segments in the first design stage. The design

TABLE IV
PERFORMANCE OF CBCC, CC, INC, AND NAIVE ALGORITHMS ON A CANTILEVER BEAM WITH CIRCULAR CROSS-SECTIONS.

Fun.	Stats.	CBCC	CC	INC	Naive
Stage 1	Median	1.94e+05	1.94e+05	1.94e+05	1.94e+05
	Mean	1.95e+05	1.95e+05	1.95e+05	1.95e+05
	StDev	5.10e+03	5.10e+03	5.10e+03	5.10e+03
	Feasible (%)	100%	100%	100%	100%
Stage 2	Median	2.47e+05	3.84e+05	5.30e+09	1.80e+13
	Mean	2.86e+05	3.94e+05	5.30e+09	1.52e+13
	StDev	8.66e+04	6.76e+04	1.36e+09	7.24e+12
	Feasible (%)	100%	100%	0%	16%
Stage 3	Median	4.62e+05	5.69e+05	2.86e+10	2.54e+13
	Mean	4.83e+05	5.64e+05	2.80e+10	2.52e+13
	StDev	9.57e+04	6.01e+04	6.75e+09	1.78e+12
	Feasible (%)	100%	100%	0%	0%

Highlighted entires are significantly better based on a series of Wilcoxon signed-rank tests with Holm p-value adjustment.
The values are based on the penalized values of the objective function.

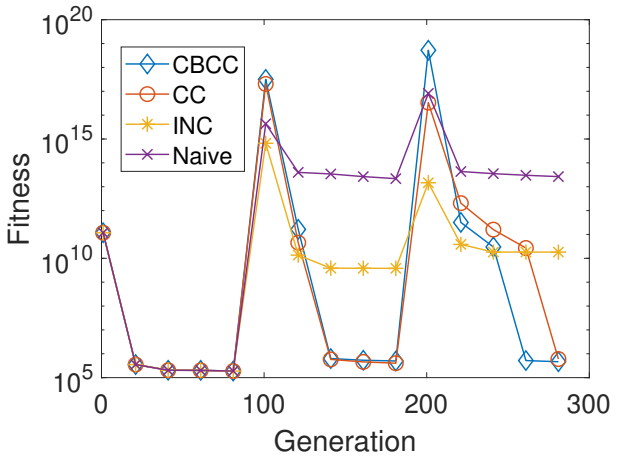


Fig. 9. The convergence profile of each method applied to the real-world application problem.

is subsequently refined by increasing the number of segments to 20 and 30 in the second and the third stages respectively. In other words, $d^1 = 10, d^2 = 20$, and $d^3 = 30$. The variable interaction analysis using DG2 shows that the cantilever beam has a partially separable objective function with the first 5 variables having interactions, while all other variables are detected to be separable (Fig. 8). Therefore, the final decomposition for CBCC and CC is as follows $\mathcal{G}^1 = \{\{1, \ldots, 10\}\}, \mathcal{G}^2 = \{\{1, \ldots, 10\}, \{11, \ldots, 20\}\}, \mathcal{G}^3 = \{\{1, \ldots, 10\}, \{11, \ldots, 20\}, \{21, \ldots, 30\}\}$. Similar to other experiments in this paper, the population size is set to 50. To test if the compared methods are able to perform fast convergence, the maximum number of objective function evaluations is set to $500D$, and the increments occur at every $500d^t$ evaluations.

The experimental results for the CBCC, CC, INC, and Naive algorithms are given in Table IV. The results are based on 31 independent runs and the highlighted entries are statistically significant using a series of Wilcoxon signed-rank tests with Holm p-value correction based on a 95% confidence interval. As can be seen, CBCC outperforms all other algorithms at the end of the second and the third stages.

The table also contains the percentage of feasible solutions for each algorithm. The cantilever beam problem has a very small feasible region. A Monte Carlo simulation using $1e9$ sample points results in only 318 feasible solutions to be sampled randomly for a 10-dimensional version of the problem. The number of randomly sampled feasible solutions for the 20- and 30-dimensional cases drops to zero. The results in Table IV clearly shows the benefit of using an incremental approach for solving such highly constrained problem. As can be seen, all algorithms have a 100% success rate at the first stage. However, at the second and third stages, the performance of INC and Naive algorithms deteriorate significantly resulting in almost no feasible solution being found by the two algorithms. By contrast, CBCC and CC have a 100% success rate across all stages. Although INC also optimizes the problem in an incremental manner, its greedy nature does not allow the previous components to be optimized after an increment.

As indicated by Fig 9, CBCC and CC show similar convergence speed; by contrast, INC and Naive completely fail to converge at the second and third stages.

VI. CONCLUSIONS

In this work, we have investigated how to efficiently solve a specific type of IOPs with increasing decision variables. First, we have represented some basic formulations of IOPs. Then, on the basis of the basic formulations, we designed a benchmark function generator for generating benchmark functions of IOPs. By taking the special property of IOPs, we have proposed a contribution based cooperative coevolution (CBCC) method. As part of the CBCC method, an incremental grouping (IG) method has also been proposed to deal with the incremental modifications.

As experimental studies, we assessed both the benchmark function generator and the proposed CBCC method. To cover different types of incremental modifications as given in the basic formulations of IOPs, we generated seven benchmark functions using the proposed generator. Using the seven benchmark functions, we have also conducted some experimental comparisons between the proposed CBCC method and three other typical methods. Our experimental results have demonstrated the robust performance of the proposed CBCC method. Furthermore, the performance of the proposed CBCC method has also been assessed on a real-world application.

As pointed out in the introduction, this work falls into the general scope of the experience-based optimization [12], where the motivation is to use the experience acquired in historical design stages to guide future optimization. As a specific instance, we have considered the variable interaction information as the important experience in solving IOPs, and we have

adopted the CC framework as the solver to make use of such experiences. In the future, we would like to further improve the efficiency of the proposed CBCC framework and the IG method in terms of the cost of function evaluations, which can be particularly meaningful when dealing with computationally expensive IOPs. It is also worth noting that the IOPs studied in this work is just one of the many possible scenarios where incremental modifications are made. Real-world problems could also involve incrementally added constraints, and solving such problems may require specially tailored constraint handling methods.

REFERENCES

[1] T. T. Nguyen, S. Yang, and J. Branke, "Evolutionary dynamic optimization: A survey of the state of the art," *Swarm and Evolutionary Computation*, vol. 6, pp. 1–24, 2012.

[2] S. Yang, Y.-S. Ong, and Y. Jin, Eds., *Evolutionary computation in dynamic and uncertain environments*, ser. Studies in Computational Intelligence. Springer, 2007, vol. 51.

[3] P. Bosman, "Learning and anticipation in online dynamic optimization," *Evolutionary computation in dynamic and uncertain environments*, pp. 129–152, 2007.

[4] I. Y. Kim and O. De Weck, "Variable chromosome length genetic algorithm for progressive refinement in topology optimization," *Structural and Multidisciplinary Optimization*, vol. 29, no. 6, pp. 445–456, 2005.

[5] M. Olhofer, Y. Jin, and B. Sendhoff, "Adaptive encoding for aerodynamic shape optimization using evolution strategies," in *Proc. of IEEE Congress on Evolutionary Computation*, vol. 1. IEEE, 2001, pp. 576–583.

[6] Y. Jin, M. Olhofer, and B. Sendhoff, "On evolutionary optimization of large problems using small populations," in *International Conference on Natural Computation*. Springer, 2005, pp. 1145–1154.

[7] J. Branke and D. C. Mattfeld, "Anticipation and flexibility in dynamic scheduling," *International Journal of Production Research*, vol. 43, no. 15, pp. 3103–3129, 2005.

[8] T. T. Nguyen and X. Yao, "Dynamic time-linkage problems revisited," in *Workshops on Applications of Evolutionary Computation*. Springer, 2009, pp. 735–744.

[9] P. A. Bosman, "Learning, anticipation and time-deception in evolutionary online dynamic optimization," in *Proceedings of the annual workshop on Genetic and evolutionary computation*. ACM, 2005, pp. 39–47.

[10] T. T. Nguyen, Z. Yang, and S. Bonsall, "Dynamic time-linkage problems-the challenges," in *Proceedings of the IEEE RIVF International Conference on Computing and Communication Technologies, Research, Innovation, and Vision for the Future*. IEEE, 2012, pp. 1–6.

[11] Y. Jin, "Surrogate-assisted evolutionary computation: Recent advances and future challenges," *Swarm and Evolutionary Computation*, vol. 1, no. 2, pp. 61–70, 2011.

[12] S. Liu, K. Tang, and X. Yao, "Experience-based optimization: A coevolutionary approach," *CoRR*, vol. abs/1703.09865, 2017. [Online]. Available: http://arxiv.org/abs/1703.09865

[13] S. J. Louis, "Case injected genetic algorithms for learning across problems," *Engineering Optimization*, vol. 36, no. 2, pp. 237–247, 2004.

[14] L. D. Whitley, F. Chicano, and B. W. Goldman, "Gray box optimization for Mk landscapes (NK landscapes and MAX-kSAT)," *Evolutionary computation*, vol. 24, no. 3, pp. 491–519, 2016.

[15] R. Santana, "Gray-box optimization and factorized distribution algorithms: where two worlds collide," *arXiv preprint arXiv:1707.03093*, 2017.

[16] M. A. Potter and K. A. De Jong, "A cooperative coevolutionary approach to function optimization," in *Proc. of International Conference on Parallel Problem Solving from Nature*, vol. 2, 1994, pp. 249–257.

[17] Z. Yang, K. Tang, and X. Yao, "Multilevel cooperative coevolution for large scale optimization," in *Proc. of IEEE Congress on Evolutionary Computation*, June 2008, pp. 1663–1670.

[18] M. N. Omidvar, X. Li, Z. Yang, and X. Yao, "Cooperative coevolution for large scale optimization through more frequent random grouping," in *Proc. of IEEE Congress on Evolutionary Computation*, 2010, pp. 1754–1761.

[19] X. Li and X. Yao, "Cooperatively coevolving particle swarms for large scale optimization," *IEEE Transactions on Evolutionary Computation*, vol. 16, no. 2, pp. 210–224, April 2012.

[20] M. N. Omidvar, X. Li, Y. Mei, and X. Yao, "Cooperative coevolution with differential grouping for large scale optimization," *IEEE Transactions on Evolutionary Computation*, vol. PP, no. 99, pp. 1–1, 2013.

[21] 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. PP, no. PP, p. (accepted on 30/3/2017), 2017.

[22] 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 (TOMS)*, vol. 42, no. 2, p. 13, 2016.

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

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

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

[26] E. Anderson, "Discontinuous plane rotations and the symmetric eigenvalue problem," 2000.

[27] B. Kazimipour, M. N. Omidvar, X. Li, and A. K. Qin, "A sensitivity analysis of contribution-based cooperative co-evolutionary algorithms," in *IEEE Congress on Evolutionary Computation*. IEEE, 2015, pp. 417–424.

[28] 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.

[29] S. Mahdavi, S. Rahnamayan, and M. E. Shiri, "Incremental cooperative coevolution for large-scale global optimization," *Soft Computing*, pp. 1–20, 2016.

[30] ——, "Cooperative co-evolution with sensitivity analysis-based budget assignment strategy for large-scale global optimization," *Applied Intelligence*, pp. 1–26, 2017.

[31] ——, "Multilevel framework for large-scale global optimization," *Soft Computing*, vol. 21, no. 14, pp. 4111–4140, 2017.

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

[33] J. E. Hopcroft and R. E. Tarjan, "Efficient algorithms for graph manipulation," Stanford University, Stanford, CA, USA, Tech. Rep. STAN-CS-71-207, 1971.

[34] P. N. Suganthan, N. Hansen, J. J. Liang, K. Deb, Y.-P. Chen, A. Auger, and S. Tiwari, "Problem definitions and evaluation criteria for the cec 2005 special session on real-parameter optimization," *KanGAL report*, vol. 2005005, p. 2005, 2005.

[35] C. Li, S. Yang, and D. A. Pelta, "Benchmark generator for the ieee wcci-2012 competition on evolutionary computation for dynamic optimization problems," Brunel University, UK, Tech. Rep., 2011.

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

[37] Z. Yang, K. Tang, and X. Yao, "Self-adaptive differential evolution with neighborhood search," in *Proc. of IEEE Congress on Evolutionary Computation*, 2008, pp. 1110–1116.

[38] G. N. Vanderplaats, "Very large scale optimization," National Aeronautics and Space Administration (NASA), Langley Research Center, Colorado, US, Tech. Rep. NASA/CR-2002-211768, 2002.