Graph-Based Deep Decomposition for Overlapping Large-Scale Optimization Problems

Xin Zhang, *Member, IEEE*, Bo-Wen Ding, Xin-Xin Xu, Jian-Yu Li, *Student Member, IEEE*, Zhi-Hui Zhan, *Senior Member, IEEE*, Pengjiang Qian, *Senior Member, IEEE*, Wei Fang, *Member, IEEE*, Kuei-Kuei Lai, and Jun Zhang, *Fellow, IEEE*

***Abstract*—Decomposition methods play a critical role in cooperative co-evolutionary algorithms (CCEAs) for solving large-scale optimization problems. Although some well-performing decomposition methods have been designed based on the interactions among variables (IaV), their grouping accuracy is still limited due to the poor performance on the overlapping problems and the computational roundoff errors of IaV in the implementation. To deal with these limitations, a graph-based deep decomposition (GDD) method is proposed to obtain more accurate grouping results, especially for the overlapping problems. On the one hand, the GDD mines the IaV information and obtains the minimum vertex separator of the interaction graph of variables, so as to group variables deeply and recursively. On the other hand, the GDD has the ability of fault tolerance to deal with the computational roundoff errors of IaV and can improve the grouping accuracy. For better experimental studies of overlapping problems, a novel overlapping function generator is designed with the random and complicate overlap type, and two new metrics are proposed to evaluate the grouping accuracy. Comprehensive experiments show that GDD can greatly improve the grouping accuracy and help CCEAs perform better than other existing algorithms, especially on the overlapping problems. In addition, the GDD is highly fault tolerant and can divide problems accurately even on the inaccurate IaV.**

***Index Terms*—Cooperative co-evolutionary algorithms (CCEAs), decomposition methods, evolutionary computation, large-scale optimization problems (LSOPs).**

Manuscript received 19 July 2022; revised 6 September 2022; accepted 28 September 2022. Date of publication 20 October 2022; date of current version 17 March 2023. This work was supported in part by the National Key Research and Development Program of China under Grant 2019YFB2102102; in part by the National Natural Science Foundations of China under Grant 62106088, Grant 62172192, Grant 62176094, Grant 62073155, and Grant 61873097; in part by the Key-Area Research and Development of Guangdong Province under Grant 2020B010166002; in part by the Guangdong Natural Science Foundation Research Team under Grant 2018B030312003; in part by the High Level Personnel Project of Jiangsu Province under Grant JSSCBS20210852; and in part by the National Research Foundation of Korea under Grant NRF-2021H1D3A2A01082705. This article was recommended by Associate Editor J. Liang. *(Corresponding authors: Zhi-Hui Zhan; Kuei-Kuei Lai.)*

Xin Zhang, Bo-Wen Ding, Pengjiang Qian, and Wei Fang are with the School of Artificial Intelligence and Computer Science and the Jiangsu Key Laboratory of Media Design and Software Technology, Jiangnan University, Wuxi 214122, China.

Xin-Xin Xu is with the School of Computer Science and Technology, Ocean University of China, Qingdao 266100, China.

Jian-Yu Li and Zhi-Hui Zhan are with the School of Computer Science and Engineering, South China University of Technology, Guangzhou 510006, China, also with Pazhou Laboratory, Guangzhou 510330, China, and also with the Guangdong Provincial Key Laboratory of Computational Intelligence and Cyberspace Information, Guangzhou 510006, China (e-mail: zhanapollo@163.com).

Kuei-Kuei Lai is with the Department of Business Administration, Chaoyang University of Technology, Taichung City 413, Taiwan (e-mail: laikk.tw@gmail.com).

Jun Zhang is with Zhejiang Normal University, Jinhua 321004, China, and also with Hanyang University, Ansan 15588, South Korea.

This article has supplementary downloadable material available at https://doi.org/10.1109/TSMC.2022.3212045, provided by the authors.

Digital Object Identifier 10.1109/TSMC.2022.3212045

I. Introduction

With data growing explosively, large-scale optimization problems (LSOPs) have aroused increasing attention and become a hot research topic in many systems engineering fields [1], [2], [3], such as the multiobjective optimization of large-scale capacitated arc routing problems [4], the constrained optimization of the large-scale power system [5], and the large-scale optimization of the supply chain system [6], [7]. Compared with the traditional optimization problems, a much larger number of decision variables need to be optimized in the LSOPs. In this case, the increase of the problem size causes it difficult to obtain the global optimal solution within the limited computation resources, e.g., within the maximal number of fitness evaluations [8], [9], [10].

Since evolutionary computation algorithms (including evolutionary algorithms (EAs) [11], [12] and swarm intelligence algorithms [13], [14]) are skilled in maintaining the solution diversity and finding optimal solutions in global optimization [15], [16], [17], [18], [19], [20], [21], they have been widely studied and extensively applied to solve the LSOPs [22], [23], [24]. EAs used for the LSOPs can be classified into two main categories according to whether decompose the problems. The first category of EAs takes the LSOP as a whole and does the main work on the design of effective learning strategies. These learning strategies are proposed to improve the solution diversity and find the optimal solutions [25], [26], [27], [28], [29].

The second category of EAs, usually called cooperative co-evolutionary algorithms (CCEAs), decomposes the LSOPs into several subproblems and then solves subproblems by different subpopulations [30], [31]. The decomposition of LSOP can reduce the search space of each subproblem and improve the search efficiency of CCEAs [32]. The most important part of CCEAs is the decomposition method that aims to put the interacting variables of the problem into the same group and divide the noninteractive variables into different groups. Each group is corresponding to a subproblem. An excellent decomposition method can divide variables accurately and, therefore, improve the optimization performance of CCEAs for solving the LSOPs. Two main kinds of decomposition methods are widely used: 1) random grouping [33] and 2) differential grouping (DG) [34], [35], [36], [37], [38], [39], [40], [41], [42]. Normally, the grouping accuracy of DG is usually higher than random grouping [34], because the DG methods calculate the interaction among variables (IaV) of the LSOPs to help group the variables.

However, due to the complexity of the LSOPs and the computational roundoff errors of IaV, it is still difficult for the existing DG methods to decompose the LSOPs effectively, especially for the overlapping LSOPs. For example, in the complex overlapping LSOPs, there are overlaps among the ideal groups where these groups and their overlaps are called overlapping groups and overlapping variables, respectively (referred to Section II-A for definition). For example, in an overlapping function $f(\boldsymbol{X}) = (x_1+x_2+x_4)^2+(x_1+x_3+x_5)^2$ and $\boldsymbol{X} = (x_1, x_2, x_3, x_4, x_5)^T$, there are two overlapping groups $(x_1, x_2, x_4)^T$ and $(x_1, x_3, x_5)^T$ and an overlapping variable x_1 in the ideal case. The two groups actually can be treated separately as two overlapping groups, but the existing DG methods will group all the variables of the problem into a larger group (i.e., $\boldsymbol{X} = (x_1, x_2, x_3, x_4, x_5)^T$), which deteriorates hampering the optimization efficiency of CCEAs for the LSOPs [38]. This is due to that the DG methods do not identify the overlapping variables and treat the overlapping groups as a whole group. Besides, the grouping accuracy of the existing DG methods is still limited by the accuracy of IaV. That is, IaV may sometimes be inaccurate (compared with the ideal IaV) due to the computational roundoff errors, e.g., grouping some noninteracting variables together. As a result, the inaccurate IaV of these DG methods will inevitably lead to the inaccurate grouping results.

Therefore, to decompose overlapping LSOPs and deal with the inaccurate IaV, this article proposes a graph-based deep decomposition (GDD) method, which is crucial for obtaining more accurate groups to enhance the performance of CCEAs. GDD is inspired by the graph cut [43] that can divide a complicated graph into small subgraphs via the minimum vertex separator (MVS), since an LSOP with IaV can be regarded as a graph. On the one hand, the GDD uses IaV to deeply and recursively decompose the overlapping groups, where three rules are designed to help to determine the recursive decomposition. On the other hand, due to the decomposition ability on overlapping LSOPs, the GDD has the ability of fault tolerance for dealing with the computational roundoff errors and improving the final grouping accuracy.

In the GDD, IaV is first obtained by the existing DG methods, and a graph can be constructed according to IaV. Then, the connected components of the graph are corresponding to the initial groups of the problem. Since some of these initial groups may have overlaps, a recursive overlapping group decomposition (ROGD) method is proposed to divide overlapping groups based on the MVS of the corresponding graph. In ROGD, three rules are designed to help to determine the recursive decomposition, including how to deal with MVS (the first two rules) and isolated variables to construct the complete groups (the third rule). Each group is regarded as a graph, and the max-flow algorithm is used to obtain the MVS [44]. If the group belongs to an overlapping group, the MVS of the corresponding graph is equivalent to the overlapping variables, and the group can be decomposed by the MVS; otherwise, the MVS of this group is empty, and the group cannot be decomposed. Finally, the overlapping groups with too small sizes will be merged to save the fitness evaluations, and the final groups are obtained for the CCEAs.

GDD can not only decompose overlapping LSOPs but also has the ability of fault tolerance. Concretely, in a nonoverlapping LSOP, if independent variables in two groups are wrongly judged as interacting variables caused by the computational roundoff errors of IaV, the two groups will be regarded as a group in some existing DG methods [36], [37], which is not efficient for the CCEA optimization. However, such mistaken grouped variables can be easily separated by the overlapping variables (the misjudged interacting variables) in GDD. For example, in $f(\boldsymbol{X}) = (x_1 + x_2 + x_4)^2 + (x_3 + x_5)^2$ and $\boldsymbol{X} = (x_1, x_2, x_3, x_4, x_5)^T$, there are two groups $(x_1, x_2, x_4)^T$ and $(x_3, x_5)^T$ in the ideal case. However, if x_1 and x_3 are wrongly judged to be interactive in IaV, only a group $(x_1, x_2, x_3, x_4, x_5)^T$ will be obtained in some DG methods. Differently, in GDD, x_1 can be regarded as an overlapping variable, and the final groups are $(x_1, x_2, x_4)^T$ and $(x_1, x_3, x_5)^T$, which are closer to the ideal case. Moreover, GDD is used after getting IaV and does not consume more fitness evaluations. The contributions of this article are presented as follows.

1) The GDD method is proposed to deeply and recursively decompose LSOPs via MVS, and three rules are designed for helping the recursive decomposition. This decomposition method can obtain more accurate grouping results, which is significant to improve the optimization efficiency of CCEAs on LSOPs, especially for the overlapping problems.
2) Due to decomposition ability on overlapping LSOPs, GDD is fault tolerant and can obtain the higher grouping accuracy on LSOPs.
3) A novel overlapping function generator is proposed with the random and complex overlap type. It can be used as a routine for generating different kinds of overlapping LSOP to test the grouping efficiency of decomposition algorithms. This is significant for further researches into overlapping LSOPs in the community.
4) Two new metrics are designed to evaluate the grouping accuracy, including the overlapping rate and the redundancy rate of the grouping results obtained by the decomposition algorithms versus the ideal grouping results.

The remainder of this article is organized as follows. Section II introduces the existing decomposition methods for the LSOPs, the overlapping problems, and the MVS of the graph. Section III describes the details of the proposed GDD method. Section IV presents the experiments, including the analysis of the grouping accuracy and the fault-tolerance ability of GDD and the optimization efficiency of CCEAs combined with GDD. Finally, Section V gives a conclusion.

II. Backgrounds

A. Overlapping Problems

In the overlapping problems, there are some subcomponents with overlap, but these subcomponents can be separated after dealing with the overlap. The overlapping problem can be defined as follows.

Definition 1: If $\arg\min f(\boldsymbol{X}) = (\arg\min f(\boldsymbol{X}_1, \ldots), \ldots, \arg\min f(\ldots, \boldsymbol{X}_k))$ and some of $\boldsymbol{X}_i$ share variables, $f(\boldsymbol{X})$ is an overlapping function and is separable with k nonseparable groups, where $\boldsymbol{X} = (x_1, \ldots, x_D)^T$ is a decision vector with D dimensions, and $\boldsymbol{X}_1$ to $\boldsymbol{X}_k$ are subvectors of $\boldsymbol{X}$. If $\boldsymbol{X}_i$ and $\boldsymbol{X}_j$ overlap, they are denoted as overlapping groups, and the shared variables of them are denoted as overlapping variables.

For example, the ideal groups of the overlapping problems f_{13} and f_{14} in IEEE Congress on Evolutionary Computation (IEEE CEC) 2013 are pairwise joint with the chain type, as shown in [45]. Therefore, the overlapping variables are the connections between the overlapping groups. In the existing grouping methods, such as DG [34], extended DG (XDG) [35], global DG (GDG) [36], DG2 [37], and recursive DG (RDG) [38], all the variables in the f_{13} or f_{14} will be regarded as interactive and will be merged in a larger group. In fact, the larger group can be efficiently separated by further dealing with the overlapping variables, although it is difficult to obtain the overlapping variables. Therefore, the GDD method is proposed in this article to identify the overlapping variables.

B. Decomposition Methods

The decomposition methods divide the variables of LSOPs into several groups. Let $f(\boldsymbol{X})$ denote the objective function of the LSOP. The separability of $f(\boldsymbol{X})$ can be defined as follows.

Definition 2 [33]: If $\arg\min f(\boldsymbol{X}) = (\arg\min f(\boldsymbol{X}_1, \ldots), \ldots, \arg\min f(\ldots, \boldsymbol{X}_k))$, $f(\boldsymbol{X})$ is separable with k nonseparable groups, where $\boldsymbol{X} = (x_1, \ldots, x_D)^T$ is a decision vector with D dimensions, and $\boldsymbol{X}_1$ to $\boldsymbol{X}_k$ are pairwise disjoint subvectors of $\boldsymbol{X}$.

The interaction exists only in variables of the same group. However, it is difficult to identify IaV and group the variables accurately, although various decomposition methods have been proposed, such as random grouping [33], [46] and DG [34], [35], [36], [37], [38], [39], [40]. Random grouping updates groups in every iteration of CCEAs. It ignores IaV and attempts different random grouping strategies during the evolutionary process. Therefore, random grouping may result in the low grouping accuracy and influences the solving efficiency of CCEAs [46].

Different from random grouping, DG identifies IaV by calculating the difference of corresponding objective function values, but it only considers a part of direct variable interactions [34]. For example, if x_1 is interactive with x_2, x_2 will be directly assigned to the same group of x_1, and other interactive variables of x_2 will not be detected. There are two shortcomings of DG: the incompleteness of IaV and the ignorance of computational errors (e.g., the roundoff errors of the floating-point operations). Therefore, the XDG is proposed to identify indirect variable interactions and to get more IaV information than DG [35]. Afterward, the GDG takes the computational errors into consideration, and takes the LSOP as a graph to decompose the problem and obtain the complete IaV [36]. Furthermore, DG2 (an improved variant of GDG) groups variables in a more accurate way by setting a reliable threshold value, and has a higher calculation efficiency than GDG [37]. However, these grouping methods cannot solve the overlapping problems where vectors from $\boldsymbol{X}_1$ to $\boldsymbol{X}_k$ are not pairwise disjoint in Definition 2. Although GDG and DG2 get the complete IaV information, they will merge two groups that have overlapping variables. Therefore, the number of variables in the group may be still too large due to the mergence, resulting in the difficulty of the optimization of the LSOPs.

As the above variants of DG methods need a number of fitness evaluations to obtain the IaV information, the RDG is proposed to detect IaV by the binary search with less computation cost [38]. Moreover, RDG2 considers the computational errors and improves the grouping accuracy of RDG [39]. Based on RDG2, the RDG3 is designed to decompose the overlapping functions [40]. During the decomposition, if the size of a group is large, RDG3 will forcedly divide the variables of this group into different smaller groups. In this way, RDG3 can decompose the overlapping functions. However, the forced decomposition may also divide the interactive variables into different groups, which will break the independence among groups and affect the optimization efficiency of CCEAs. Besides, as RDG, RDG2, and RDG3 do not identify the interactions between each pair of variables, they cannot obtain the complete IaV.

C. MVS of the Graph

For a connected graph G, the removal of its MVS will disconnect G, and the size of its MVS is equal to the vertex connectivity of G which is denoted as $\kappa(G)$ [43], [47]. Let $\boldsymbol{V}$ and $\boldsymbol{E}$ be the vertex set and the edge set of the graph $G(\boldsymbol{V}, \boldsymbol{E})$, respectively. If there exist $k(1 \le k \le |\boldsymbol{V}|)$ vertices whose removal makes G unconnected, and the removal of arbitrary $(k–1)$ vertices does not disconnect G, then $\kappa(G)$ is equal to k and the set of the k vertices is recorded as the MVS of G. Fig. S-1 of the supplementary material shows an example of the MVS of a graph.

As the vertices and the edges of the graph can be corresponding to the vertices, the edges, and the weights of edges of a network, it has been proved that the graph can be transformed to a network, and the calculation of the vertex connectivity can be transformed into the max-flow problem in the network [44]. Before calculating the maximum flow, the graph G needs to be transformed into the network N. To be more specific, as described in [44], each vertex $v \in \boldsymbol{V}$ in the graph G is corresponding to two vertices v' and v'' and an edge (v', v'') with the weight of 1 in the network N. Each edge $(u, v) \in \boldsymbol{E}$ in the graph G is corresponded with two edges (u'', v') and (v'', u') with the weight of infinity (∞) in the network N. The corresponding network N of the graph G in Fig. S-1(a) is shown in Fig. S-2 of the supplementary material. In this way, the vertex connectivity $\kappa(G)$ is equal to the maximum flow of N. This

Algorithm 1: ($Group$, sep) = GDD($\mathbf{\Theta}$, D)

```
Begin
1  $Group = \phi$;
2  $sep = \phi$;
3  $CNC$ = ConnComp($\mathbf{\Theta}$);
4  If $CNC$ has separable variables Then
5      $sep$ = {all separable variables in $CNC$};
6      Remove $sep$ from $CNC$;
7  If $CNC \neq \phi$ Then // divide overlapping groups
8      $Group$ = ROGD($\mathbf{\Theta}$, $CNC$, $D$);
9      $Group$ = Adjust($Group$, $D$);
End
```

article uses the Dinic algorithm [48] to solve the max-flow problem. Then, the MVS can be obtained by traversing the residual network [49].

III. Graph-Based Deep Decomposition Method

The decomposition method that groups variables according to IaV is an effective approach for CCEAs to solve the LSOPs [34], [35], [36], [37], [38], [39], [40]. IaV can be represented by a (0, 1)-matrix $\mathbf{\Theta}$ [36], [37]. If $\mathbf{\Theta}(i, j) = 1$, where i and j are two variables, it represents that i is interactive with j; otherwise, it represents that i and j are independent. Based on the IaV matrix $\mathbf{\Theta}$, a graph can be obtained. Each variable of the problem is corresponding to a vertex in the graph. If two variables are interactive, there will be an edge connecting the corresponding two vertices; otherwise, there will be no connection between them. The nonseparable (i.e., interactive) groups are equivalent to the connected components of the graph [36], [37]. Separable variables are equivalent to the groups with a vertex.

In this section, the GDD method is proposed, inspired by the idea of the graph cut. The proposed method mines the interactive information among variables obtained by the DG methods to deeply divide variables more accurately, especially for overlapping problems.

Algorithm 1 shows the pseudocode of the GDD method. The input $\mathbf{\Theta}$ is obtained by the existing DG methods, and D is the dimension of the problem. The output $Group$ is the set of the final nonseparable groups, and sep is the set of separable variables. The function ConnComp is used to obtain the set of connected components CNC in line 3. After removing sep from CNC, if CNC is not empty, the ROGD algorithm is carried out for dividing the overlapping groups (nonseparable groups with overlapping variables) efficiently. After the ROGD, an adjustment strategy (Adjust) is carried out to merge too small groups that have overlapping variables.

A. Recursive Overlapping Group Decomposition

The ROGD algorithm aims to divide overlapping groups (connected components) into smaller groups. It can improve the decomposition efficiency of the overlapping LSOPs and help CCEAs to solve problems more effectively.

If each connected component (a nonseparable group) is regarded as a graph, it can be divided into several components after the removal of MVS (described in Section II-C). In overlapping groups, MVS can be regarded as the overlapping variables. Let $\boldsymbol{S}$ denote the vertex set of the MVS of a nonseparable group G, and let $\boldsymbol{U}$ denote the graph after removing $\boldsymbol{S}$ from G. If $\boldsymbol{U}$ can be separated into two connected components ($\boldsymbol{U}'$ and $\boldsymbol{U}''$), as shown in Fig. S-3 of the supplementary material, G can be divided into two groups $\{\boldsymbol{U}' \cup \boldsymbol{S}\}$ and $\{\boldsymbol{U}' \cup \boldsymbol{S}\}$ with overlapping vertices (i.e., s_1 and s_2).

However, after removing the MVS, if the size of the new connected components (i.e., $\boldsymbol{U}'$ and $\boldsymbol{U}''$) is still too large, these components need to be further divided. Therefore, the ROGD algorithm is designed in a recursive way to divide an overlapping group into smaller groups with appropriate sizes. In the ROGD, the termination of the algorithm and the completeness of grouping (the complete groups include all variables of the problem) are described as follows.

1) Termination of ROGD: For a group G, if the number of its vertices $|\boldsymbol{V}|$ is too small or the size of its MVS ($|\boldsymbol{S}|$) is too large, G will not be separated any more. Therefore, in the proposed ROGD algorithm, if $|\boldsymbol{V}| \leq D/\alpha$ or $|\boldsymbol{S}| \geq |\boldsymbol{V}|/\beta$, where D is the dimension of the problem, the recursive decomposition of the group G will terminate.

2) Completeness of Grouping: To describe the ROGD algorithm clearly, the decomposition process is regarded as the tree structure. Each tree node represents a group or a separable variable, and leaf nodes represent groups that are not divided any more. It should be noted that a tree node is also corresponding to a graph.

For the completeness of grouping, not only the intergroup independence but also the intragroup interaction should be satisfied. That is, for the variable v in the group G ($v \notin$ the overlapping components), all its interactive variables should be added into G. Therefore, a complete group should include the MVS of its ancestor nodes (denoted as MVS_anc), because the MVS_anc is interactive with some variables of this group. In this way, the groups are also independent after ignoring the overlapping variables. For example, the decomposition of the graph G in Fig. S-3 is shown in Fig. S-4 of the supplementary material. The nodes with shadow represent the leaf nodes. $\boldsymbol{U}'$ and $\boldsymbol{U}'$ are two independent leaf nodes, and the complete groups are $\{\boldsymbol{U}' \cup \boldsymbol{S}\}$ and $\{\boldsymbol{U}'' \cup \boldsymbol{S}\}$. Fig. 1 shows an example of the decomposition process of the ROGD algorithm. S_i represents the MVS of the graph G_i. As shown in the figure, $\{7, 8\}$ and $\{6\}$ are chosen as the MVS of G_1 and G_3, respectively. The decomposition of f_{14} in IEEE CEC 2013 [45] is also given in Section S-II of the supplementary material as an example.

There are two main parts of the recursive decomposition. The first part is to use the breadth-first search (BFS) [50] to implement the recursive decomposition. A group will be divided recursively, until it satisfies the termination condition of ROGD. For example, in Fig. 1(b), G_1 is divided into G_2, G_3, and G_4. G_2 and G_4 do not need to be divided any more, but G_3 is further divided into G_5 and G_6 by removing S_3. G_2, G_4, G_5, and G_6 are the leaf nodes. The second part is to add the MVS to groups for the completeness of grouping. Herein, three points should be taken into consideration, and the three corresponding rules are designed as follows.

1) First point is which MVS to be added.

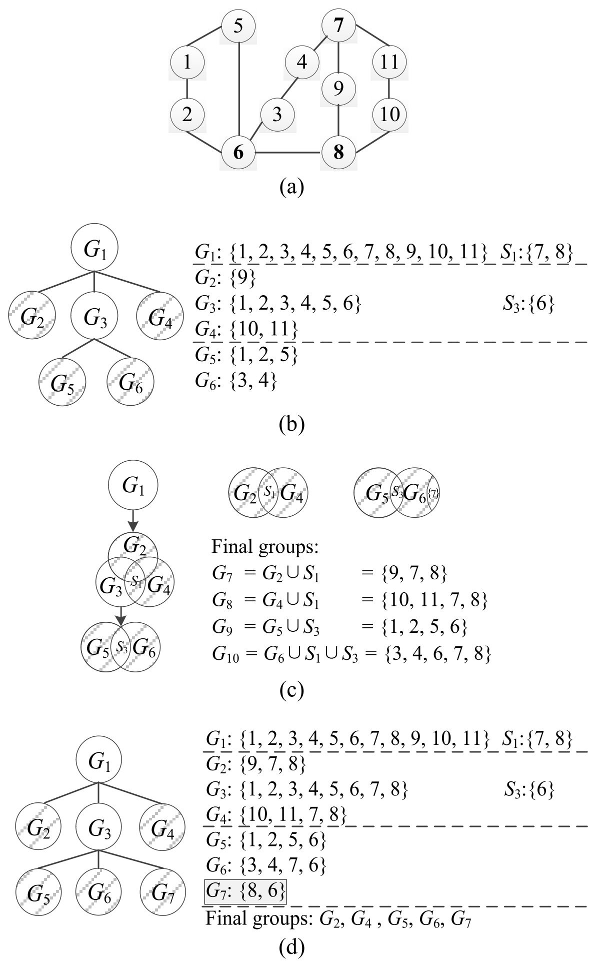

(a)
G1: {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11} S1:{7, 8}
G2: {9}
G3: {1, 2, 3, 4, 5, 6} S3:{6}
G4: {10, 11}
G5: {1, 2, 5}
G6: {3, 4}
(b)
Final groups:
G7 = G2∪S1 = {9, 7, 8}
G8 = G4∪S1 = {10, 11, 7, 8}
G9 = G5∪S3 = {1, 2, 5, 6}
G10 = G6∪S1∪S3 = {3, 4, 6, 7, 8}
(c)
G1: {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11} S1:{7, 8}
G2: {9, 7, 8}
G3: {1, 2, 3, 4, 5, 6, 7, 8} S3:{6}
G4: {10, 11, 7, 8}
G5: {1, 2, 5, 6}
G6: {3, 4, 7, 6}
G7: {8, 6}
Final groups: G2, G4, G5, G6, G7
(d)

Fig. 1. Example of the decomposition of an overlapping group. (a) Graph G_1. (b) Tree structure of the decomposition of G_1 (without adding MVS during the decomposition). (c) Illustration of the final groups of (b) (adding MVS after the whole decomposition). (d) Tree structure of the decomposition of G_1 (adding MVS during the decomposition).

Rule 1: A node (group) G only adds the MVS_anc that interacts with its vertices $\boldsymbol{V}$.

Explanation: This rule is to avoid blindly adding all MVS_anc into G. There may be two cases. The first case is that the MVS of its father node is certainly be added, since this MVS is the minimum set of vertices to separate its father node and is certainly interactive with $\boldsymbol{V}$. The second case is that the MVS of its grandfather nodes or more old ancestor nodes will be added if and only if the MVS interacts with $\boldsymbol{V}$. For example, in Fig. 1(c), G_5 and G_6 add the MVS (i.e., S_3) of their father node (i.e., G_3) to form the final groups. However, for the node G, not all the MVS_anc is interactive with $\boldsymbol{V}$, and only the interactive MVS_anc with $\boldsymbol{V}$ is added into G. For example, for G_5 in Fig. 1(b), only S_3 ({6}, the MVS of its father node G_3) is interactive with the vertices (2 and 5) in G_5. Therefore, only S_3 is added into G_5, but S_1 (the MVS of its grandfather node G_1) is not added (as S_1 has no interaction with G_5).

2) Second point is when to add the MVS of the group G into nodes to construct complete groups, such as after dividing G at the first time or after finding all the leaf nodes of G.

Algorithm 2: *Tnode* = InitTnode(g)

```
  Begin
1 $Tnode.v$ = Variable set of $g$;
2 $Tnode.ovp = \phi$;
3 $Tnode.parent = -1$; // if $Tnode.parent = -1$, it is the root node
  End
```

Rule 2: The MVS of the group G is added after the whole decomposition to construct the final groups.

Explanation: If the MVS is added during the decomposition of G, it will be involved in the following decomposition of G, which may increase the repetition of the MVS. However, if the MVS is added after the whole decomposition, it will be only involved in the leaf nodes of G, which helps to accelerate the decomposition process. The depth-first search (DFS) [51] is used to find the MVS_anc of a node. For a leaf node, all MVSs of its ancestor nodes are traversed and judged whether to be added or not. Fig. 1(b) and (c) shows the example of adding the MVS after finding all the leaf nodes. After removing the MVS of G_1 (S_1), G_1 can be decomposed into G_2, G_3, and G_4, and G_2 and G_4 are leaf nodes. After removing the MVS of G_3 (S_3), G_3 can be divided into two leaf nodes G_5 and G_6. For the leaf nodes G_5, the MVS_anc is $\{S_1 \cup S_3\}$, and only the interactive MVS_anc (S_3) is added into G_5 to form the final group G_9 in Fig. 1(c). Similarly, S_1 is added into G_2 and G_4 to form G_7 and G_8, respectively, and S_1 and S_3 are added into G_6 to form G_{10}. The final groups in Fig. 1(c) include G_7, G_8, G_9, and G_{10}.

Fig. 1(d) shows the example of adding the MVS after dividing the group for the first time. Different with the decomposition in Fig. 1(b) and (c), after removing S_1 and dividing G_1, S_1 is added into the decomposed parts to form the nodes G_2, G_3, and G_4. Similarly, after removing S_3 and dividing G_3, S_3 is added into the decomposed parts to form the nodes G_5, G_6, and G_7. The final groups in Fig. 1(d) include G_2, G_4, G_5, G_6, and G_7. G_7 is redundant in the decomposition in Fig. 1(d), since it includes the vertices 6 and 8 which both have appeared in other final groups (G_2, G_4, G_5, and G_6).

3) Third point is how to deal with the isolated vertices after removing the MVS of the group G, such as the vertex D in Fig. S-1 of the supplementary material.

Rule 3: All the isolated vertices are taken as an isolated group after adding the MVS.

Explanation: For example, in Fig. S-1 of the supplementary material, vertex D is merged with the MVS of the group $G(\{C\})$ to form an isolated group $\{C \cup D\}$.

To construct the tree structure of ROGD, a tree node (*Tnode*) which represents a group g should include the variable set of g (v), the MVS (overlapping variables) of g (ovp), its parent node (*parent*). The initialization of a tree node based on group g is shown in Algorithm 2.

The DFS_MVSanc algorithm is to find the interactive MVS_anc with the current traversing node, as shown in Algorithm 3. For the input parameters, *TNode* is the set of tree nodes. *Tnode* is the current traversing node. $\boldsymbol{\Theta}$ is gotten from the DG methods. For the output parameters, *MVS* includes the MVS_anc that is interactive with the variables of *Tnode*. All

Algorithm 3: MVS = DFS_MVSanc($TNode$, $Tnode$, Θ)

```
Begin
1  $MVS = \phi$;
2  $p = Tnode.parent$; // the current traversing ancestor node
3  While $p \neq -1$ Do // $TNode_p$ is not the root node
4      If there are variables in $Tnode.v$ is interactive with variables in $TNode_p.ovp$ Then // Rule 1
5          $MVS = MVS \cup TNode_p.ovp$;
6      $p = TNode_p.parent$; // obtain the next ancestor node of $Tnode$
End
```

Algorithm 4: $Group$ = ROGD(Θ, CNC, D)

```
Begin
1  $Group = \phi$;
2  $TNode = \phi$;     // set of tree nodes
3  $queNode = \phi$;   //queue of indexes of tree nodes traversed by BFS
4  $r = 1$;
5  For all groups $g \in CNC$ Do
6      $Tnode$ = InitTnode($g$);
7      $TNode = TNode \cup \{Tnode\}$;
8      $queNode = queNode \cup \{r\}$; $r = r + 1$;
9  $h = 1$;   // $h$ is the index of the current traversing $queNode$
10 While $h<r$ Do        // traverse the tree nodes by BFS
11     $t = queNode_h$; $h = h + 1$;
12     $Tnode = TNode_t$;       // the current traversing node
13     $Tnode.ovp$ = getMVS($Tnode.v$);
14     If $|Tnode.v| \leq D/\alpha$ or $|Tnode.ovp| < 1$ or $|Tnode.ovp| \geq |Tnode.v|/\beta$ Then // $Tnode$ is a leaf node (termination of ROGD, Section III-A)
15         If $Tnode.parent \neq -1$ Then       // not the root node
16             $MVS$ = DFS_MVSanc($TNode$, $Tnode$, $\Theta$);
17             $Tnode.v = MVS \cup Tnode.v$;              //Rule 2
18         $Group = Group \cup \{Tnode.v\}$;
           // a leaf node is a final group
19     Else       // Tnode needs to be divided again
20         Remove $Tnode.ovp$ from $Tnode.v$;
21         $Children$=ConnComp($Tnode.v$);
22         If $Children$ has separable variables $seps$ Then
           // $seps$ is saved as a set in $Children$, Rule 3
23             $node$ = InitTnode($Tnode.sep$); $node.parent = t$;
24             $MVS$ = DFS_MVSanc($TNode$, $node$, $\Theta$);
25             $Group = Group \cup \{MVS \cup node.v\}$;
26             Remove $seps$ from $Children$;
27         For all connected components $cnc \in Children$ Do
28             $node$ = InitTnode($cnc$); $node.parent = t$;
29             $TNode = TNode \cup \{node\}$;
30             $queNode = queNode \cup \{r\}$; $r = r + 1$;
End
```

ancestor nodes of *Tnode* are traversed by DFS in lines 3 to 6. If there are variables of the *Tnode.v* that are interactive with variables in $TNode_p.ovp$, $TNode_p.ovp$ are added into *MVS* in lines 4 and 5.

3) Complete ROGD: The details of ROGD are shown in Algorithm 4, where *CNC* is the set of connected components (the initial groups) calculated from the IaV matrix Θ, and D is the dimension of the problem, and *Group* is the set of final groups. The function getMVS in line 13 obtains the MVS by the Dinic algorithm [49] and the traversal of the residual network (as shown in Section II-C).

TNode records the set of tree nodes, and *queNode* records the queue of indices of tree nodes (*TNode*) traversed by BFS. At the beginning, the groups (tree nodes) in *CNC* are added to *TNode*, and the corresponding indices are added to *queNode*, in lines 5–8. For example, in Fig. 1(b), $CNC = \{G_1\}$ and, therefore, $TNode_1$ is G_1 and $queNode_1$ is 1. Then, BFS is used to traverse all tree nodes in lines 10–30.

In BFS, getMVS is used to find the MVS (overlapping variables) of the current traversing node *Tnode* (the node in *TNode* with the index $queNode_h$) for judging whether the node can be divided or not. If the condition in line 14 (the termination of ROGD, including the minimum size of groups D/α and the maximum size of MVS $|Tnode.v|/\beta$) is satisfied, *Tnode* is a leaf node and does not need to be divided again. After adding the MVS_anc (lines 15–17), *Tnode* is regarded as a final group and is added into *Group* (line 18). If the condition in line 14 is not satisfied, it represents that *Tnode* needs to be divided again (lines 20–30). After removing the overlapping variables (*Tnode.ovp*), *Tnode* is divided and *Children* are the decomposed groups (lines 20 and 21). If there are separable variables (*seps*) in *Children*, *seps* will be taken as a final group after adding the interactive MVS_anc and be removed from *Children* (lines 23–26). Then, all the children nodes in *Children* are added into *TNode*, and the indices are added into *queNode* (lines 27–30). Afterward, the next tree node will be traversed.

For example, in Fig. 1(b), group G_1 in *CNC* is added into *TNode*, and queNode $= \{1\}$ in lines 5–8. In the first loop (lines 10–30), queNode$_h$ $= 1$ and the node G_1 is traversed. The tree nodes G_2, G_3, and G_4 (children nodes of G_1) are added into *TNode*, and queNode $= \{1, 2, 3, 4\}$. In the second loop, queNode$_h$ $= 2$ and G_2 is traversed. Because G_2 is a leaf node, no nodes are added into *TNode*. In the third loop, queNode$_h$ $= 3$ and G_3 is traversed. G_5 and G_6 (children nodes of G_3) are added into *TNode*, and queNode $= \{1, 2, 3, 4, 5, 6\}$. In the following loops, G_4, G_5, and G_6 are traversed, respectively, and no nodes are added into *TNode*, since G_4, G_5, and G_6 are all the leaf nodes and do not have children nodes.

B. Adjustment of Grouping

After the recursive decomposition in ROGD, if the number of groups is still small, it shows that the connectivity of the graph is strong. Considering the independence among groups in CCEAs, there is no need to divide the strongly connected graph. However, if the number of groups is big, such as f_{12} in IEEE CEC 2013 [43] which has 496 groups after being decomposed by ROGD, it will consume lots of fitness evaluations in every iteration and shorten the evolutionary process of CCEAs (assumed that the terminating condition is the maximum number of fitness evaluations). To reduce the number of groups, a grouping adjustment method is proposed to merge small groups with overlapping components.

For a problem with D dimensions (a graph with D vertices), if the number of its final groups is more than D/α, small groups will be merged via their overlapping variables. Algorithm 5 shows the details of the grouping adjustment method, where *Group* is the set of final groups. First, all overlapping variables

Algorithm 5: *Group* = Adjust(*Group*, *D*)

```
Begin
1  While $|Group| \geq D/\alpha$ Do
2      $MVS.V = \phi$; // set of overlapping variables
3      $MVS.V\_num = \phi$; // $MVS.V\_num_i$ represents the number
       of variables that appear in same groups with $MVS.V_i$
4      For all variables $v \in Group$ Do
5          if $v$ appears more than twice in $Group$ Then
6              $MVS.V = MVS.V \cup \{v\}$;
7      For $i = 1$: $|MVS.V|$ Do
8          $MVS.V\_num_i = 0$;
9          For all groups $g \in Group$ Do
10             if $MVS.V_i$ appears in $g$ Then
11                 $MVS.V\_num_i = MVS.V\_num_i + |g| - 1$; // $|g|$
                   $- 1$, the number of all variables in $g$, except
                   for $MVS.V_i$
12     Sort $MVS$ by $MVS.V\_num$ in ascending order;
13     Merge all groups in $Group$ that include $MVS.V_1$;
End
```

(*MVS.V*) are stored in lines 4–6. Then, the number of variables that appear in the same groups with $MVS.V_i$ ($MVS.V\_num_i$) is calculated in lines 7–11, where $MVS.V_i$ is the ith variable in $MVS.V$. If $MVS.V\_num_i$ is relatively small, there is no need to separate the groups including the overlapping variable $MVS.V_i$. Therefore, the groups including the overlapping variable with the smallest $MVS.V\_num$ are merged (lines 12 and 13). This process is repeated until the number of groups is not more than D/α.

For example, if the grouping result in Fig. 1(d) needs to be adjusted, $MVS.V = \{7, 8, 6\}$. Vertex 7 appears in G_2, G_4, and G_6, and $MVS.V\_num_1$ is equal to 8 ($|G_2| - 1 + |G_4| - 1 + |G_6| - 1$). Similarly, $MVS.V\_num_2 = 6$, and $MVS.V\_num_3 = 7$ ($MVS.V\_num = \{8, 6, 7\}$). Therefore, the number of variables that occur in same groups with the vertex 8 is least ($MVS.V\_num_2 = 6$), and the groups (G_2, G_4, and G_7) that include vertex 8 will be merged.

C. Complexity Analysis

From the observation of Algorithm 1, it can be seen that the GDD method does not consume fitness evaluations. For the analysis of the time complexity, all components of GDD are needed to be analyzed, including ConnComp, ROGD (Algorithm 4), and Adjust (Algorithm 5) which are all implemented based on the IaV matrix $\mathbf{\Theta}$. ConnComp is also used in GDG [36] and DG2 [37] to construct the initial graph, and it can be implemented by the Warshall algorithm [52] whose complexity is $O(D^3)$.

It is assumed that there are N tree nodes and M final groups (leaf nodes) during the decomposition ($M < N$). In the worst case, such as the chain-shaped grouping result (f_{12} in IEEE CEC 2013 [45], shown in Fig. S-5 of the supplementary material), the middle nodes only have an overlapping variable and only one variable will be separated every time, and the number of variables of $N-M$ middle nodes are $D, D-2, D-4, \ldots, D/\alpha + 2$ (which is the lower bound of the number of nodes of a group in line 14 in Algorithm 4), respectively. Therefore, in f_{12}, M is equal to $D/2 \times (1 - 1/\alpha) + 1 (D - 2 \times (M - 1) = D/\alpha)$, and N is equal to $(M - 1) \times 2 + 1 = (1 - 1/\alpha) \times D + 1$. In the best case, all groups (CNC) obtained in line 3 in Algorithm 1 are not needed to be divided again, such as $f_4 - f_{11}$ in IEEE CEC 2013 [45], and $N = M = |CNC|$. Without loss of generality, it is assumed that a function F, all the middle nodes have only an overlapping variable, and they are divided in half until the number of variables of the nodes reaches the lower bound D/α (termination of ROGD). The decomposition of F is shown in Fig. S-6 of the supplementary material. Therefore, in F, M is equal to $\alpha \times (D + 1)/(D + \alpha)((D - (M - 1))/M = D/\alpha)$, and N is equal to $M \times 2 - 1 = 2\alpha \times (D + 1)/(D + \alpha) - 1$.

ROGD (Algorithm 4) includes three main components: getMVS (line 13), DFS_MVSanc (line 16), and ConnComp (line 21). getMVS is implemented by the Dinic algorithm with the complexity of $O(D'^3)$ [49] in every loop where D' is the number of variables of the current traversing node *Tnode* (line 12). Therefore, getMVS is executed N times in ROGD, and the total complexity of this part is equal to $O(|G_1|^3 + |G_2|^3 + \cdots + |G_N|^3)$ where G_1 to G_N are the tree nodes of the decomposition. We assume that the nodes from G_{N-M+1} to G_N are the M leaf nodes (the final decomposed groups). For DFS_MVSanc (Algorithm 3), the operations between lines 4 and 5 in Algorithm 3 will be executed ($|Tnode.v| \times |V|$) times, where V is the MVS_anc of *Tnode*. DFS_MVSanc is only executed for M leaf nodes in ROGD, and the total complexity of this part is $O(|G_{N-M+1}| \times |V_{N-M+1}| + \cdots + |G_N| \times |V_N|)$, where V_i is the MVS_anc of G_i, and $|G_i|$ and $|V_i|$ are both smaller than D. ConnComp is only executed for each middle nodes (G_1 to G_{N-M}), and the complexity of this part in ROGD is $O(|G_1|^3 + |G_2|^3 + \cdots + |G_{N-M}|^3)$. It can be seen that the complexity of getMVS in ROGD is higher than DFS_MVSanc and ConnComp. Therefore, the complexity of ROGD mainly depends on getMVS ($O(|G_1|^3 + |G_2|^3 + \cdots + |G_N|^3)$), denoted as O(ROGD).

In the worst case (f_{12} in IEEE CEC 2013), the number of variables of each tree node is shown in Fig. S-5 in the supplementary material, and O(ROGD) is equal to $O(D^3 + (D - 2)^3 + \cdots + (D/\alpha)^3 + M) < O(D^4)$. In the best case ($f_4$–$f_{11}$ in IEEE CEC 2013), O(ROGD) is equal to $O(|G_1|^3 + |G_2|^3 + \cdots + |G_{|CNC|}|^3) < |CNC| \times O(D^3) \approx O(D^3)$ if $|CNC| << D$. For function F, the number of variables of each tree node is shown in Fig. S-6 in the supplementary material, and O(ROGD) is equal to $O(D^3 + ((D - 1)/2)^3 \times 2 + ((D - 3)/4)^3 \times 4 + \cdots + (D/\alpha)^3 \times M) < O(D^3 + D^3 \times 2 + D^3 \times 4 + \cdots + D^3 \times M) \approx O(D^3)$. Therefore, the complexity of ROGD is $O(D^4)$ in the worst case and $O(D^3)$ in other cases.

If M is bigger than D/α, the Adjust algorithm (Algorithm 5) is used to adjust the decomposed results. In the worst case, the number of groups decreases from M to D/α by only one every time. The operations between lines 2 and 13 in Algorithm 5 will be executed ($M - D/\alpha$) times and $M - D/\alpha < D$. The complexity of operations in lines 4–6, lines 7–11, line 12, and line 13 are $O(D^2)$, $O(V\_num \times D)$, $O(V\_num \times \log(V\_num))$, and $O(D)$, where V_num is the number of overlapping variables in the current loop and smaller than D. Therefore, the complexity of Adjust is $O(D^2 \times (M - D/\alpha)) < O(D^3)$.

Overall, ROGD is the most time-consuming part of GDD, and the time complexity of GDD is $O(D^4)$ in the worst case and $O(D^3)$ in other cases.

IV. Experimental Results and Analysis

In the comparative experiment, five benchmark functions from IEEE CEC 2013 [45] and 20 overlapping functions which are randomly generated are tested to verify the performance of GDD on decomposing the LSOPs. As GDD is based on the IaV matrix $\boldsymbol{\Theta}$ to deeply decompose the problem, it will be performed on the complete and high accurate $\boldsymbol{\Theta}$ obtained by GDG [36] and DG2 [37], resulting in the corresponding algorithms denoted as GDG_GDD and DG2_GDD, respectively. It should be noted that RDG2 and RDG3 are not combined with GDD, because they cannot get the complete IaV. Let $\boldsymbol{\Theta}_I$ and $\boldsymbol{\Theta}_A$ denote the ideal $\boldsymbol{\Theta}$ and the $\boldsymbol{\Theta}$ obtained by the algorithm A, respectively. $\boldsymbol{\Theta}_I$ can be obtained by the method used in [37], with the source code available from https://bitbucket.org/mno/differential-grouping2/src/master/matlab/adjmatrix2013.m.

In the following experiments, the test suite is first introduced, including 20 new overlapping functions. Then, the grouping accuracy of the decomposition methods before and after employing GDD are analyzed. Afterward, the grouping efficiency and the fault-tolerance ability of GDD are verified. Finally, the GDD-enhanced decomposition methods are incorporated into the third version of contribution-based cooperative co-evolutionary algorithm (CBCC3) [53] to compare with some state-of-the-art large-scale optimization algorithms. The CCBC3 framework will first consider and optimize the subproblem which contributes more to the current improvement of the whole problem optimization.

A. Test Suite

The test suite includes five functions in IEEE CEC 2013 [45], such as $f_7, f_{11}, f_{12}, f_{13}$, and f_{14}. f_{12}, f_{13}, and f_{14} are overlapping functions with conforming components and conflicting components, respectively, and their ideal grouping cases are descripted in Section S-III of the supplementary material. f_7 and f_{11} are partially additively separable functions. As the experimental results on f_7 and f_{11} can validate the fault-tolerance ability of GDD (as shown in Section IV-D), they are also chosen as the test functions.

Moreover, since there are only three overlapping functions in IEEE CEC 2013, 20 new overlapping functions are designed based on f_{13} and f_{14}, denoted as o_1 to o_{20}. For simplicity, the only difference between these generated overlapping functions and f_{13} and f_{14} is the vector $\boldsymbol{P}$ that represents the ideal grouping case. Vector $\boldsymbol{P}$ and the ideal groups of o_{2i-1} are the same as o_{2i} $(i = 1, 2, \ldots, 10)$. Except for $\boldsymbol{P}$, other parameters of o_{2i-1} are the same as f_{13}, and other parameters of o_{2i} are the same as f_{14}. In f_{13} and f_{14}, the grouping type is chain shaped (groups are pairwise joint) with a fixed overlap size [45]. Therefore, in order to increase the overlapping types of the tested problems, new functions are designed based on random overlapping variables and overlap sizes to enhance the diversity of the overlap type. For example, Fig. S-7 of the supplementary material shows another complicated overlap type rather than the chain-shaped. The generation of different types of overlapping functions is described in Section S-IV of the supplementary material, including the generation of vector $\boldsymbol{P}$, differences between o_1 to o_{20} and f_{13} and f_{14}, and their ideal grouping results. Datasets are available at https://github.com/zhangxin-Jancy/Benchmarks_for_overlappingLSOP.

B. Analysis of Grouping Accuracy

GDG and DG2 use three metrics ρ_1, ρ_2, and ρ_3 to measure the accuracy of identifying three types of relationships in IaV [36], [37]. However, these metrics are only related to the matrix $\boldsymbol{\Theta}$ and cannot evaluate the real grouping results of algorithms. Therefore, two new metrics are proposed in this article, including the overlapping rate (R_{ol}) and the redundancy rate (R_{rd}). Let $\boldsymbol{Group}^I = \{G_1^I, G_2^I, \ldots, G_n^I\}$ denote the ideal grouping result and $\boldsymbol{Group}^A = \{G_1^A, G_2^A, \ldots, G_m^A\}$ denote the grouping result obtained by a decomposition method A, where n and m are the number of groups in $\boldsymbol{Group}^I$ and $\boldsymbol{Group}^A$, respectively. The ideal grouping results $\boldsymbol{Group}^I$ can be obtained based on the parameters of the benchmark functions [45], as mentioned in Section II-A and Section S-III of the supplementary material. It should be noted that each separable variable is regarded as a single group to evaluate the identification of separable variables accurately. Before the calculation of R_{ol} and R_{rd}, the maximum matching of groups in $\boldsymbol{Group}^I$ and $\boldsymbol{Group}^A$ is obtained, denoted as $\boldsymbol{Group}^I \cap_{\max} \boldsymbol{Group}^A$. The group ${G^I}_i$ corresponds to the group ${G^A}_j$ which has the most common variables with ${G^I}_i$. If groups ${G^I}_{i1}$ and ${G^I}_{i2}$ both correspond to the group ${G^A}_j$, ${G^A}_j$ will choose the group with more common variables, and different ${G^I}_i$ corresponds to different ${G^A}_j$. Fig. S-8 of the supplementary material shows an example of $\boldsymbol{Group}^I \cap_{\max} \boldsymbol{Group}^A$, where ${G^I}_1$ corresponds to ${G^A}_1$, and ${G^I}_2$ corresponds to ${G^A}_2$ or ${G^A}_3$ and, therefore, $\boldsymbol{Group}^I \cap_{\max} \boldsymbol{Group}^A = \{\{1, 2, 4\}, \{4, 5\}\}$ or $\{\{1, 2, 4\}, \{4, 6\}\}$.

The overlapping rate R_{ol} is the ratio of the number of variables grouped in the right groups to the total number of variables in $\boldsymbol{Group}^I$ (including the repetitive variables). The redundancy rate R_{rd} is the ratio of the number of redundant variables to the total number of variables in $\boldsymbol{Group}^A$, where redundant variables are included in $\boldsymbol{Group}^A$ but not in $\boldsymbol{Group}^I \cap_{\max} \boldsymbol{Group}^A$. R_{ol} and R_{rd} are calculated as follows:

$$R_{ol} = \frac{\sum_{i=1}^{n} |G_i^I \cap_{\max} G_{ji}^A|}{\sum_{i=1}^{n} |G_i^I|} \times 100\% \tag{1}$$

$$R_{rd} = \frac{\sum_{j=1}^{m} |G_j^A| - \sum_{i=1}^{n} |G_i^I \cap_{\max} G_{ji}^A|}{\sum_{j=1}^{m} |G_j^A|} \times 100\% \tag{2}$$

where G_{ji}^A is corresponding to ${G^I}_i$ in the maximum matching. If $R_{ol} = 100\%$ and $R_{rd} = 0\%$, it represents that $\boldsymbol{Group}^A$ is same as $\boldsymbol{Group}^I$. Besides, $\boldsymbol{Group}^A$ with a bigger value of R_{ol} and a smaller value of R_{rd} is closer to $\boldsymbol{Group}^I$. Therefore, the larger R_{ol} and the smaller R_{rd} are better. If the $\boldsymbol{Group}^I$ of the graph in Fig. 1(a) is $\{9, 10, 11, 7, 8\}$, $\{1, 2, 5, 6\}$, and $\{3, 4, 6, 7\}$, R_{ol} and R_{rd} of the grouping result in Fig. 1(c) are 92.3% and 25%, respectively, and the R_{ol} and R_{rd} of the grouping result in Fig. 1(d) are 92.3% and 29.4%, respectively. It also indicates that adding MVS after finding the leaf nodes (**Rule 2**) is helpful to decrease R_{rd} of decomposition methods.

TABLE I
R_{ol} AND R_{rd} OF DIFFERENT DECOMPOSITION METHODS (%)

Func.	R_{ol}					R_{rd}				
	RDG3	GDG	GDG _GDD	DG2	DG2 _GDD	RDG3	GDG	GDG _GDD	DG2	DG2 _GDD
f_7	**100.00**	**100.00**	**100.00**	95.00	**100.00**	**0.00**	**0.00**	**0.00**	5.00	0.30
f_{11}	**100.00**	10.00	**100.00**	**100.00**	**100.00**	**0.00**	90.00	1.86	**0.00**	**0.00**
f_{13}	79.50	10.00	**100.00**	10.00	**100.00**	12.15	88.95	**0.00**	88.95	**0.00**
f_{14}	73.40	10.00	99.20	10.00	**100.00**	18.90	88.95	4.25	88.95	**0.00**
o_1	40.10	11.20	54.40	10.00	**99.90**	55.69	87.62	48.44	88.95	**19.11**
o_2	39.90	10.20	95.50	10.00	**100.00**	55.91	88.73	4.02	88.95	**0.00**
o_3	40.90	12.10	71.10	12.50	**100.00**	54.81	86.63	50.38	86.19	**0.00**
o_4	41.20	13.00	95.50	12.50	**100.00**	54.48	85.64	3.83	86.19	**0.00**
o_5	69.40	11.20	59.60	10.00	**99.50**	23.31	87.62	40.46	88.95	**25.97**
o_6	49.40	10.50	95.50	10.00	**100.00**	45.41	88.40	3.83	88.95	**0.00**
o_7	63.40	11.40	37.20	10.10	**95.90**	29.94	87.40	69.78	88.84	**22.35**
o_8	59.90	10.50	95.50	10.00	**100.00**	33.81	88.40	3.24	88.95	**0.00**
o_9	41.70	10.50	51.70	10.00	**100.00**	53.92	88.40	68.36	88.95	**2.25**
o_{10}	43.30	10.20	95.50	10.00	**100.00**	52.15	88.73	3.83	88.95	**0.00**
o_{11}	68.90	11.60	66.00	10.20	**94.60**	23.87	87.18	41.90	88.73	**26.15**
o_{12}	65.30	10.20	95.50	10.00	**100.00**	27.85	88.73	3.14	88.95	**0.00**
o_{13}	61.20	10.90	75.90	10.10	**98.70**	32.38	87.96	31.13	88.84	**20.53**
o_{14}	58.20	10.50	95.50	10.00	**100.00**	35.69	88.40	3.14	88.95	**0.00**
o_{15}	48.80	10.20	93.80	10.00	**100.00**	46.08	88.73	25.67	88.95	**0.00**
o_{16}	48.80	10.50	95.50	10.00	**100.00**	46.08	88.40	3.73	88.95	**0.00**
o_{17}	61.80	11.70	46.00	11.70	**80.40**	31.71	87.07	66.74	87.07	**26.91**
o_{18}	41.70	10.50	95.50	10.00	**100.00**	53.92	88.40	3.54	88.95	**0.00**
o_{19}	72.00	10.30	91.50	10.00	**100.00**	20.44	88.62	23.11	88.95	**0.00**
o_{20}	72.00	10.20	95.40	10.00	**100.00**	20.44	88.73	19.97	88.95	**0.00**
Avg.	60.03	14.48	83.39	17.59	**98.71**	34.54	84.49	21.85	81.42	**5.98**

With the use of R_{ol} and R_{rd}, the hyperparameter tuning is investigated in Section S-V of the supplementary material.

Table I shows R_{ol} and R_{rd} of RDG3, GDG, GDG_GDD, DG2, and DG2_GDD. The last line represents the average results (Avg.). The **bold** data represent the best results among all algorithms. Moreover, the results with underlining are the better results between two variants of compared algorithms (e.g., GDG and GDG_GDD).

For the metric of R_{ol}, if R_{ol} of algorithm A is equal to 100%, it indicates that $\boldsymbol{Group}^A \supset \boldsymbol{Group}^I$. As shown in Table I, DG2_GDD obtains the highest the average R_{ol}, and the R_{ol} of it reaches 100% on 18 functions (24 functions in total). Following DG2_GDD, GDG_GDD can also obtain higher R_{ol} than other algorithms. Because GDG and DG2 cannot divide the overlapping functions (f_{13}, f_{14}, and o_1 to o_{20}) and obtain accurate IaV (f_7 and f_{11}), there will be fewer groups to match $\boldsymbol{Group}^I$, and the average R_{ol} of them is much lower than GDG_GDD and DG2_GDD, respectively. RDG3 will forcedly divide the variables of a larger group into different smaller groups, and break the intergroup independence and ignore the completeness of grouping (as mentioned in Section III-A). Therefore, although the average R_{ol} of RDG3 is higher than GDG and DG2, it is lower than GDG_GDD and DG2_GDD.

For the metric of R_{rd}, if the R_{rd} of an algorithm is equal to 0%, it indicates $\boldsymbol{Group}^A$ has no more variables than the common variables with $\boldsymbol{Group}^I$ ($\boldsymbol{Group}^I \cap_{\max} \boldsymbol{Group}^A$). As shown in Table I, DG2_GDD followed by GDG_GDD can obtain the lowest average R_{rd}, and the R_{rd} of it can reach 0% on 16 functions. In addition, because the groups of GDG and DG2 are larger without decomposition, and there are more redundant variables after matching with the groups in $\boldsymbol{Group}^I$, the average R_{rd} of GDG and DG2 is worse (i.e., higher) than RDG3, GDG_GDD, and DG2_GDD.

Based on the comprehensive analysis of the results of R_{ol} and R_{rd}, it can be concluded that GDG_GDD and DG2_GDD can obtain higher accurate groups not only in overlapping functions but also in partially separable functions (f_7 and f_{11}). As the accuracy IaV obtained by DG2 is higher than that obtained by GDG (i.e., $\boldsymbol{\Theta}_{\text{DG2}}$ is higher than $\boldsymbol{\Theta}_{GDG}$) [37], DG2 and DG2_GDD can get more accurate groups than GDG and GDG_GDD, respectively. The grouping accuracy of RDG3 is higher than GDG and DG2, but lower than the methods combined with GDD.

C. Analysis of the Grouping Efficiency of GDD

From the observation of the grouping results, it can be seen that GDG and DG2 cannot divide the variables of overlapping components. After being combined with GDD, the average R_{ol} of GDG and DG2 has an increase, and the R_{rd} of them has a decrease. It indicates that GDD can help GDG and DG2 to get more accurate group on these functions, including overlapping functions f_{13} and f_{14}, o_1–o_{20}, and partially separable functions f_7 and f_{11}. The reason is that GDD can find the overlapping variables (i.e., the MVS) among connected components (i.e., the overlapping groups) through the IaV matrix $\boldsymbol{\Theta}$. Therefore, GDD can divide the overlapping functions accurately if $\boldsymbol{\Theta}_A$ is close to the ideal $\boldsymbol{\Theta}_I$, such as $\boldsymbol{\Theta}_{\text{DG2}}$ on f_{13} and f_{14} [37]. In addition, if the independent variables are wrongly judged as interactive variables (i.e., $\boldsymbol{\Theta}_I(i, j) = 0$ but $\boldsymbol{\Theta}_A(i, j) = 1$), GDD can also help decomposition methods to overcome the wrong interactive information to get more accurate groups. That is because the groups of $\boldsymbol{Group}^I$ including the variables i and j will be merged into a group G^A of $\boldsymbol{Group}^A$, but G^A can be decomposed after GDD removing the MVS ({the variable i} or {the variable j}) of G^A, as shown in Fig. S-9 of the supplementary material.

To validate the effectiveness of **Rule 2**, GDG_GDD and DG2_GDD without this rule are tested, as shown in Table II. GDG_GDD and DG2_GDD without **Rule 2** are denoted as GDG_GDD2 and DG2_GDD2, respectively. The last line of the table is the average value of R_{ol} or R_{rd}. The results in **bold** are the best results among all algorithms, and the results with underlining are the better results between two variants of compared algorithms (e.g., GDG_GDD and GDG_GDD2).

As shown in Table II, R_{rd} of DG2_GDD2 is higher than DG2_GDD. The reason is that as described in Section III-A, if MVS is added after dividing a group for the first time (as in DG2_GDD2), MVS will always be involved in the following decomposition, and there will be more redundant variables in groups. On the other hand, the problem will be divided into more groups, and the number of variables in each group will decrease corresponding to the matched $\boldsymbol{Group}^I$. Therefore, R_{ol} of DG2_GDD2 is lower than DG2_GDD. The differences between GDG_GDD2 and GDG_GDD are similar.

To verify the effect of **Rule 2** on the solving efficiency of the LSOPs, GDG_GDD2, GDG_GDD, DG2_GDD2, and DG2_GDD are incorporated into CBCC3 which is sensitive to the grouping accuracy [33], [37], [53], and SaNSDE [54]

TABLE II
R_{ol} AND R_{rd} OF GDG_GDD AND DG2_GDD WITH AND WITHOUT RULE 2 (%)

Func.	R_{ol}				R_{rd}			
	GDG_GDD	GDG_GDD2	DG2_GDD	DG2_GDD2	GDG_GDD	GDG_GDD2	DG2_GDD	DG2_GDD2
f_7	**100.00**	**100.00**	**100.00**	**100.00**	**0.00**	**0.00**	0.30	0.79
f_{11}	**100.00**	**100.00**	**100.00**	**100.00**	1.86	9.58	**0.00**	**0.00**
f_{13}	**100.00**	**100.00**	**100.00**	**100.00**	**0.00**	**0.00**	**0.00**	**0.00**
f_{14}	99.20	**100.00**	**100.00**	**100.00**	4.25	**0.00**	**0.00**	**0.00**
o_1	54.40	55.90	**99.90**	74.30	48.44	46.86	**19.11**	42.45
o_2	95.50	95.50	**100.00**	**100.00**	4.02	4.02	**0.00**	**0.00**
o_3	71.10	64.10	**100.00**	**100.00**	50.38	43.67	**0.00**	**0.00**
o_4	95.50	95.50	**100.00**	**100.00**	3.83	3.83	**0.00**	**0.00**
o_5	59.60	59.60	**99.50**	67.10	40.46	40.76	**25.97**	47.62
o_6	95.50	95.50	**100.00**	**100.00**	3.83	3.83	**0.00**	**0.00**
o_7	37.20	56.30	**95.90**	51.50	69.78	48.77	**22.35**	59.38
o_8	95.50	95.50	**100.00**	**100.00**	3.24	3.24	**0.00**	**0.00**
o_9	51.70	76.40	**100.00**	**100.00**	68.36	37.48	**2.25**	**2.25**
o_{10}	95.50	95.50	**100.00**	**100.00**	3.83	3.83	**0.00**	**0.00**
o_{11}	66.00	54.90	**94.60**	72.00	41.90	50.18	**26.15**	42.45
o_{12}	95.50	95.50	**100.00**	**100.00**	3.14	3.14	**0.00**	**0.00**
o_{13}	75.90	73.60	**98.70**	65.70	31.13	35.33	**20.53**	49.62
o_{14}	95.50	95.50	**100.00**	**100.00**	3.14	3.14	**0.00**	**0.00**
o_{15}	93.80	75.70	**100.00**	**100.00**	25.67	38.56	**0.00**	**0.00**
o_{16}	95.50	95.50	**100.00**	**100.00**	3.73	3.73	**0.00**	**0.00**
o_{17}	46.00	50.50	**80.40**	74.30	66.74	54.30	**26.91**	33.06
o_{18}	95.50	95.50	**100.00**	**100.00**	3.54	3.54	**0.00**	**0.00**
o_{19}	91.50	77.70	**100.00**	**100.00**	23.11	36.05	**0.00**	**0.00**
o_{20}	95.40	67.30	**100.00**	**100.00**	19.97	47.71	**0.00**	**0.00**
Avg.	83.39	82.15	**98.71**	91.87	21.85	21.73	**5.98**	11.57

TABLE III
ACCURACY RATE (ρ_1, ρ_2, AND ρ_3) OF $\mathbf{\Theta}_{\text{DG2}}$ AND THE GROUPING ACCURACY (R_{ol} AND R_{rd}) OF DG2_GDD (%)

Func.	$\mathbf{\Theta}_{\text{DG2}}$			DG2_GDD	
	ρ_1	ρ_2	ρ_3	R_{ol}	R_{rd}
f_7	100.00	*100.00*	*100.00*	100.00	0.30
f_{11}	99.95	100.00	*100.00*	100.00	0.00
f_{13}	100.00	100.00	100.00	100.00	0.00
f_{14}	99.98	100.00	*100.00*	100.00	0.00
o_1	99.40	100.00	99.95	99.90	19.11
o_2	99.98	100.00	*100.00*	100.00	0.00
o_3	99.68	100.00	99.97	100.00	0.00
o_4	99.98	100.00	*100.00*	100.00	0.00
o_5	97.73	100.00	99.81	99.50	25.97
o_6	99.98	100.00	*100.00*	100.00	0.00
o_7	95.48	100.00	99.63	95.90	22.35
o_8	99.98	100.00	*100.00*	100.00	0.00
o_9	*100.00*	99.92	99.92	100.00	2.25
o_{10}	99.98	100.00	*100.00*	100.00	0.00
o_{11}	96.76	100.00	99.73	94.60	26.15
o_{12}	99.98	100.00	*100.00*	100.00	0.00
o_{13}	98.96	100.00	99.91	98.70	20.53
o_{14}	99.98	100.00	*100.00*	100.00	0.00
o_{15}	100.00	100.00	100.00	100.00	0.00
o_{16}	99.98	100.00	*100.00*	100.00	0.00
o_{17}	83.90	100.00	98.67	80.40	26.91
o_{18}	99.98	100.00	*100.00*	100.00	0.00
o_{19}	100.00	100.00	100.00	100.00	0.00
o_{20}	99.98	100.00	*100.00*	100.00	0.00
Avg.	98.82	100.00	99.90	98.71	5.98

is chosen as the optimizer. The optimization results of them are shown in Tables S-I and S-II of the supplementary material, respectively. The maximum number of fitness evaluations is set to 300 0000. Each experiment is conducted for 25 times independently. In addition, the Wilcoxon rank-sum test at a 5% significance level is used for the statistical comparisons. The last column of the tables is the number of wins, ties, and losses of GDG_GDD or DG2_GDD against other algorithms. From the observation of Table S-I and Table S-II in the supplementary material, it can be seen that the optimization results of GDG_GDD and DG2_GDD are better than GDG_GDD2 and DG2_GDD2, respectively.

In summary, the designed rules can not only prescribe how to get the complete grouping results after adding MVS, but also help to improve the accuracy of grouping and get better optimization results of the LSOPs.

D. Analysis of the Grouping Fault Tolerance of GDD

From the observation of the results mentioned above, it can be drawn that GDD helps decomposition methods to get more accurate groups not only on overlapping functions but on some nonoverlapping functions. It also shows that GDD has the capacity of fault tolerance, since the GDD decomposes the problems via the MVS deeply and recursively. Differently, the compared decomposition methods, such as RDG3, GDG, and DG2, directly obtain the groups without mining IaV for deep decomposition and, therefore, they are not fault tolerant.

Taking DG2_GDD as an example, the grouping accuracy of it partially depends on the accuracy rate of $\mathbf{\Theta}_{\text{DG2}}$. To analyze the influence of the accuracy rate of $\mathbf{\Theta}_{\text{DG2}}$ on DG2_GDD, Table III shows the accuracy rate of $\mathbf{\Theta}_{\text{DG2}}$ and the grouping accuracy of DG2_GDD. ρ_1, ρ_2, and ρ_3 represent three different metrics of $\mathbf{\Theta}_{\text{DG2}}$ [36]. The data in *italic* type represents that the results reach to 100.00 after being rounded up. If $\rho_1 = \rho_2 = \rho_3 = 100\%$, it represents that the accuracy rate of $\mathbf{\Theta}_{\text{DG2}}$ reaches 100%, and $\mathbf{\Theta}_{DG2}$ is the same as $\mathbf{\Theta}_I$. As shown in Table III, if $\mathbf{\Theta}_{\text{DG2}}$ is the same as $\mathbf{\Theta}_I$ ($\rho_1 = \rho_2 = \rho_3 = 100\%$), the grouping accuracy of DG2_GDD can also reach to 100% ($R_{ol} = 100\%$ and $R_{rd} = 0\%$), such as on f_{13}, o_{15}, and o_{19}. However, when the accuracy rate of $\mathbf{\Theta}_{\text{DG2}}$ is close but not equal to 100% ($\rho_1 \approx \rho_2 \approx \rho_3 \approx 100\%$), DG2_GDD can still get the same grouping result as $\boldsymbol{Group}^{I}$ on most functions, such as the results on f_{11}, f_{14}, o_2, o_3, o_4, o_6, o_8, o_{10}, o_{12}, o_{14}, o_{16}, o_{18}, and o_{20}. Therefore, although the grouping accuracy of DG2_GDD is related to the accuracy rate of $\mathbf{\Theta}_{\text{DG2}}$, DG2_GDD can also get the ideal grouping results with the inaccurate $\mathbf{\Theta}_{\text{DG2}}$, which also proves DG2_GDD to be fault tolerant.

The reason why GDD is fault tolerant is due to the decomposition ability on overlapping LSOPs. Specifically, there are three factors. First, GDD can identify the overlapping variables (MVS) among overlapping groups, and divides these groups into smaller groups. As shown in Section IV-C, GDD can get more accurate groups after dividing the wrongly judged overlapping groups which are merged in $\boldsymbol{Group}^{\text{DG2}}$ but are separated in $\boldsymbol{Group}^{\text{DG2\_GDD}}$ and $\boldsymbol{Group}^{I}$. For example, as shown in Table S-III of the supplementary material, $\boldsymbol{Group}^{\text{DG2}}$ merges $\text{G}^{\text{DG2\_GDD}}{}_6$ and $\text{G}^{\text{DG2\_GDD}}{}_7$, and $\boldsymbol{Group}^{\text{DG2\_GDD}}$ on f_7 is closer to the corresponding $\boldsymbol{Group}^{I}$. Therefore, GDD can get more accurate groups. Second, the three rules, designed in Section III-A, help to add the overlapping variables to improve the grouping accuracy. **Rule 1** and **Rule 2** avoid adding redundant variables of overlapping components. These rules are helpful for the algorithms with inaccurate $\boldsymbol{\Theta}_A$ to get more accurate groups. For example, although $\boldsymbol{\Theta}_{\text{DG2}}$ on f_7, f_{11}, and f_{14} are inaccurate (as shown in Table III), DG2_GDD can still obtain the approximately ideal groups on these functions ($R_{ol} \approx 100\%$ and $R_{rd} \approx 0\%$). Third, the graph connectivity also helps to improve the fault-tolerance ability of GDD. For example, $\boldsymbol{\Theta}(1, 2) = 1$ and $\boldsymbol{\Theta}(1, 3) = 1$, variables 1, 2, and 3 will be assigned to a group whether $\boldsymbol{\Theta}(2, 3) = 1$ or $\boldsymbol{\Theta}(2, 3) = 0$. For example, $\boldsymbol{\Theta}_I$ and $\boldsymbol{\Theta}_{\text{DG2}}$ on f_{14} are different (e.g., $\boldsymbol{\Theta}_I$ (106, 225) = 1 and $\boldsymbol{\Theta}_{\text{DG2}}$(106, 225) = 0). The index of variables starts from 0), but $\boldsymbol{Group}^{\text{DG2\_GDD}}$ is same as $\boldsymbol{Group}^{I}$.

E. Comparison of Optimization Results

Table S-IV in the supplementary material shows the comparative results of DG2_GDD, DG2, GDG_GDD, GDG, and RDG3 with the CBCC3 framework [53] (denoted as CBCC3-DG2_GDD, CBCC3-DG2, CBCC3-GDG_GDD, CBCC3-GDG, and CBCC3-RDG3, respectively) and other well-known large-scale optimization algorithms, such as the version 2 of CC particle swarm optimization (PSO), i.e., CCPSO2 [46], competitive swarm optimization (CSO) [25], social learning PSO (SLPSO) [55], dynamic segment-based predominant learning swarm optimizer (DSPLSO) [56], and dynamic level-based learning swarm optimizer (DLLSO) [26].

Comparing the results of CBCC3-DG2 and CBCC3-DG2_GDD, it can be seen that CBCC3-DG2_GDD is better than CBCC3-DG2 on all test functions with the significant difference, except for f_{11} where the grouping result of DG2_GDD is same as DG2. It should be noted that CBCC3-DG2_GDD performs much better than CBCC3-DG2 on f_{12}. It is due to that DG2_GDD divides variables into several overlapping groups, which is useful for the CBCC3 framework to solve problems efficiently, but DG2 considers all variables of f_{12} as a single group. It can be concluded that GDD is helpful to improve not only the grouping accuracy but also the optimization efficiency of DG2. The comparative results of GDG and GDG_GDD are similar to DG2 and DG2_GDD.

From the observation of Table S-IV in the supplementary material, it can be concluded that CBCC3-DG2_GDD performs significantly better than other algorithms on most functions. GDG_GDD, RDG3, and CCPSO2 apply the random grouping, but DG2_GDD has a higher grouping accuracy and helps CBCC3 search solutions in a relatively clear direction within the decomposed space. Compared with CSO, SLPSO, DSPLSO, and DLLSO that solve problems as a whole, CBCC3-DG2_GDD decomposes LSOPs into relatively independent parts and solves them separately, which can search solutions broadly.

In this section, it can be concluded that GDD can not only help GDG and DG2 to decompose the overlapping problems but help CCEAs to outperform other excellent algorithms.

V. Conclusion

In this article, the GDD method was proposed to decompose overlapping LSOPs and improve the grouping accuracy. Specifically, GDD first obtains the MVS based on IaV and then separates the overlapping groups by their MVS. In addition, GDD uses the recursive method and the adjustment strategy to divide the overlapping problems into appropriate sizes. GDD has three advantages. First, GDD improves the grouping accuracy of decomposition methods, not only on the overlapping problems but also on partially separable problems. Second, GDD has a high fault-tolerance ability. It can help decomposition methods to divide problems into approximately ideal groups even if the corresponding IaV matrix $\boldsymbol{\Theta}$ is inaccurate. Third, GDD can help CCEAs to get better optimization results than other well-known optimization algorithms, especially on the overlapping problems.

To evaluate the grouping accuracy of decomposition methods, two new metrics were proposed, including the overlapping rate (R_{ol}) and the redundancy rate (R_{rd}). The exhaustive experiments were conducted on five functions of IEEE CEC 2013 and 20 designed overlapping functions, and the results showed that CCEAs combined with GDD achieve a better performance on the LSOPs.

References

[1] D. M. Cabrera, "Evolutionary algorithms for large-scale global optimisation: A snapshot, trends and challenges," *Progr. Artif. Intell.*, vol. 5, no. 2, pp. 85–89, Feb. 2016.

[2] X. Luo, Y. Yuan, S. Chen, N. Zeng, and Z. Wang, "Position-transitional particle swarm optimization-incorporated latent factor analysis," *IEEE Trans. Knowl. Data Eng.*, vol. 34, no. 8, pp. 3958–3970, Aug. 2022.

[3] Z.-J. Wang, J.-R. Jian, Z.-H. Zhan, Y. Li, S. Kwong, and J. Zhang, "Gene targeting differential evolution: A simple and efficient method for large scale optimization," *IEEE Trans. Evol. Comput.*, early access, Jun. 23, 2022, doi: 10.1109/TEVC.2022.3185665.

[4] R. Shang, K. Dai, L. Jiao, and R. Stolkin, "Improved memetic algorithm based on route distance grouping for multiobjective large scale capacitated arc routing problems," *IEEE Trans. Cybern.*, vol. 46, no. 4, pp. 1000–1013, Apr. 2016.

[5] F. Guo, G. Li, C. Wen, L. Wang, and Z. Meng, "An accelerated distributed gradient-based algorithm for constrained optimization with application to economic dispatch in a large-scale power system," *IEEE Trans. Syst., Man, Cybern., Syst.*, vol. 51, no. 4, pp. 2041–2053, Apr. 2021.

[6] X. Zhang, K.-J. Du, Z.-H. Zhan, S. Kwong, T.-L. Gu, and J. Zhang, "Cooperative coevolutionary bare-bones particle swarm optimization with function independent decomposition for large-scale supply chain network design with uncertainties," *IEEE Trans. Cybern.*, vol. 50, no. 10, pp. 4454–4468, Oct. 2020.

[7] X. Zhang, Z.-H. Zhan, W. Fang, P. Qian, and J. Zhang, "Multipopulation ant colony system with knowledge-based local searches for multiobjective supply chain configuration," *IEEE Trans. Evol. Comput.*, vol. 26, no. 3, pp. 512–526, Jun. 2022.

[8] S. Liu, Q. Lin, Q. Li, and K. C. Tan, "A comprehensive competitive swarm optimizer for large-scale multiobjective optimization," *IEEE Trans. Syst., Man, Cybern., Syst.*, vol. 52, no. 9, pp. 5829–5842, Sep. 2022.

[9] S.-H. Wu, Z.-H. Zhan, and J. Zhang, "SAFE: Scale-adaptive fitness evaluation method for expensive optimization problems," *IEEE Trans. Evol. Comput.*, vol. 25, no. 3, pp. 478–491, Jun. 2021.

[10] Y. Pan, K. Gao, Z. Li, and N. Wu, "Solving biobjective distributed flow-shop scheduling problems with lot-streaming using an improved Jaya algorithm," *IEEE Trans. Cybern.*, early access, Apr. 25, 2022, doi: 10.1109/TCYB.2022.3164165.

[11] J.-Y. Li, Z.-H. Zhan, C. Wang, H. Jin, and J. Zhang, "Boosting data-driven evolutionary algorithm with localized data generation," *IEEE Trans. Evol. Comput.*, vol. 24, no. 5, pp. 923–937, Oct. 2020.

[12] Z. H. Zhan et al., "Cloudde: A heterogeneous differential evolution algorithm and its distributed cloud version," *IEEE Trans. Parallel Distrib. Syst.*, vol. 28, no. 3, pp. 704–716, Mar. 2017.

[13] X.-F. Liu, Z.-H. Zhan, Y. Gao, J. Zhang, S. Kwong, and J. Zhang, "Coevolutionary particle swarm optimization with bottleneck objective learning strategy for many-objective optimization," *IEEE Trans. Evol. Comput.*, vol. 23, no. 4, pp. 587–602, Aug. 2019.

[14] J.-Y. Li et al., "A multipopulation multiobjective ant colony system considering travel and prevention costs for vehicle routing in COVID-19-like epidemics," *IEEE Trans. Intell. Transp. Syst.*, early access, Jun. 17, 2022, doi: 10.1109/tits.2022.3180760.

[15] X.-F. Liu et al., "Historical and heuristic-based adaptive differential evolution," *IEEE Trans. Syst., Man, Cybern., Syst.*, vol. 49, no. 12, pp. 2623–2635, Dec. 2019.

[16] K. Qiao, K. Yu, B. Qu, J. Liang, H. Song, and C. Yue, "An evolutionary multitasking optimization framework for constrained multiobjective optimization problems," *IEEE Trans. Evol. Comput.*, vol. 26, no. 2, pp. 263–277, Apr. 2022.

[17] C. E. da Silva Santos, R. C. Sampaio, L. dos Santos Coelho, G. A. Bestard, and C. H. Llanos, "Multi-objective adaptive differential evolution for SVM/SVR hyperparameters selection," *Pattern Recognit.*, vol. 110, Feb. 2021, Art. no. 107649.

[18] L. Meng, K. Gao, Y. Ren, B. Zhang, H. Sang, and Z. Chaoyong, "Novel MILP and CP models for distributed hybrid flowshop scheduling problem with sequence-dependent setup times," *Swarm Evol. Comput.*, vol. 71, Jun. 2022, Art. no. 101058.

[19] Z.-H. Zhan, Z.-J. Wang, H. Jin, and J. Zhang, "Adaptive distributed differential evolution," *IEEE Trans. Cybern.*, vol. 50, no. 11, pp. 4633–4647, Nov. 2020.

[20] J.-Y. Li, Z.-H. Zhan, K. C. Tan, and J. Zhang, "A meta-knowledge transfer-based differential evolution for multitask optimization," *IEEE Trans. Evol. Comput.*, vol. 26, no. 4, pp. 719–734, Aug. 2022.

[21] N. Zeng, Z. Wang, W. Liu, H. Zhang, K. Hone, and X. Liu, "A dynamic neighborhood-based switching particle swarm optimization algorithm," *IEEE Trans. Cybern.*, vol. 52, no. 9, pp. 9290–9301, Sep. 2022.

[22] J.-Y. Li, K.-J. Du, Z.-H. Zhan, H. Wang, and J. Zhang, "Distributed differential evolution with adaptive resource allocation," *IEEE Trans. Cybern.*, early access, Mar. 14, 2022, doi: 10.1109/TCYB.2022.3153964.

[23] J.-R. Jian, Z.-H. Zhan, and J. Zhang, "Large-scale evolutionary optimization: A survey and experimental comparative study," *Int. J. Mach. Learn. Cybern.*, vol. 11, no. 3, pp. 729–745, 2020.

[24] Z.-H. Zhan, L. Shi, K. C. Tan, and J. Zhang, "A survey on evolutionary computation for complex continuous optimization," *Artif. Intell. Rev.*, vol. 55, pp. 59–110, Jan. 2022.

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

[26] Q. Yang, W.-N. Chen, J. D. Deng, Y. Li, T. Gu, and J. Zhang, "A level-based learning swarm optimizer for large-scale optimization," *IEEE Trans. Evol. Comput.*, vol. 22, no. 4, pp. 578–594, Aug. 2018.

[27] Z.-J. Wang, Z.-H. Zhan, S. Kwong, H. Jin, and J. Zhang, "Adaptive granularity learning distributed particle swarm optimization for large-scale optimization," *IEEE Trans. Cybern.*, vol. 51, no. 3, pp. 1175–1188, Mar. 2021.

[28] Z.-J. Wang et al., "Dynamic group learning distributed particle swarm optimization for large-scale optimization and its application in cloud workflow scheduling," *IEEE Trans. Cybern.*, vol. 50, no. 6, pp. 2715–2729, Jun. 2020.

[29] J.-R. Jian, Z.-G. Chen, Z.-H. Zhan, and J. Zhang, "Region encoding helps evolutionary computation evolve faster: A new solution encoding scheme in particle swarm for large-scale optimization," *IEEE Trans. Evol. Comput.*, vol. 25, no. 4, pp. 779–793, Aug. 2021.

[30] J. Liang et al., "Cooperative co-evolutionary comprehensive learning particle swarm optimizer for formulation design of explosive simulant," *Memet. Comput.*, vol. 12, no. 4, pp. 331–341, Oct. 2020.

[31] Y.-H. Jia, Y. Mei, and M. Zhang, "Contribution-based cooperative co-evolution for nonseparable large-scale problems with overlapping subcomponents," *IEEE Trans. Cybern.*, vol. 52, no. 6, pp. 4246–4259, Jun. 2022.

[32] T. Jansen and R. P. Wiegand, "The cooperative coevolutionary (1+1) EA," *Evol. Comput.*, vol. 12, no. 4, pp. 405–434, 2004.

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

[34] M. N. Omidvar, X. Li, Y. Mei, and X. Yao, "Cooperative co-evolution with differential grouping for large scale optimization," *IEEE Trans. Evol. Comput.*, vol. 18, no. 3, pp. 378–393, Jun. 2014.

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

[36] Y. Mei, M. N. Omidvar, X. Li, and X. Yao, "A competitive divide-and-conquer algorithm for unconstrained large-scale black-box optimization," *ACM Trans. Math. Softw.*, vol. 42, no. 2, pp. 1–24, Jun. 2016.

[37] M. N. Omidvar, M. Yang, Y. Mei, X. Li, and X. Yao, "DG2: A faster and more accurate differential grouping for large-scale black-box optimization," *IEEE Trans. Evol. Comput.*, vol. 21, no. 6, pp. 929–942, Dec. 2017.

[38] Y. Sun, M. Kirley, and S. K. Halgamuge, "A recursive decomposition method for large scale continuous optimization," *IEEE Trans. Evol. Comput.*, vol. 22, no. 5, pp. 647–661, Oct. 2018.

[39] Y. Sun, M. N. Omidvar, M. Kirley, and X. Li, "Adaptive threshold parameter estimation with recursive differential grouping for problem decomposition," in *Proc. ACM Genet. Evol. Comput. Conf.*, 2018, pp. 889–896.

[40] Y. Sun, X. Li, A. Ernst, and M. N. Omidvar, "Decomposition for large-scale optimization problems with overlapping components," in *Proc. IEEE Congr. Evol. Comput.*, 2019, pp. 326–333.

[41] M. Yang, A. Zhou, C. Li, and X. Yao, "An efficient recursive differential grouping for large-scale continuous problems," *IEEE Trans. Evol. Comput.*, vol. 25, no. 1, pp. 159–171, Feb. 2021.

[42] J.-Y. Li, Z.-H. Zhan, K. C. Tan, and J. Zhang, "Dual differential grouping: A more general decomposition method for large-scale optimization," *IEEE Trans. Cybern.*, early access, Mar. 25, 2022, doi: 10.1109/TCYB.2022.3158391.

[43] W. Benameur and M. D. Biha, "On the minimum cut separator problem," *Networks*, vol. 59, no. 1, pp. 30–36, 2012.

[44] S. Even, "Applications of network flow techniques," in *Graph Algorithms*. Potomac, MD, USA: Comput. Sci. Press, 1979, pp. 121–130.

[45] 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," Dept. Evol. Comput. Mach. Learn. Subpopulation, RMIT Univ., Melbourne, VIC, Australia, Rep., 2013.

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

[47] H. A. Esfahanian, *On the Evolution of Graph Connectivity Algorithms*, Michigan State Univ., Michigan, MI, USA, 2002, pp. 10–11.

[48] E. A. Dinits, "Algorithms for solution of a problem of maximum flow in a network with power estimation," *Soviet Math. Doklad*, vol. 11, no. 11, pp. 1277–1280, 1970.

[49] J. X. Hao and J. B. Orlin, "A faster algorithm for finding the minimum cut in a directed graph," *J. Algorithms*, vol. 17, no. 3, pp. 424–446, Nov. 1994.

[50] R. Zhou and E. A. Hansen, "Breadth-first heuristic search," *Artif. Intell.*, vol. 170, nos. 4–5, pp. 385–408, Apr. 2006.

[51] C.-H. Peng, B.-F. Wang, and J.-S. Wang, "Recognizing unordered depth-first search trees of an undirected graph in parallel," *IEEE Trans. Parallel Distrib. Syst.*, vol. 11, no. 6, pp. 559–570, Jun. 2000.

[52] U. Berger, H. Schwichtenberg, and M. Seisenberger, "The warshall algorithm and Dickson's lemma: Two examples of realistic program extraction," *J. Autom. Reason.*, vol. 26, no. 2, pp. 205–221, Feb. 2001.

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

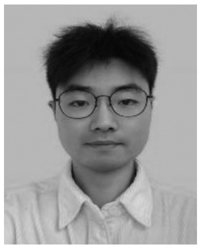
[54] Z. Yang, K. Tang, and X. Yao, "Self-adaptive differential evolution with neighborhood search," in *Proc. IEEE Congr. Evol. Comput.*, 2008, pp. 1110–1116.

[55] R. Cheng and Y. Jin, "A social learning particle swarm optimization algorithm for scalable optimization," *Inf. Sci.*, vol. 291, pp. 43–60, Jan. 2015.

[56] Q. Yang et al., "Segment-based predominant learning swarm optimizer for large-scale optimization," *IEEE Trans. Cybern.*, vol. 47, no. 9, pp. 2896–2910, Sep. 2017.

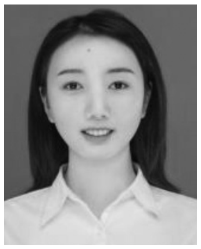
Xin Zhang (Member, IEEE) received the Ph.D. degree in computer science and technology from the South China University of Technology, Guangzhou, China, in 2020.

She is currently a Lecturer with the School of Artificial Intelligence and Computer Science, Jiangnan University, Wuxi, China. Her research interests include evolutionary computation, swarm intelligence, and their applications in large-scale optimization, supply chain network, and intelligent manufacturing.



Bo-Wen Ding received the B.E. degree in software engineering from the Nanyang Institute of Technology, Nanyang, China, in 2019. He is currently pursuing the B.S. degree in software engineering with Jiangnan University, Wuxi, China.

His research interests include evolutionary computation and its applications.



Xin-Xin Xu received the B.S. degree in computer science from Qilu Normal University, Jinan, China, in 2016, and the M.S. degree in computer science from Shandong Normal University, Jinan, in 2019. She is currently pursuing the Ph.D. degree in computer science with the Ocean University of China, Qingdao, China.

Her research interests mainly include computational intelligence, hyperheuristic optimization, dynamic multiobjective optimization, and their applications in real-world problems.

Jian-Yu Li (Student Member, IEEE) received the B.S. degree in computer science and technology from the South China University of Technology, Guangzhou, China, in 2018, where he is currently pursuing the Ph.D. degree in computer science and technology with the School of Computer Science and Engineering.

His research interests mainly include computational intelligence, data-driven optimization, machine learning, including deep learning, and their applications in real-world problems, and in environments of distributed computing and big data.

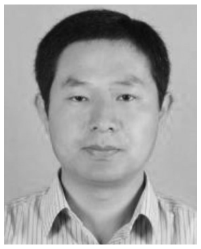
Zhi-Hui Zhan (Senior Member, IEEE) received the bachelor's and Ph.D. degrees in computer science from Sun Yat-sen University, Guangzhou, China, in 2007 and 2013, respectively.

He is currently the Changjiang Scholar Young Professor with the School of Computer Science and Engineering, South China University of Technology, Guangzhou. His current research interests include evolutionary computation, swarm intelligence, and their applications in real-world problems and in environments of cloud computing and big data.

Dr. Zhan was a recipient of the IEEE Computational Intelligence Society (CIS) Outstanding Early Career Award in 2021, the Outstanding Youth Science Foundation from the National Natural Science Foundations of China in 2018, and the Wu Wen-Jun Artificial Intelligence Excellent Youth from the Chinese Association for Artificial Intelligence in 2017. His doctoral dissertation was awarded the IEEE CIS Outstanding Ph.D. Dissertation and the China Computer Federation Outstanding Ph.D. Dissertation. He is one of the World's Top 2% Scientists for both Career-Long Impact and Year Impact in Artificial Intelligence and one of the Highly Cited Chinese Researchers in Computer Science. He is currently the Chair of the Membership Development Committee in IEEE Guangzhou Section and the Vice-Chair of the IEEE CIS Guangzhou Chapter. He is currently an Associate Editor of the IEEE TRANSACTIONS ON EVOLUTIONARY COMPUTATION, *Neurocomputing*, *Memetic Computing*, and *Machine Intelligence Research*.

Pengjiang Qian (Senior Member, IEEE) received the Ph.D. degree in information technology and engineering of light industry from Jiangnan University, Wuxi, China, in 2011.

He is currently a Full Professor with the School of Artificial Intelligence and Computer Science, Jiangnan University. His research interests include data mining, pattern recognition, bioinformatics, and their applications, such as analysis and processing for medical imaging and intelligent traffic dispatching.



Wei Fang (Member, IEEE) received the Ph.D. degree in information technology and engineering of light industry from Jiangnan University, Wuxi, China, in 2008.

He is a Professor of Computer Science with Jiangnan University. His current research interests involve the evolutionary computation.

Prof. Fang serves as an Editorial Board Member for the *International Journal of Swarm Intelligence Research* and *International Journal of Computing Science and Mathematics*.

Kuei-Kuei Lai received the Ph.D. degree in management science from Tamkang University, New Taipei. Taiwan.

He is currently a Professor with the Department of Business Administration, Chaoyang University of Technology, Taichung, Taiwan. He has authored more than 90 articles. His research interests include quantitative analysis, patent citation analysis, social networks analysis, technology strategy and technological forecasting, computational intelligence, and applications in management.

Jun Zhang (Fellow, IEEE) received the Ph.D. degree in electrical engineering from the City University of Hong Kong, Hong Kong, in 2002.

He is currently a Korea Brain Pool Fellow Professor with Hanyang University, Seoul, South Korea. His current research interests include computational intelligence, cloud computing, operations research, and power electronic circuits. He has published over more than 150 IEEE TRANSACTIONS papers in his research areas.

Dr. Zhang was a recipient of the Changjiang Chair Professor from the Ministry of Education, China, in 2013, the National Science Fund for Distinguished Young Scholars of China in 2011, and the First-Grade Award in Natural Science Research from the Ministry of Education, China, in 2009. He is currently an Associate Editor of the IEEE TRANSACTIONS ON CYBERNETICS and IEEE TRANSACTIONS ON EVOLUTIONARY COMPUTATION.