Incremental Recursive Ranking Grouping for Large-Scale Global Optimization

Marcin Michal Komarnicki, Michal Witold Przewozniczek, Halina Kwasnicka, and Krzysztof Walkowiak, Senior Member, IEEE

Abstract—Real-world optimization problems may have a different underlying structure. In black-box optimization, the dependencies between decision variables remain unknown. However, some techniques can discover such interactions accurately. In large-scale global optimization (LSGO), problems are high dimensional. It was shown effective to decompose LSGO problems into subproblems and optimize them separately. The effectiveness of such approaches may be highly dependent on the accuracy of problem decomposition. Many state-of-the-art decomposition strategies are derived from differential grouping (DG). However, if a given problem consists of nonadditively separable subproblems, DG-based strategies may discover many nonexistent interactions. On the other hand, monotonicity checking strategies proposed so far do not report nonexistent interactions for any separable subproblems but may miss discovering many of the existing ones. Therefore, we propose incremental recursive ranking grouping (IRRG) that suffers from none of these flaws. IRRG consumes more fitness function evaluations than the recent DG-based propositions, e.g., recursive DG 3 (RDG3). Nevertheless, the effectiveness of the considered cooperative co-evolution frameworks after embedding IRRG or RDG3 was similar for problems with additively separable subproblems that are suitable for RDG3. After replacing the additive separability with nonadditive, embedding IRRG leads to results of significantly higher quality.

Index Terms—Large-scale global optimization (LSGO), monotonicity checking, nonadditive separability, problem decomposition.

I. INTRODUCTION

In practice, many continuous large-scale global optimization (LSGO) problems emerge [1], [2], [3]. They were classified as complex continuous optimization problems [4]. Thus, solving them may be considered important but difficult. Real-value encoded instances are classified as large if they have at least 500 decision variables. However, it is frequent to consider the LSGO instances with at least 1000 variables [1], [4]. The size of the search space is exponentially proportional to the number of problem dimensions [4]. Thus, increasing the dimensionality may cause the exponential growth of the number of local optima [1] or the enormously high cost of running Estimation of Distribution Algorithms [5]. Therefore, tackling LSGO instances may be hard for Evolutionary Algorithms designed to solve less dimensional optimization problems.

In the state-of-the-art LSGO-dedicated optimizers, two main approaches are considered. The first, involves hybrid optimization methods, which do not utilize problem decomposition, e.g., success-history-based parameter adaptation for differential evolution with iterative local search (SHADE-ILS) [6]. The other approach is to decompose a problem first and then optimize its subproblems independently, frequently using cooperative co-evolution (CC) [7], [8], [9], [10]. Nowadays, the state-of-the-art problem decomposition strategies derive from differential grouping (DG) [11], [12], [13], [14], [15], [16]. These strategies assume that problem consists of additively separable subproblems. If this assumption is false, the DG-based strategies may report false linkage that takes place when two independent variables are found dependent [17]. False linkage may significantly decrease the quality of the decomposition and decrease the overall method effectiveness [18]. Decomposition strategies that do not assume the existence of only additively separable subproblems are based on monotonicity checking [19], [20], [21], [22]. However, they may miss finding some of the interactions (missing linkage [17]), and DG-based strategies where shown to be less vulnerable to this inconsistency [11], [13], [14], [22].

In this article, we propose incremental recursive ranking grouping (IRRG), a new problem decomposition strategy that does not assume the existence of only additively separable subproblems. IRRG is derived from monotonicity checking, thus, it never reports false linkage. Additionally, it significantly limits the issue of missing linkage. This article has two main objectives. First, we show that IRRG can decompose continuous LSGO problems accurately regardless of the additive or nonadditive separability. Second, we show that although IRRG is more expensive than recursive DG 3 (RDG3) [16], its influence on the overall optimization cost is negligible also when the problems are DG-suitable. To meet these objectives, we embed both, IRRG and RDG3, into two different CC frameworks. We consider a set of 45 benchmark problems, including additively and nonadditively separable ones. This set is built from the CEC’2013 functions [23]. Since the standard...
CEC’2013 set contains only one function with nonadditively separable subfunctions, we propose two transformations of the additive separability into nonadditive. Moreover, we analyze a real-world optimization problem with nonadditively separable subproblems.

The remainder of this article is organized as follows. The related work is discussed in the next section. In Section III, we compare DG and monotonicity checking. Section IV presents the details of IRRG, whereas the fifth section reports the results of the experiments. Finally, in the last section, we conclude this article and propose the directions of future work.

II. RELATED WORK

A. Problem Decomposition

The quality of the problem decomposition is related to the separability structure of its objective function [13]. A function \(f : \Omega \rightarrow \mathbb{R}\) is partially separable [24], i.e., consists of \(m\) independent subfunctions, \(2 \leq m \leq n\), if

\[
\arg \min_x f(x) = \left[ \arg \min_{x_1} f(x_1, \ldots), \ldots, \arg \min_{x_m} f(\ldots, x_m) \right]
\]

where \(x \in \Omega\) is a vector of \(n\) decision variables \(\{x_1, \ldots, x_n\}\), \(\forall i \in \{1, \ldots, m\}\) \(x_i \in \Omega_i\), and \(\Omega = \Omega_1 \times \cdots \times \Omega_m\). When \(m = n\) then each decision variable does not interact with the others and a function is fully separable.

Problem decomposition strategies include uniformed decomposition [25], random grouping [26], delta grouping [27], meta modeling [28], formula-based grouping [29], DG [11], [12], [13], [14], [15], [16], and monotonicity checking [19], [20], [21], [22]. The last two approaches are highly related to this article’s scope, thus, we describe them in detail.

1) Differential Grouping: Many real-world optimization problems are partially separable [30], e.g., additively separable [11]. The definition of an additively separable function is similar to (1): \(f(x) = \sum_{i=1}^{m} f_i(x_i)\). DG [11] is a problem decomposition strategy proven to group only dependent variables if a problem is additively separable. DG marks two variables as interacting if \(\exists a, b_1 \neq b_2, \delta > 0, x^* \in \Omega\) such that the following condition holds:

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

where \(x_p\) and \(x_q\) indicate the \(p\)th and \(q\)th problem variables, whereas the definition of function \(\Delta_{\delta, x_p} |f|\) is as follows:

\[
\Delta_{\delta, x_p} |f| (x) = f(\ldots, x_p + \delta, \ldots) - f(\ldots, x_p, \ldots)
\]

By \(f(x)|_{x_p=a, x_q=b}\) we mean the value of \(f\) after setting the values of \(x_p\) and \(x_q\) to \(a\) and \(b\), respectively. Each perturbation of the \(p\)th variable, i.e., \(x_p + \delta\), must produce a feasible solution. Without loss of generality, we may denote the left and right sides of (2) as \(\Delta_1\) and \(\Delta_2\), respectively. Due to the inaccuracy of the float numbers representation, a user-defined \(\epsilon\) is employed to control the sensitivity of DG: \(|\Delta_1 - \Delta_2| > \epsilon\) instead of \(|\Delta_1 \neq \Delta_2|\). Its appropriate value may be different for various optimization problems [13]. DG optimizes the number of pair-wise checks [the single interaction check requires four fitness function evaluations (FFEs)] by ignoring some pairs of variables. The worst scenario is when a problem is fully separable, then, \(2n(n-1)\) FFEs are needed. The minimal cost, for a fully nonseparable problem, is \(4(n-1)\) FFEs. Thus, the DG’s time complexity is \(O(n^2)\).

DG2 is an improved version of DG [13]. It decreases by half the FFE cost of decomposition for fully separable problems. Additionally, DG2 can detect the overlapping structure of a given optimization problem by checking all possible pairs of variables. In DG2, \(\epsilon\) is automatically computed as follows. Each interaction check requires fitness values of four solutions: \(x^*_1, x^*_2, x^*_3, \) and \(x^*_4\). Thus, \(|\Delta_1 - \Delta_2| > \epsilon\) can be reformulated as \(|f(x^*_1) - f(x^*_2)| - |f(x^*_3) - f(x^*_4)|| > \epsilon\), where

\[
\epsilon = f_x(\sqrt{n} + 2 \cdot (|f(x^*_1)| + |f(x^*_2)| + |f(x^*_3)| + |f(x^*_4)|))
\]

where \(\mu_M\) is a machine-dependent constant [31].

2) Recursive Differential Grouping: Recursive DG (RDG) [14] reduces the complexity of DG2 from \(O(n^2)\) to \(O(n \log(n))\). DG and DG2 perform the pair-wise interaction check, whereas RDG consider two disjoint groups of variables \(X_1\) and \(X_2\) that are subsets of \(X = \{x_1, \ldots, x_n\}\). Groups interact if at least one pair of variables \(x_p \in X_1\) and \(x_q \in X_2\) is nonseparable. Then, there exist such values \(\delta_1, \delta_2 > 0,\) unit vectors \(u_1 \in U_{X_1}, u_2 \in U_{X_2}\), and decision vector \(x^* \in \Omega\) that meet

\[
f(x^* + \delta_1 u_1 + \delta_2 u_2) - f(x^*) \neq f(x^* + \delta_2 u_2) - f(x^*)
\]

where \(u_j = [u_1, \ldots, u_n] \in U_X\) such that \(\forall i \in \{1, \ldots, n\} u_i = 0 \Leftrightarrow x_i \notin X_j\). Thus, RDG is executed recursively, dividing subsequent groups of decision variables by half until a one-element group is reached or the separability is discovered.

RDG3 [16], introduces two new thresholds, i.e., \(\epsilon_a\) and \(\epsilon_n\). It tries to join all separable variables into groups of size \(\epsilon_n\). The \(\epsilon_n\) threshold is useful for overlapping problems. The variables of such problems are divided into groups of approximately \(\epsilon_n\) size instead of creating one group. CC frameworks embedding RDG3 were shown more effective than those using its predecessors [16].

3) Monotonicity Checking: Monotonicity checking strategies [19], [20], [21] do not assume that a problem is additively separable. Two decision variables \(x_p\) and \(x_q\) are interacting if \(\exists a_1 \neq a_2, b_1 \neq b_2, x^* \in \Omega\) such that

\[
\begin{align*}
|f(x^*)|_{x_p=a_1, x_q=b_1} &\leq |f(x^*)|_{x_p=a_2, x_q=b_2} \\
|f(x^*)|_{x_p=a_1, x_q=b_2} &> |f(x^*)|_{x_p=a_2, x_q=b_1}
\end{align*}
\]

Fast variable interdependence learning (FVIL) [22] replaces this pair-wise interaction check by examining two disjoint groups of decision variables \(X_1\) and \(X_2\). They are interacting if \(\exists \delta_1, \delta_2 > 0, u_1 \in U_{X_1}, u_2 \in U_{X_2}, x^* \in \Omega\) such that

\[
f(x^*) \leq f(x^* + \delta_1 u_1) \wedge f(x^* + \delta_2 u_2) > f(x^* + \delta_1 u_1 + \delta_2 u_2).
\]

To check if \(X_1\) and \(X_2\) interact, randomly created \(\delta_1, \delta_2, u_1, u_2,\) and \(x^*\) are used. This procedure is repeated at most \(N\)
times. As in RDG, subsequent groups of variables are being recursively divided. Thus, the complexity is also $O(n \log(n))$.

B. Cooperative Co-Evolution and Hybrid Optimization

The idea behind CC is to optimize each component (subproblem) separately. Thus, the decomposition strategies are useful for CC [7]. The decomposition quality is not the only factor that influences the effectiveness of CC—the lower quality decomposition may lead to better results [16]. In the original CC, the expenses for optimizing each component are similar. Since different subproblems may have a different (high or low) impact on global fitness improvements [8], it seems reasonable to concentrate the optimization on those components that will have the highest impact on fitness.

Contribution-based CC (CBCC) [8], [10] computes an accumulated contribution $\Delta F_i$ of the $i$th component considering all previous fitness improvements. Improvements found earlier have a lower impact on the accumulated contribution. CBCC does not require components of equal sizes. A subpopulation of a different size may optimize each component, but the FFE budget of the single run is the same for each component. In CCFR2 (another extended CC framework) [9], each subpopulation (that may be of a different size as in CBCC) is executed every time for the same number of iterations. Additionally, in CCFR2, the accumulated contribution considers the computational cost of finding the improvement.

In CBCC and CCFR2, we provide them the decomposition of a considered problem and execute separate optimization processes for separate components. In hybrid methods, we search for the most appropriate optimizer for optimizing all variables together or their random subset [6], [32], [33]. SHADE-ILS [6] hybridizes SHADE [34] and two local search methods. SHADE is used to explore the search space, while a local search is used to improve the quality of promising solutions. SHADE-ILS employs MTS-L1 [35] and L-BFGS-B [36] that have complementary pros and cons. MTS-L1 is effective for separable problems, while L-BFGS-B approximates the gradient and is more robust. Additionally, SHADE-ILS restarts itself when the improvement ratio is low.

III. DIFFERENTIAL GROUPING AND MONOTONICITY CHECKING—ANALYSIS OF MAIN DIFFERENCES

This section presents a comparison between DG and monotonicity checking. We focus on functions for which at least one of these two strategies reports false or missing linkage. To this end, we adapt the definition of the variable separability from [17] to continuous domains. For the sake of clarity, we consider a two-variable function $f$ and two univariate subfunctions that simplify $f$ by replacing the second variable by a constant value: $\tilde{g}(x) = f(x)|_{x_2 = b_1}$ and $\tilde{h}(x) = f(x)|_{x_1 = a_1, x_2 = b_2}$, where $b_1$ and $b_2$ are two different constant values. Conclusions drawn for a two-variable function can be easily generalized to more variables.

A. Monotonicity Checking as Empirical Linkage Learning

According to (1), and taking subfunctions $\tilde{g}$ and $\tilde{h}$ into account, we can state that if $\arg\min_{x_1} \tilde{g}(x) \neq \arg\min_{x_1} \tilde{h}(x)$, then $x_1$ and $x_2$ are interacting. Otherwise, although the global optimum is not changed, it is not certain that $x_1$ and $x_2$ are separable, because only two different values of $x_2$, i.e., $b_1$ and $b_2$, have been considered. After setting $x_2$ to $b_3$ and $b_4$, such that $b_3 \neq b_1 \land b_4 \neq b_2$, the condition $\arg\min_{x_1} \tilde{g}(x) \neq \arg\min_{x_1} \tilde{h}(x)$ could be met. This kind of interaction condition can omit fitness landscape characteristics [37], which may significantly affect the optimizer effectiveness [38], [39].

The provided examples show that even if two variables are possibly separable [according to (1)], an optimizer may behave differently depending on which subfunctions will be taken into account. Thus, the condition comparing only global optimum changes to distinguish between the separability and the non-separability seems insufficient. In the example presented in Fig. 2, we consider a greedy optimizer that starts its search in point $S^g$ or $S^h$ for subfunctions $\tilde{g}$ and $\tilde{h}$, respectively. Since $\tilde{g}$ and $\tilde{h}$ have the same global optimum, then $x_1$ and $x_2$ are possibly separable according to (1). Let us consider the following candidate solutions, $S^g_1$ and $S^g_2$ for subfunction $\tilde{g}$, and $S^h_1$ and $S^h_2$ for subfunction $\tilde{h}$. Double circles indicate the better candidate solution for $\tilde{g}$ and $\tilde{h}$. In Fig. 2(a), the choice is the same for $\tilde{g}$ and $\tilde{h}$. Thus, $x_1$ and $x_2$ may be, indeed, separable. In Fig. 2(b) the choice is different for each subfunction. The optimization process of $\tilde{g}$ and $\tilde{h}$ may result in different points in the search space. Therefore, it seems reasonable to state that the variables $x_1$ and $x_2$ are interacting. For more complex subfunctions, an optimizer may be deceived to a local optimum only for one subfunction, although both have the same global optimum. The selection pressure favors better-fitted individuals. Thus,
a single change in the outcome of the individuals’ fitness comparison may influence a direction of the search [40].

The above remarks are consistent with the idea of empirical linkage learning (ELL) [17], a new class of linkage learning techniques. In ELL, interactions are discovered by comparing local search results before and after perturbing one of the decision variables. Two variables interact if a local search method converges to a different value after perturbing one of them. Such techniques are proven never to report false linkage. The formal proof assumes the following definition of the variable separability [17]. Two disjoint sets of variables $X_1$ and $X_2$ are independent if for each values $\delta_1$, $\delta_2 > 0$, unit vectors $u_1 \in U_{X_1}$, $u_2 \in U_{X_2}$, and decision vector $x^* \in \Omega$ the following condition holds: $f(x^*) < f(x^* + \delta_1 u_1) \Leftrightarrow f(x^* + \delta_2 u_2)$ and $f(x^*) = f(x^* + \delta_1 u_1) \Leftrightarrow f(x^* + \delta_2 u_2)$ on the premise that all arguments of $f$ are feasible solutions. In the latter part of this article, unless stated otherwise, we consider the above (ELL-based) definition of separability. Thus, if exists such $\delta_1$, $\delta_2$, $u_1$, $u_2$, and $x^*$ that $f(x^*) \leq f(x^* + \delta_1 u_1)$ and $f(x^* + \delta_2 u_2) > f(x^* + \delta_1 u_1 + \delta_2 u_2)$, then $X_1$ and $X_2$ sets are interacting. Thus, monotonicity checking strategies may be classified as ELL techniques. In [20], they are also identified as strategies that do not report false linkage.

**B. Decomposition Inaccuracies: False and Missing Linkage**

DG-based decomposition strategies are proven that they will never report false linkage if a function is additively separable [11], [14]. A function is additively separable if every pair of its subfunctions is also additively separable. Let $f_1$ and $f_j$ be separable subfunctions. They are additively separable if $\forall \delta_1, \delta_2 > 0, x_p, x_q \in \Omega$, and $x_1, x_2$ are arguments of subfunctions $f_1$ and $f_j$, respectively, [14]. Otherwise, $f_1$ and $f_j$ are nonadditively separable. If a function is nonadditively separable, i.e., every pair of its subfunctions is also nonadditively separable, DG-based decomposition strategies may report false linkage. Let us consider the minimization of the following function: $\tilde{f}_{c,1} : [-5, 5]^2 \rightarrow [0, 100]$ defined as $\tilde{f}_{c,1}(x) = (|x_1| + |x_2|)^2$. A greedy optimizer that is deterministic (e.g., with a fixed value of the random seed) applied to optimize $x_1$, for any value of $x_2$, will converge to the same value and vice versa. If this greedy optimizer has infinite precision, it will converge to 0 for both variables. Therefore, $\tilde{f}_{c,1}$ is fully separable. However, DG-based strategies will find $x_1$ and $x_2$ dependent, because using (2), we get $\Delta_1 = 5$ and $\Delta_2 = 7$ for $(a, a + \delta, b_1, b_2) = (1, 2, 1, 2)$.

On the other hand, monotonicity checking never reports false linkage (see Section III-A). However, in practice, due to the inaccuracy of the representation of float numbers, the appropriate value of $\epsilon$ is necessary to determine if the inequalities are satisfied [see (7)]. Then, monotonicity checking strategies will not report false linkage also for problems with nonadditively separable subproblems. DG-based strategies do not have this advantage (and also suffer from the floating-point error). However, the results presented in [11], [13], [14], and [22] show that DG-based strategies are more accurate. The reason may be that monotonicity checking strategies proposed so far are more vulnerable to missing linkage. Let us consider more examples that employ function $f$, and its subfunctions $\tilde{g}$ and $\tilde{h}$, defined at the beginning of this section. We simplify (2) to $\tilde{g}(a + \delta) - \tilde{g}(a) \neq \tilde{h}(a + \delta) - \tilde{h}(a)$ that is equivalent to $\Delta_1 \neq \Delta_2$. Similarly, we simplify (3) to $\tilde{g}(a_1) \leq \tilde{g}(a_2)$ and $\tilde{h}(a_1) > \tilde{h}(a_2)$.

Fig. 3 presents another example in which DG may report false linkage. We multiply a function by a positive number, which does not change relations between its outputs, i.e., the relation between function outputs (smaller, greater, and equal) for any two arguments will be the same for the original and multiplied functions. In Fig. 3(a), $\Delta_1$ and $\Delta_2$ are indicated by vectors for various values of $a$ spread along the horizontal axis and the same value of $\delta$. For at least one pair $(\Delta_1, \Delta_2)$ the referring vectors differ. Thus, although, the considered problem is separable, DG may detect (nonexisting) interaction between $x_1$ and $x_2$. Oppositely, monotonicity checking will consider the monotonicity intervals [Fig. 3(b)], which are the same for $\tilde{g}$ and $\tilde{h}$. Note that it is a necessary, but not sufficient, condition. Hence, we can use another condition, this time sufficient. If $\tilde{h}(x) = \alpha \cdot \tilde{g}(x)$, where $\alpha > 0$, then $\forall a_1, a_2 (\tilde{g}(a_1) > \tilde{g}(a_2) \vee \tilde{h}(a_1) \leq \tilde{h}(a_2))$, because $\forall a_1, a_2 (\tilde{h}(a_1) \leq \tilde{h}(a_2) \Leftrightarrow \tilde{g}(a_1) \leq \tilde{g}(a_2))$. Therefore, monotonicity checking will report $x_1$ and $x_2$ as independent.

One of the main advantages of DG is its high resistance to missing linkage. In Fig. 4(a), we show that the probability of selecting $a$ that leads to the dependency discovery is high. Monotonicity checking will always report $x_1$ and $x_2$ as dependent only if $a_1$ and $a_2$ come from the middle interval [Fig. 4(b)]. Otherwise, the interaction may not be discovered.

---

1 See the supplementary material for examples and additional explanations.
Feasible values of decision variables may also influence monotonicity checking. Let us consider the decomposition of \( \tilde{f}_{c,2} : [-8, 8] \times [-2, 2] \to [0, 100] \) defined as \( \tilde{f}_{c,2}(x) = (x_1 + x_2)^2 \). For \((a_1, a_2, b_1, b_2) = (-2, -1, 2, 1)\), monotonicity checking will discover the dependency between \( x_1 \) and \( x_2 \). However, if \( a_1 = -6 \) and \( a_2 = -5 \), then \( \forall b_1, b_2 \in [-2, 2] \), \( \tilde{f}_{c,2}(a_1, b_1) > \tilde{f}_{c,2}(a_1, b_2) \). Thus, monotonicity checking will not find the dependency for any feasible values of \( b_1 \) and \( b_2 \). Note that \( \tilde{f}_{c,2} \) is fully nonseparable, because the value of \( \tilde{f}_{c,2} \) is minimal when \( x_1 = -x_2 \). Finally, DG-based strategies should discover the interaction between \( x_1 \) and \( x_2 \) easily.

### IV. INCREMENTAL RECURSIVE RANKING GROUPING

Decomposition strategies based on DG and monotonicity checking may report false or missing linkage, respectively. Therefore, we propose IRRG employing the idea of monotonicity checking. IRRG never reports false linkage and significantly mitigates the issue of missing linkage. First, we describe recursive ranking grouping (RRG) that is a key part of our proposition, then we present the general view of IRRG. Finally, we compare IRRG with other monotonicity checking strategies and discuss the time complexity of IRRG.

#### A. Basic Concept

The simplest way to decrease the level of missing linkage in monotonicity checking is to create uniformly and randomly a huge number of samples. A single sample is defined by \((a_1, a_2, b_1, b_2, x^*) \) and by \( \delta_1, \delta_2, u_1, u_2, x^* \) for (7) and (8), respectively. Such an approach increases the probability that at least one sample will result in dependency discovery if two variables or two sets of variables indeed interact. However, its cost may not be reasonable for the optimization process.

To increase the sensitivity of monotonicity checking, IRRG creates two rankings of samples. Each ranking consists of \( n_s \) (user-defined parameter) samples. To check if variables \( x_p \) and \( x_q \) interact, we need \( b_1, b_2 \) (\( b_1 \neq b_2 \)), \( x^* \), and \( n_s \) values that are evenly taken from the feasible set of \( x^* \). Let us denote these \( n_s \) values as \( \tilde{a}_1, \ldots, \tilde{a}_{n_s} \). Then, we create two rankings \( \mathbf{r}_1 = [r_{1,1}, \ldots, r_{1,n_s}] \) and \( \mathbf{r}_2 = [r_{2,1}, \ldots, r_{2,n_s}] \), where \( r_{j,i} \) is computed for \( f(x^*) |_{x_p=\tilde{a}_i, x_q=b_j} \). If \( \exists r_{1,i} \in [1, n_s] \), \( r_{1,i} \neq r_{2,j} \) then variables \( x_p \) and \( x_q \) interact. For instance, for \( \tilde{f}_{c,3} : [-3, 3]^4 \to [-111, 111] \) defined as \( \tilde{f}_{c,3}(x) = (x_1 + x_2)^2 \cdot x_3 + x_4, p = 1, q = 2, b_1 = 1, b_2 = 2, x^* = [1, 0, 3, 2] \), and \( n_s = 3 \), we get \( \tilde{a}_1 = -3, \tilde{a}_2 = 0, \) and \( \tilde{a}_3 = 3 \). Thus, \( \tilde{f}_{c,3}(x^*) |_{x_3=-3, x_4=1} = 14, \tilde{f}_{c,3}(x^*) |_{x_3=0, x_4=1} = 5, \tilde{f}_{c,3}(x^*) |_{x_3=3, x_4=1} = 50, \) and \( \tilde{f}_{c,3}(x^*) |_{x_3=-3, x_4=2} = 5 \). Since \( r_{1,1} \neq r_{2,1}, r_1, x_1, x_2 \) are dependent. Comparing rankings may be found equivalent to searching for \( a_1, a_2, b_1, b_2, \) and \( x^* \) that satisfy (7).

Using (8), we define the recursive ranking check that can detect interaction between two disjoint sets of variables \( X_1 \) and \( X_2 \). First, \( n_s \) values for each \( x_j \in X_1 \), namely, \( \tilde{a}_{1,i}, \ldots, \tilde{a}_{n_s,i} \) are evenly generated from the \( x_j \) feasible set. Then, we define \( n_d \) decision vectors \( \mathbf{x}_i = [\tilde{x}_1, \ldots, \tilde{x}_{n_d}] \) based on a given vector \( x^* = [x_1^*, \ldots, x_{n_d}^*] \), where

\[
\tilde{x}_{j,i} = \begin{cases} \tilde{a}_{j,i} & x_j \in X_1 \\ x_j^* & x_j \notin X_1 \end{cases}
\]

where \( \pi_j \) denotes a random permutation of \([1, \ldots, n_d]\) for the \( j \)th variable. These vectors are then used to generate the rankings. Using all \( n_s \) available vectors is impossible due to practical reasons. The random permutation is used to overcome the possible bias. The \( j \)th value of the first ranking \( r_{1,i} \) is based on \( f(\tilde{x}_i) \), whereas \( r_{2,i} \) is computed for \( f(\tilde{x}_i + \delta_2 u_2) \). Analogously to the pair-wise interaction check, if \( \exists r_{1,i} \in [1, n_s] \), \( r_{1,i} \neq r_{2,i} \) then the sets \( X_1 \) and \( X_2 \) interact, because for at least one \( i^* \in [1, \ldots, n_d] \), condition \( f(\tilde{x}_{i^*,1}) \leq f(\tilde{x}_{i^*,2}) + \delta_2 u_2 \). Note that this condition specifies (8).

#### B. Recursive Ranking Grouping

RRG is the main part of IRRG. It discovers dependencies between disjoint groups of variables. First, we define a set of functions that will be used in the description of RRG. A set of groups of variables that were already found interacting will be denoted as \( G \), while \( V \) is a set of variables for which no interactions were discovered yet. In matrix \( \mathbf{X}_1 \) of size \( n_s \times n \), each \( j \)th column consists of \( n_s \) randomly ordered values that are evenly generated from the feasible set of the \( j \)th variable.

Function \texttt{ConsiderVariables} (Pseudocode 1) decides if the variables in \( V \) should be considered in the interaction check. It will happen if one of the following conditions holds. 1) \( G \) is empty, i.e., no dependencies were discovered yet (line 2). 2) \( V \) contains only one variable (line 2). 3) \( V \) is randomly divided into two disjoint subsets \( V_1 \) and \( V_2 \). The sizes of both are equal or differ by one. If we detect interaction between \( V_1 \) and \( V_2 \), then the variables in \( V \) will be considered in the interaction search (lines 6–14). 4) \( V \) interacts with at least one group from \( G \) (lines 15–21). Function \texttt{ConsiderVariables} uses \texttt{CreateFirstRanking} (Pseudocode 2) that creates a ranking using the concept presented in Section IV-A. Except for ranking \( \mathbf{r}_1 \), this function returns a vector of values \( \mathbf{y}_1 \) for which the ranking was computed. \( \mathbf{y}_1 \) is used by other functions defined in this section.

Function \texttt{IsInteraction} (Pseudocode 3) is used to check the dependency between two groups of variables denoted as \( X_1 \) and \( X_2 \). To this end, we create ranking \( \mathbf{r}_2 \) and check if it differs from \( \mathbf{r}_1 \). To save computational effort, the order of ranking \( \mathbf{r}_2 \) is checked during its creation.
**Pseudocode 1** CONSIDERVARIABLES; Checking if It Is Worth Considering Separable so Far Variables in the Interaction Search

```plaintext
input: V: variables to check, G: groups of nonseparable variables, x, a high-quality  
decision vector, X: a n × n matrix of samples' values, x: a decision vector different than x,  
if |G| ≥ 1: if at least one variable from V is not separable  
output: a decision if at least one variable from V can be separable  
1. function CONSIDERVARIABLES(V, G, x, X, x, f, ns)  
   2. if |G| = 0 or |V| = 1 then  
      3. return true  
   4. if |V| = 0 then  
      5. return false  
   6. V ← shuffle V  
   7. V ← the first half of V  
   8. V ← the second half of V  
   9. ISINTERACTION(V, x, X, x, f, ns)  
   10. if ISINTERACTION(V, x, X, x, f, ns) then  
      11. return true  
   12. (y1, r1) ← CREATEFIRSTRANKING(V, x, X, x, f, ns)  
   13. if ISINTERACTION(V, x, X, x, f, ns) then  
      14. return true  
   15. for i ← 1 to |G| do  
      16. (y1, r1) ← CREATEFIRSTRANKING(V, x, X, x, f, ns)  
      17. if ISINTERACTION(V, x, X, x, f, ns) then  
      18. return true  
   19. (y1, r1) ← CREATEFIRSTRANKING(V, x, X, x, f, ns)  
   20. if ISINTERACTION(V, x, X, x, f, ns) then  
      21. return true  
   22. return false
```

**Pseudocode 2** CREATEFIRSTRANKING

```plaintext
input: X: set of variables, x: a decision vector, X: a n × n matrix of samples' values, f: an optimization problem, n: the number of samples  
output: y: values of f calculated based on x and X, r1: the ranking of f  
1. function CREATEFIRSTRANKING(X, x, X, f, ns)  
   2. x ← x  
   3. for i ← 1 to n do  
      4. s[Xi] ← X[i][i]  
      5. y[i] ← f(s)  
   6. r1 ← indices i from 1 to n ordered by y[i]  
   7. return (y, r1)
```

**Pseudocode 3** ISINTERACTION; Checking if Two Sets of Variables Are Interacting

```plaintext
input: X1, X2: disjoint sets of variables, x: a high-quality decision vector, X: a n × n matrix of samples' values, x: a decision vector different than x, X: values of f calculated based on x, f: an optimization problem, n: the number of samples  
output: a decision if X1 and X2 are interacting  
1. function ISINTERACTION(X1, X2, x, X, x, X, f, ns)  
   2. x ← x  
   3. s[X1] ← X[X1]  
   4. s[X1] ← X[X1]  
   5. y[i] ← f(s)  
   6. for i ← 2 to n do  
      7. e1 ← f(x) + 1) · (y[i] − y[i − 1]) + (y[i] − y[i − 1])  
   8. if sgn(y[i][i]) = y[i][i − 1], e1 = 0 then  
   9. continue  
   10. s[X1] ← X[X1]  
   11. y[i] ← f(s)  
   12. e2 ← f(x) + 1) · (y[i] − y[i − 1])  
   13. if sgn(y[i][i]) = y[i][i − 1], e2 < 0 then  
   14. return true  
   15. return false
```

Similarly to DG-based strategies [11], [12], [13], [14], [15], [16], the result of this comparison of rankings may be affected by the inaccuracy of the representation of float numbers. Instead of using a user-defined ε, we employ the automatic estimation of its value derived from [15] (lines 7 and 12). If we denote two vectors as x and x, then, analogously to (4) [15], the estimation of ε is computed as

\[
\epsilon = f(x, \sqrt{n} + 1) \cdot (f(x) + f(x))
\]

To decide interactions, we use n values for X1 and only two values for X2. Thus, ISINTERACTION is not a symmetric function. We call it twice in CONSIDERVARIABLES for each pair of sets of variables we examine (lines 10, 13, 17, and 20).

High-quality solutions occupy local optima or are close to them. Therefore, if the interaction exists, then it should reveal mainly while observing high-quality solutions [17]. To achieve this, we start using the function values around x and x from the samples of the highest quality in terms of R1. We believe that the best samples of different rankings may correspond to different optima if an interaction exists. Additionally, to decrease the probability that too small ε causes false linkage, we discover an interaction only if both results of signum function (10) are decisive (lines 8 and 13).

To discover interactions, we use n3 values for X1 and only two values for X2. Thus, ISINTERACTION is not a symmetric function. We call it twice in CONSIDERVARIABLES for each pair of sets of variables we examine (lines 10, 13, 17, and 20).

In the last paragraph of Section III-B, we show that constraints, which are a part of constrained optimization problems, e.g., bounding-box constraints, may prevent monotonicity checking from discovering interactions. To this end, we prefer to use a high-quality solution x that was optimized before RRG execution. The motivation behind this step is as follows. If x is close to a local optimum and X1, X2 interact, then the following situations are possible. 1) x changed, but the relations between function values around x changed [see Fig. 2(b)]. In such a situation, RRG is likely to discover interaction. 3) The monotonicity intervals and the relations of function values around x remain the same. RRG will not discover interaction in this case, but such a situation is not likely if X1 and X2 interact. Additionally, the number of local optima also influences the RRG sensitivity. The higher number of local optima, the higher number of points from different basins of attraction that have a similar interaction discovery potential as x. Thus, RRG will have a higher chance of interaction discovery.

Function INTERACT (Pseudocode 4) extends function ISINTERACTION. Its objective is to search for groups of variables in G2 that interact with at least one group from G1. The output of INTERACT is a set of groups G2 such that G1 ⊆ G2. Initially, G1 is a copy of G1 (line 2). To check if G1 and G2 interact, they are flatten (let S = {S1, . . . , Sk} be a set of k sets, then flattening S results in a set S = {S1 ∪ . . . ∪ Sk}) to X1 = X1 ∪ G1 and X2 = X2 ∪ G2 (lines 3 and 4), and ISINTERACTION is called (line 5). If an interaction is discovered, G2 is divided into two subsets (preferably of equal size) G2 and G2 (lines 9 and 10). INTERACT is then called recursively for G2 and G2 (lines 11 and 12). Once an interaction is found for G2 containing only one element (group), then this single group is added to G1 (line 7).
The general procedure of RRG is presented in Pseudocode 5. RRG requires providing the interaction matrix $\Theta$, which single element $\theta_{ij} \in \{0, 1\}$ informs if two variables, $x_p$ and $x_q$, are dependent. $\Theta$ may be represented by a graph. Finding the connected components of this graph leads to identifying groups of interacting variables [13]. Considering remarks to $\text{ISINTERACTION}$ presented above, we assume that at least one provided decision vector is of high quality. First, using $\Theta$, we create groups of interacting variables. The order of variables within groups is random to remove possible bias (lines 5–10). Then, according to the description presented in Section IV-A, we create $\bar{X}_1$ (lines 11–14). Using $\text{CONSIDERVARIABLES}$ we decide if variables that were not found interacting yet, shall take part in the interaction discovery (line 16). We shuffle $G$ to remove possible bias and create $G_1$ and $G_2$ (lines 17–19).

The latter part of RRG focuses on interaction discovery (lines 20–40). We create ranking $r_1$ and call $\text{INTERACT}$ to find new interactions between groups of variables from $G_1$ and $G_2$. If interactions were found, we add dependent groups to $G_1$, remove them from $G_2$ (lines 37 and 38) and call $\text{INTERACT}$ again for the extended $G_1$. If no new interactions were found, but $G_1$ was extended and contains more than one group, then $G_1$ is flattened and added as a single group to the output set of groups of interacting variables, $G_1$ is replaced with a single group that is excluded from $G_2$ (it is equivalent to the random choice, because $G_2$ was shuffled) (lines 33–35). If no extensions to the group initially inserted to $G_1$ were discovered (i.e., $G_1$ contains only one group after calling $\text{INTERACT}$), then we randomly remove half of the variables from the single group in $G_1$ (lines 26–28). The intuition behind this step is as follows. If $\bigcup G_1$ is large, then some subgroup of variables in $\bigcup G_1$ may influence rankings $r_1$ and $r_2$ so significantly that the influence of the variables in $\bigcup G_2$ will become negligible. Thus, if we remove some of the variables from a single group in $G_1$ randomly, then some of the overwhelmed variables in $\bigcup G_2$ may gain their influence on $r_1$ and $r_2$ rankings, resulting in the interaction discovery and mitigating missing linkage.

Usually, the situation described above occurs when most interactions are discovered. Thus, it seems reasonable to decrease the size of a single group in $G_1$ only if its size is not less than the size of the shortest group in $G_2$. The above procedure mitigates missing linkage and does not seem computationally expensive.

C. Incremental Grouping

IRRG builds the interaction matrix $\Theta$ incrementally. The general IRRG procedure is presented in Pseudocode 6. First, a user-defined optimizer is executed to find $x_{\text{high}}$ (a high-quality solution) (line 2), and the interaction matrix $\Theta$ is initialized as reporting no interactions (line 3). Then, RRG is called to get $\text{NonSeptsTemp}$—the set of groups of interacting variables (line 9). This set is used to update $\Theta$ by inserting the newly discovered interactions (line 10). The $\Theta$ updates ensure transitivity. These operations are repeated until RRG will not find any new interaction in $x_{\text{high}}$ (a user-defined parameter) iterations in a row or if RRG did not discover any new interactions during the first iteration (line 13). The latter part of this condition was introduced to decrease the computational cost of IRRG. The motivation is as follows. The lower is the number of interactions in $\Theta$, the easier is to supplement it. Thus, if RRG can
Pseudocode 6 IRRG

| Line | Description |
|------|-------------|
| 1:   | input: | f: an optimization problem, n: the problem size, lb: the lower bounds of f, ub: the upper bounds of f, optimizer: using during the initial optimization, nq: the number of samples, s: the preferred size of groups of separable variables, s: the state iterations threshold |
| 2:   | output: | Seps: groups of separable variables, NonSeps: groups of nonseparable variables |
| 3:   | 1) an initial optimizer; 2) the number of samples, s; 3) the stale iterations threshold |
| 4:   | if any new interactions were not discovered then |
| 5:   | see line 10 |
| 6:   | else |
| 7:   | firstIter ← false |
| 8:   | while not terminate do |
| 9:   | if any new interactions were not discovered then |
| 10:  | see line 10 |
| 11:  | else |
| 12:  | cntsti ← cntsti + 1 |
| 13:  | terminate ← firstIter or cntsti = εst |
| 14:  | return Seps, NonSeps |

not discover any new interactions in the first iteration, then repeating it does not seem reasonable.

IRRG returns two sets of variable groups. NonSeps contains groups of interacting variables, whereas in Seps, separable variables are joined into groups of a user-defined size εs (lines 26–32). This step is adopted from RDG3 [16] to improve separable variables processing by CC frameworks.

IRRG can be tuned by adjusting values of four parameters: 1) an initial optimizer; 2) the number of samples nq; 3) the state iteration threshold εsti; and 4) the preferred size of groups of separable variables s. The optimizer is executed to find a high-quality solution xhq. In case of LSGO problems, especially partially separable ones, we recommend exploring the search space first and then exploiting the found promising region. Otherwise, high-quality solutions for some subproblems may not be found. However, for some problems, the initial optimization will not be helpful and will not lead to obtaining better decomposition. The cost of this initial optimization should be reasonable, e.g., at most 2 · nq · FFEs, which corresponds to the lowest time complexity of IRRG. The higher the values of nq and εsti, the higher the probability of obtaining better decomposition. However, the decomposition cost increases at the same time. The discussion about εs can be found in [16].

D. Comparison With Other Monotonicity Checking Strategies

IRRG and FVIL are recursive monotonicity checking strategies. Both use (7) to decide if two groups of variables (X1 and X2) are interacting. If the interaction is discovered, X2 is divided into two (nearly) equally sized groups, X1 and X2.

Another difference that causes FVIL to be more vulnerable to missing linkage is the size of X1. In FVIL, X1 is always a single-element group. In IRRG, variables found dependent are added to X1 in subsequent interaction checks. Subsequent iterations of IRRG also try to extend the already found groups of interacting variables. Such interaction checks using larger X1 decrease the probability of missing linkage. Finally, FVIL does not apply the initial optimization. As explained in Section IV-B, using xhq may significantly mitigate missing linkage for some optimization problems.

Some monotonicity checking strategies employ an optimizer during interaction search [19], [20], e.g., CC with variable interaction learning (CCVIL) [20]. CCVIL executes an optimizer for each variable in each interaction discovery phase. However, the optimizer is executed only for one iteration, and its population is limited to only three individuals. Thus, we may expect that xhq employed by IRRG is of significantly higher quality.

E. Time Complexity

IRRG is a recursive decomposition strategy, similar to RDG-based strategies [14], [15], [16]. The typical time complexity (considering FFEs) for this class is $O(n \log(n))$. In this article, we consider LSGO, thus, we may expect that for large n, the cost of initial optimization (held to obtain xhq) is negligible. Thus, we ignore this part of IRRG in the analysis presented below. We refer to the analogous analysis of RDG [14]. The creation of rI ranking costs nq FFEs in each IRRG iteration. The cost of a single INTERACT execution is not higher than nq FFEs (the analogous operation in RDG requires three FFEs [14]). Thus, the cost of decomposing n-dimensional problem in the first iteration of IRRG is as follows.

1) Fully separable problems—about nq · n + nq · n FFEs.
2) Fully nonseparable problems—at most 2 · nq · n + nq FFEs.
3) Partially separable problems with m subproblems of size l—at most 2 · nq · n · log2(n) + nq · m FFEs.
4) Partially separable problems with one nonseparable subproblem of size l—at most nq · (n − l) + 2 · nq · l · log2(n) + nq · (n − l + 1) FFEs.
5) Rosenbrock function [23] that is an overlapping problem—about 2 · nq · n · log2(n) + nq · n/2 FFEs.

For consecutive IRRG iterations, we propose the similar analysis. Let us denote the number of variables that
were not found interacting yet as $n_V$ and the number of already discovered groups of interacting variables as $n_G$. Thus, $n_V + n_G < n$. If all possible interactions were already discovered, then similar to fully separable problems, approximately $2 \cdot n_V \cdot (nV + nG)$ additional FFEs are needed. Otherwise, since $n_V + n_G < n$, each consecutive IRRG iteration will still use less FFEs than the first iteration.

RRG includes the operation of removing half of variables from the single group belonging to $G_i$ (Pseudocode 5, line 28). The number of variable removals will be the highest when $\| G_1 \| = n - 1 \land \| G_2 \| = 1$ and will not exceed $\log_2(n-1)$. The cost of each cut is not higher than $2 \cdot n_V$ FFEs, $n_s$ FFEs for the creation of ranking $r_1$ and another $n_s$ FFEs for INTERACT. Thus, the total cost of each cut is below $2 \cdot n_s \cdot \log_2(n-1)$ FFEs. In the worst scenario, the cost of $\text{CONSIDER-} \text{VARIABLES}$ resulting from calling $\text{IS-INTERACT}$ and creating ranking $r_1$ twice for single variables and twice for each group of interacting variables is approximately $4 \cdot n_s + 4 \cdot n_s \cdot nG$ FFEs. Therefore, if $n_s \ll n$ and the number of all IRRG iterations is much smaller than $n$, the time complexity of IRRG is $O(n \log(n))$.

One of the cases with a high number of IRRG iterations is as follows. The considered problem consists of $n/2$ subproblems of size 2, one interaction is found during the first iteration, and at every $\epsilon_{\min}$ iteration another single interaction is found. Then, the number of IRRG iterations is $1 + \epsilon_{\min} \cdot n/2$. Since the time complexity of finding a new single interaction is $O(n)$, the worst time complexity of IRRG is $O(n^2)$.

V. Experiments and Results Analysis

The objective of the experiments was twofold. First—to check if the proposed problem decomposition strategy is more robust than the DG-based strategies. The second objective was to verify if IRRG can be successfully embedded into CC frameworks. To this end, we integrate IRRG with CBCC [8], [10] and CCFR2 [9], which were shown effective in solving LSGO problems. We compare these methods denoted as CBCC-IRRG and CCFR2-IRRG with their versions employing RDG3 denoted as CBCC-RDG3 [8], [10], [16] and CCFR2-RDG3 [9], [16], respectively. The competing methods were supplemented by SHADE-ILS [6]. Note that CBCC-RDG3 and SHADE-ILS are the state-of-the-art methods when LSGO is considered. To the best of our knowledge, CCFR2-RDG3 was not proposed yet. Nevertheless, considering it among the competing methods seems justified and desirable. IRRG- and RDG3-embedded CCs significantly outperformed CBCC-FVIL [8], [22] and CCFR2-FVIL [9], [22]. Therefore, FVIL-based results are reported in the supplementary material.

This section is divided into four sections. First, we present the setup of all considered optimization methods. Then, we describe the chosen test problems. In Section V-C, we discuss the decomposition accuracy of IRRG and RDG3 as well as their decomposition costs. Finally, the optimization results are reported in the last section.

A. Optimization Methods’ Setup

CC frameworks require component optimizers. For this purpose, we employ CMA-ES [41], which was shown to be the best option for both considered CC frameworks [9], [14]. We use CMA-ES implementation, which was shown to be the most efficient in [42]. The original implementation of CBCC-RDG3 was supported in MATLAB. It was rewritten to Python to integrate it with the source code of CMA-ES. The implementation of CCFR2 was made based on its pseudocodes presented in [9]. For RDG3, we use our original MATLAB source code. The decomposition that resulted from using this source code was used as an input to the Python implementations of CBCC and CCFR2. Similarly, IRRG was written in C++, but its output was passed to Python. The pack with the source code, the detailed results of all runs, and the results of all performed statistical tests can be downloaded from the public repository.

For SHADE-ILS, we took the source code provided by its Authors.

The parameters of the considered decomposition strategies, CC frameworks, and SHADE-ILS are adopted from articles in which they were proposed or recently applied. For RDG3, we use $\epsilon_s = 100$ and $\epsilon_a = 50$ [16]. CBCC is executing CMA-ES for 1000 FFEs and the smoothing factor $w$ is 0.5 [10], [16]. For CCFR2, we use $w = 0.1$ and CMA-ES (executed for 100 iterations) as a component optimizer [9]. SHADE-ILS uses the population size of 100 for its underlying differential evolution, MTS-LSI [35] with the initial step set to 20, the minimum improvement ratio is 5% and the allowed number of iterations without the improvement is 3 [6].

The configuration of the proposed IRRG is as follows. To obtain $x_{\infty}$, we use two optimizers. First, we use SHADE with the recommended settings [34] (population size: 100; arc ratio: 2; $p_{\text{best}}$ ratio: 0.1; memory size: $10^3$). Then, the solution proposed by SHADE is optimized by MTS-LS1 [35] (search range: 20% of the difference between the upper bound and the lower bound). The motivation behind using two optimizers is to consider SHADE as a global search optimizer and MTS-LS1 as a local search optimizer. They are executed for $5 \cdot 10^3$ and $1.5 \cdot 10^4$ FFEs, respectively. The other IRRG parameters are: $\epsilon_{\text{init}} = 15$, $n_s = 10$, and $\epsilon_s = 100$ (the value of $\epsilon_s$ is adopted from [16]). The FFE-based stop conditions and parameters $\epsilon_{\text{init}}$ and $n_s$ were tuned to find their minimal values that still lead to the perfect decomposition of considered nonoverlapping benchmarks.

B. Test Problems

As benchmarks we use functions from IEEE CEC’2013 special session on LSGO [23]. CEC’2013 was also considered by other up-to-date articles that considered LSGO [6], [9], [13], [14], [15], [16]. However, in CEC’2013 almost all separable subfunctions are additively separable, which favors DG-based strategies (see Section III-B). To this end, we propose two new sets based on CEC’2013, which eliminate this drawback.

Let us consider a function $f$ and its range $Y$. Let $g$ be a function that is strictly increasing over $Y$. Then, we can state that $\forall x_1, x_2 \in Y \Rightarrow f(x_1^*) < f(x_2^*) \Rightarrow h(x_1^*) < h(x_2^*)$, where $h = g \circ f$. 

\[2\] https://github.com/CMA-ES/pycma
\[3\] https://bitbucket.org/yuans/rdg3
\[4\] https://github.com/kommar/IRRG
\[5\] https://github.com/dmolina/shadeils
For each such function composition, any result of a comparison between solutions does not change. Therefore, \( f \) and \( h \) have the same intervals of monotonicity, optima, and variable interactions (see Section III-A). However, even if \( f \) is additively separable, \( h \) may be nonadditively separable, e.g., when \( g \) is a nonlinear function. Nevertheless, their ideal interaction matrices are the same.

The minimal value of all fifteen CEC’2013 functions is not lower than 0. Thus, as \( g \) we need functions that are strictly increasing over non-negative arguments. In this article, we use \( g_1(y) = y^2 \) and \( g_2(y) = \sqrt{y} \). They were chosen, because they increase in different ways. The test cases created using \( g_1 \) and \( g_2 \) are denoted as \( (f_i)^2 \) and \( \sqrt{f_i} \), respectively. For the sake of clarity, we define a set of the \( i \)th functions \( F_i \) as \( \{f_i, (f_i)^2, \sqrt{f_i}\} \) for \( i \in \{1, \ldots, 15\} \). To summarize, we consider the following five categories of functions [9], [23].

1. Fully separable functions \((F_1 – F_3)\).
2. Partially separable functions with separable variables \((F_4 – F_7)\).
3. Partially separable functions with no separable variables \((F_8 – F_{11})\).
4. Overlapping functions \((F_{12} – F_{14})\).
5. Fully nonseparable functions \((F_{15})\).

Considering the type of interaction between separable sub-functions, we split the first four categories into:

1. functions with additively separable subfunctions \((f_1, f_2, f_4 – f_{14})\); and
2. functions with nonadditively separable subfunctions \((f_3, (f_3)^2, f_5 – f_6, \sqrt{f_8} – \sqrt{f_9})\).

The computation budget was set to \( 3 \cdot 10^6 \) FFEs and each experiment was repeated 25 times [6], [9], [13], [14], [15], [16], [23].

As a real-world optimization problem with nonadditively separable subproblems, we analyze a multipath routing problem in computer and communication networks. In contrast to the traditional routing approach that transmits all traffic of a given demand along a single path, multipath routing splits the traffic among several paths. Multipath routing can improve network efficiency in terms of various measures of Quality of Service, including survivability, congestion, and throughput [43], [44], [45]. More details, including the results, can be found in the supplementary material.

C. Accuracy and Costs of Decomposition

IRRG is a decomposition strategy that never reports false linkage (in contrast to DG-based strategies) and should not suffer from missing linkage, which is a typical drawback of monotonicity checking. We employ three measures proposed in [46] to verify the accuracy of the decomposition. The first measure \( (\rho_1) \) returns the accuracy of finding interaction, and its value is maximum when there is no missing linkage. The second measure \( (\rho_2) \) refers to false linkage. The higher its value is, the lower is the number of nonexistent interactions detected. The value of the third measure \( (\rho_3) \) is maximal when there is neither false nor missing linkage

\[
\rho_1 = \frac{\sum_{i=1}^{n} \sum_{j=1,j \neq i}^{n} (\Theta \circ (\Theta^*))_{i,j}}{\sum_{i=1}^{n} \sum_{j=1,j \neq i}^{n} (\Theta^*)_{i,j}} \cdot 100\% \quad (11)
\]

\[
\rho_2 = \frac{\sum_{i=1}^{n} \sum_{j=1,j \neq i}^{n} ((1 - \Theta) \circ (1 - (\Theta^*)))_{i,j}}{\sum_{i=1}^{n} \sum_{j=1,j \neq i}^{n} (1 - (\Theta^*))_{i,j}} \cdot 100\% \quad (12)
\]

\[
\rho_3 = \frac{\sum_{i=1}^{n} \sum_{j=1,j \neq i}^{n} ((1 - \Theta - (\Theta^*))_{i,j}}{n \cdot (n - 1)/2} \cdot 100\% \quad (13)
\]

where \( n \) is the problem size, \( \Theta^* \) is the ideal interaction matrix, \( I \) denotes the all-ones matrix, \( o \) indicates the Hadamard product of two matrices, and \( (\Theta - \Theta^*) \) is a matrix which elements are defined as \( [(\Theta)_{i,j} - (\Theta^*)_{i,j}] \).

In Table I, we present the decomposition accuracy of IRRG and RDG3. IRRG is nondeterministic, thus, we report the summarized results of 50 runs. We summarize the runs of CBCC-IRRG and CCFR2-IRRG to show the quality of decomposition that was later on used during the optimization. For all nonoverlapping functions, IRRG is highly repeatable—the standard deviation is equal to 0% for these test cases. For functions with separable variables, \( \rho_2 \) and \( \rho_3 \) were computed just before joining them into groups of \( \epsilon_i \) size (otherwise, IRRG and RDG3 would report false linkage). For the overlapping functions the \( \rho_2 \) values are below 100% for IRRG, although it never reports false linkage. This is caused by the fact that \( \rho_1, \rho_2, \) and \( \rho_3 \) allow only for direct interactions, while IRRG and RDG3 consider also the indirect ones [12], [47].

For the separable and partially separable functions (additive or nonadditive), the values of all measures are optimal for all IRRG runs, i.e., IRRG found all interactions and never reported false linkage regardless of the type of interaction between subfunctions (Table I, \( F_1 – F_{11} \)). For the standard CEC’2013 functions, except for \( f_3 \) and \( f_6 \), RDG3 did not report false linkage. Such results were expected for \( f_3 \), because it is the only function from the standard CEC’2013 set that is nonadditively separable. Thus, \( \rho_2 = \rho_3 = 0\% \) for RDG3 reported for \( f_3 \) mean that all separable variables were grouped together. On the other hand, \( f_6 \) is additively separable, which makes \( \rho_1 = 71.93\% \) an unexpected result. The possible reason behind this result seems to be the inaccuracy of the automatic \( \epsilon \) estimation procedure, i.e., in some cases the value of \( \epsilon \) might be too low. Additionally, for \( f_3, f_8, \) and \( f_{10} \) the value of \( \rho_1 \) is below 100%, which means that RDG3 did not discover all interactions. The most convincing explanation seems to be too high value of \( \epsilon \). To omit the issue of missing linkage, IRRG executes RRG several times, which increases the risk of reporting false linkage (in the case when \( \epsilon \) is too low). Nevertheless, Table I shows that considering only those samples for which the outcome of \( f \) differs, in terms of the \( \epsilon \) value, is sufficient to report false linkage never or very rarely.

All modified \( f_1 – f_3 \) functions are nonadditively separable. Thus, we expect that for these functions \( \rho_2 \) is 0% for RDG3. The results for \((f_1)^2\) and \( \sqrt{f_1} \) have significantly higher values (although still much worse than the optimal 100%). These results are caused by cases when too high value of \( \epsilon \) allows RDG3 to omit discovering some of the nonexistent dependencies. Moreover, RDG3 tries to limit the size of a single group to \( \epsilon_i \), which may mitigate false linkage phenomenon. Note that the inappropriate value of \( \epsilon \) influences RDG3 differently in \((f_1)^2\) and \( \sqrt{f_1} \) cases. For the modified versions of partially separable functions, we also expect grouping all variables together by RDG3, i.e., \( \rho_1 = 100\% \) and \( \rho_2 = 0\% \). However, similarly
to \((f_1)^2-(f_3)^2\) and \(\sqrt{f_1}-\sqrt{f_3}\), too high value of \(\epsilon\) may lead to mitigating the false linkage phenomenon.

A perfect, unique decomposition does not exist for overlapping problems \((F_{12}-F_{14})\), and it may be beneficial to use various decompositions during the optimization process [17]. In all overlapping test cases, every variable is dependent (directly or indirectly) on all the others. For these problems, the value of \(\rho_1\) is maximum when only one group is created, while \(\rho_2\) is maximum if all created groups consist of only directly dependent variables. Finally, \(\rho_3\) is a tradeoff between \(\rho_1\) and \(\rho_2\). Thus, its value cannot reach 100%. Therefore, each IRRG execution may lead to various groupings and the standard deviation values for \(F_{12}-F_{14}\) are higher than 0%.

IRRG can detect both types of separability (additive and nonadditive). Thus, we expect it to return similar results for each overlapping CEC’2013 function and its modifications. The results reported in Table I confirm this expectation—for most of the functions the values of all three measures are similar for the standard, squared, and square rooted function versions. This observation was verified by the unpaired Wilcoxon test (significance level of 5%) with the Holm-Bonferroni correction for measures \(\rho_1-\rho_3\) and pairs \((f_i, (f_i)^2), (f_i, \sqrt{f_i}), (\sqrt{f_i}, (f_i)^2)\), where \(i \in \{12, 13, 14\}\). Such results of statistical tests may be surprising for \(f_{14}\) for which the differences between the median values of \(\rho_1\) seem high. Nevertheless, it is justified by the high value of the standard deviation.

### Table I

| Func. | Med | Avg | Std | Med | Avg | Std | Med | Avg | Std |
|-------|-----|-----|-----|-----|-----|-----|-----|-----|-----|
| \(f_1\) | N/A | N/A | N/A | 100% | 100% | 0% | 100% | 100% | 0% |
| \(\sqrt{f_1}\) | N/A | N/A | N/A | 100% | 100% | 0% | 100% | 100% | 0% |
| \(f_2\) | N/A | N/A | N/A | 100% | 100% | 0% | 100% | 100% | 0% |
| \(\sqrt{f_2}\) | N/A | N/A | N/A | 100% | 100% | 0% | 100% | 100% | 0% |
| \(f_3\) | N/A | N/A | N/A | 100% | 100% | 0% | 100% | 100% | 0% |
| \(\sqrt{f_3}\) | N/A | N/A | N/A | 100% | 100% | 0% | 100% | 100% | 0% |
| \(f_4\) | 100% | 100% | 0% | 100% | 100% | 0% | 100% | 100% | 0% |
| \(\sqrt{f_4}\) | 100% | 100% | 0% | 100% | 100% | 0% | 100% | 100% | 0% |
| \(f_5\) | 100% | 100% | 0% | 100% | 100% | 0% | 100% | 100% | 0% |
| \(\sqrt{f_5}\) | 100% | 100% | 0% | 100% | 100% | 0% | 100% | 100% | 0% |
| \(f_6\) | 100% | 100% | 0% | 100% | 100% | 0% | 100% | 100% | 0% |
| \(\sqrt{f_6}\) | 100% | 100% | 0% | 100% | 100% | 0% | 100% | 100% | 0% |
| \(f_7\) | 100% | 100% | 0% | 100% | 100% | 0% | 100% | 100% | 0% |
| \(\sqrt{f_7}\) | 100% | 100% | 0% | 100% | 100% | 0% | 100% | 100% | 0% |
| \(f_8\) | 100% | 100% | 0% | 100% | 100% | 0% | 100% | 100% | 0% |
| \(\sqrt{f_8}\) | 100% | 100% | 0% | 100% | 100% | 0% | 100% | 100% | 0% |
| \(f_9\) | 100% | 100% | 0% | 100% | 100% | 0% | 100% | 100% | 0% |
| \(\sqrt{f_9}\) | 100% | 100% | 0% | 100% | 100% | 0% | 100% | 100% | 0% |
| \(f_{10}\) | 100% | 100% | 0% | 100% | 100% | 0% | 100% | 100% | 0% |
| \(\sqrt{f_{10}}\) | 100% | 100% | 0% | 100% | 100% | 0% | 100% | 100% | 0% |
| \(f_{11}\) | 100% | 100% | 0% | 100% | 100% | 0% | 100% | 100% | 0% |
| \(\sqrt{f_{11}}\) | 100% | 100% | 0% | 100% | 100% | 0% | 100% | 100% | 0% |
| \(f_{12}\) | 99.20% | 99.07% | 0.32% | 84.86% | 83.94% | 4.80% | 84.89% | 83.97% | 4.79% |
| \(\sqrt{f_{12}}\) | 99.20% | 99.14% | 0.29% | 84.56% | 81.80% | 9.14% | 84.59% | 81.83% | 9.12% |
| \(f_{13}\) | 99.93% | 99.84% | 0.22% | 84.23% | 23.88% | 8.44% | 27.79% | 22.34% | 100% |
| \(\sqrt{f_{13}}\) | 99.93% | 99.84% | 0.18% | 84.23% | 23.88% | 8.44% | 27.79% | 22.34% | 100% |
| \(f_{14}\) | 99.70% | 99.70% | 0.24% | 37.54% | 33.07% | 28.28% | 42.66% | 38.55% | 25.93% |
| \(\sqrt{f_{14}}\) | 99.70% | 99.70% | 0.22% | 37.54% | 33.07% | 28.28% | 42.66% | 38.55% | 25.93% |
| \(f_{15}\) | 99.89% | 99.78% | 0.24% | 13.35% | 22.66% | 45.41% | 20.47% | 29.01% | 33.30% |
| \(\sqrt{f_{15}}\) | 99.89% | 99.78% | 0.22% | 13.35% | 22.66% | 45.41% | 20.47% | 29.01% | 33.30% |

Authorized licensed use limited to the terms of the applicable license agreement with IEEE. Restrictions apply.
RDG3 and IRRG costs are statistically significant. Bonferroni correction confirms that the differences between the results of the Student’s t-test with the Holm–Bonferroni correction are not only observed, but significant for RDG3. The results of the Student’s t-test with the Holm–Bonferroni correction are confirmed by the unpaired Wilcoxon test with the Holm–Bonferroni correction of the considered optimization methods are confirmed by the unpaired Wilcoxon test with the Holm–Bonferroni correction at the significance level of 5%.

Finally, both decomposition strategies handle the fully nonseparable function \( f_{15} \). In this case, the proposed modifications do not affect the quality of the decomposition.

The results presented above show that IRRG outperforms RDG3 in decomposing the fully and partially separable functions. For fully nonseparable functions, both strategies perform equally well, whereas for the overlapping functions the situation is more complex. However, we can state that, in general, IRRG proposes the higher-quality decomposition than RDG3. However, this advantage has its costs—as shown in Table II for the standard CEC’2013 functions, IRRG consumes from 2 to 12 times more FFEs on average. For the modified versions of F12 and F13, the values of \( \rho_2 \) are equal to 0%. For \( (f_{12})^2 \) and \( \sqrt{f_{14}} \), the results differ most probably due to the influence of the inappropriate value of \( \epsilon \) for some interaction checks and the impact of \( \epsilon_n \).

For RDG3, the observations for \( F_{12} \)–\( F_{14} \) are opposite to the observations for IRRG. For \( f_{12} \)–\( f_{14} \), RDG3 is able to find a balance between missing and false linkage, i.e., the values of all measures are above 90%. However, for the modified versions of \( F_{12} \) and \( F_{13} \), the values of \( \rho_2 \) are equal to 0%. For \( (f_{12})^2 \) and \( \sqrt{f_{14}} \), the results differ most probably due to the influence of the inappropriate value of \( \epsilon \) for some interaction checks and the impact of \( \epsilon_n \).

Table III presents the comparison of the effectiveness of CBCC and CCFR2 after embedding IRRG and RDG3. Additionally, we also consider SHADE-ILS, which is dedicated to solving LSGO problems, but does not use the problem decomposition. The observed differences in the results quality of the considered optimization methods are confirmed by the unpaired Wilcoxon test with the Holm–Bonferroni correction method at the significance level of 5%.

The maximum number of IRRG iterations was reached for \( \sqrt{f_{12}} \) and was equal to 169 whereas only one iteration was needed for \( F_1 \)–\( F_3 \). On average IRRG executed RRG 22.74 ± 29.47 times. After excluding the fully separable functions and \( F_{12} \), which are special cases for IRRG, the average number of iterations is 19.22 ± 4.44. These average values are much smaller than considered problem sizes. Thus, the typical time complexity of IRRG may be found as \( O(n \log(n)) \).

### D. Optimization Results

Decomposition of a problem is only a part of the whole optimization process. Additionally, even the high-quality decomposition does not guarantee the successful optimization of a given problem. Optimization methods that can use such decomposition wisely are also needed [17]. For instance, CBCC and CCFR2 offer a speed-up of a CC framework by greedily choosing the most promising subproblem to optimize [8], [9], [10]. Another example is joining separable variables into groups and optimizing them by CMA-ES [16], which can effectively solve fully separable problems up to 100 variables [48]. Thus, in this section, we compare the effectiveness of CBCC and CCFR2 after embedding IRRG and RDG3.

| Func | RDG3 Avg | IRRG (Avg) / RDG3 Avg | Avg | Std | Min | Max | Med | Avg | Std |
|------|-----------|------------------------|-----|-----|-----|-----|-----|-----|-----|
| \( f_{12} \)–\( f_{14} \) | \( 4.0 \pm 0.5 \) | \( 1.7 \pm 0.3 \) | 6.7 | 13.3 | 13.3 | 11.1 | 3.8 |
| \( f_{12} \)–\( f_{14} \) | \( 1.0 \pm 0.4 \) | \( 1.1 \pm 0.3 \) | 4.8 | 8.8 | 7.8 | 7.3 | 1.8 |
| \( f_{12} \)–\( f_{14} \) | \( 1.9 \pm 0.4 \) | \( 2.5 \pm 0.2 \) | 3.6 | 5.4 | 4.2 | 4.3 | 0.8 |
| \( f_{13} \)–\( f_{14} \) | \( 2.7 \pm 0.4 \) | \( 1.9 \pm 0.4 \) | 4.2 | 14.6 | 4.4 | 7.7 | 5.9 |
| \( f_{15} \) | \( 6.0 \pm 0.3 \) | N/A | 4.7 | 4.7 | 4.7 | 4.7 | N/A |
| \( (f_{12})^2 \)–\( (f_{14})^2 \) | \( 6.3 \pm 0.3 \) | \( 5.3 \pm 0.2 \) | 5.8 | 6.7 | 6.7 | 6.4 | 0.5 |
| \( (f_{12})^2 \)–\( (f_{14})^2 \) | \( 8.1 \pm 0.3 \) | \( 7.6 \pm 0.3 \) | 5.6 | 12.7 | 10.0 | 9.6 | 3.1 |
| \( (f_{12})^2 \)–\( (f_{14})^2 \) | \( 1.4 \pm 0.4 \) | \( 5.4 \pm 0.5 \) | 3.8 | 9.3 | 6.8 | 6.7 | 2.6 |
| \( (f_{13})^2 \)–\( (f_{14})^2 \) | \( 6.3 \pm 0.3 \) | \( 1.5 \pm 0.3 \) | 8.3 | 124.2 | 12.7 | 48.4 | 65.7 |
| \( f_{15} \) | \( 6.0 \pm 0.3 \) | N/A | 4.8 | 4.8 | 4.8 | 4.8 | N/A |
| \( \sqrt{f_{12}} \)–\( \sqrt{f_{14}} \) | \( 4.6 \pm 0.5 \) | \( 7.9 \pm 0.3 \) | 5.4 | 6.7 | 6.7 | 6.3 | 0.7 |
| \( \sqrt{f_{12}} \)–\( \sqrt{f_{14}} \) | \( 1.0 \pm 0.4 \) | \( 3.1 \pm 0.3 \) | 5.4 | 12.8 | 7.2 | 8.1 | 3.5 |
| \( \sqrt{f_{12}} \)–\( \sqrt{f_{14}} \) | \( 1.2 \pm 0.4 \) | \( 5.0 \pm 0.6 \) | 3.8 | 13.5 | 8.2 | 8.4 | 4.1 |
| \( \sqrt{f_{13}} \)–\( \sqrt{f_{14}} \) | \( 3.6 \pm 0.3 \) | \( 1.1 \pm 0.3 \) | 8.9 | 123.0 | 12.8 | 48.2 | 64.8 |
| \( \sqrt{f_{15}} \) | \( 6.0 \pm 0.3 \) | N/A | 4.9 | 4.9 | 4.9 | 4.9 | N/A |

The maximum number of IRRG iterations was reached for \( \sqrt{f_{12}} \) and was equal to 169 whereas only one iteration was needed for \( F_1 \)–\( F_3 \). On average IRRG executed RRG 22.74 ± 29.47 times. After excluding the fully separable functions and \( F_{12} \), which are special cases for IRRG, the average number of iterations is 19.22 ± 4.44. These average values are much smaller than considered problem sizes. Thus, the typical time complexity of IRRG may be found as \( O(n \log(n)) \).
In Table IV, we compare the effectiveness of CCFR2-IRRG to CBCC-IRRG and SHADE-ILS. The results show that the effectiveness of CCFR2-IRRG and CBCC-IRRG is highly similar. CCFR2-IRRG was chosen for the comparison to SHADE-ILS, because it performs slightly better. SHADE-ILS is significantly different from CC-based optimizers, and it does not rely on problem decomposition. As expected, for partially separable problems from the standard and modified CEC’2013 sets, CCFR2-IRRG outperforms SHADE-ILS significantly. However, the situation is the opposite for fully separable, overlapping, and nonseparable functions. Such results are a consequence of the construction of CBCC and CCFR2. Additionally, SHADE-ILS employs MTS-L1, a local search optimizer dedicated to fully separable problems [6], [35]. MTS-L1 along with the restart mechanism seems to be the reason why SHADE-ILS has outperformed all CC-based optimizers for $F_1 \rightarrow F_3$. Note that CMA-ES (employed by all of the considered CC-based optimizers) is dedicated to optimizing problems with at most 100 variables [48], while all of the considered fully nonseparable functions are high dimensional. Finally, all functions in the CEC’2013 set are homogeneous, i.e., all their subfunctions derive from the same base function (e.g., Rastrigin’s function) [23]. Such a feature may favor SHADE-ILS that adjusts the parameter values of its underlying SHADE and the choice of a local search method (MTS-LS1 or L-BFGS-B) to fitness landscape characteristics. If considered problems are not homogeneous, then such problems should be more suitable for CC-based optimizers because they may adjust their behavior to the subset of interacting variables not to the whole variable set.

Usually, when IRRG-embedded CC frameworks outperform SHADE-ILS, the order of magnitude of optimization result differences is higher when compared to the opposite situations. In supplementary material, we support an extended discussion on this issue and Fig. S-1 to picture it.

In Table V, we present the results for the standard CEC’2013 functions. These detailed results reveal some phenomena discussed below. According to Table I, only IRRG decomposes $f_5$ precisely. Therefore, CCF2-IRRG produced about $10^8$ and $10^{16}$ times better results than RDG3-embedded CCs and SHADE-ILS, respectively. However, CBB-IRRGG proposed results of quality that was similar to RDG3-embedded CCs. The reason seems to be as follows. CBCC and CCFR2 focus on optimizing those components that have the highest impact on the global fitness. However, in CCFR2, the component’s initial relatively poor contribution becomes insignificant faster. Thus, CCFR2 may spend more time improving those initially less significant components. This example shows that even having the perfect decomposition of a partially separable problem may not be enough to reach the high-quality results. The differences between CBCC and CCFR2 are also visible for $f_{11}$, for which both CCFR2 versions outperform the CBCC versions. For overlapping problems, the perfect decomposition may not exist [13], [17]. Therefore, the structure of the proposed groups influenced the quality of the final results the most significantly. $f_{13}$ and $f_{14}$ are defined as overlapping problems with conforming and conflicting subproblems, respectively, [49]. When subproblems are conforming, shared variables (variables

| $f_1$ | $f_2$ | $f_3$ | $f_4$ | $f_5$ |
|-------|-------|-------|-------|-------|
| Avg   | Avg   | Avg   | Avg   | Avg   |
| 9.16E-19 | 7.41E-19 | 8.57E-19 | 8.85E-19 | 7.64E-19 |
| Std   | Std   | Std   | Std   | Std   |
| 3.35E-19 | 1.90E-19 | 2.77E-19 | 3.07E-19 | 5.10E-28 |
| Med   | Med   | Med   | Med   | Med   |
| 2.93E-03 | 2.33E-03 | 2.33E-03 | 2.33E-03 | 1.03E-03 |

| $f_6$ | $f_7$ | $f_8$ | $f_9$ | $f_{10}$ |
|-------|-------|-------|-------|---------|
| Avg   | Avg   | Avg   | Avg   | Avg     |
| 2.02E-01 | 1.98E-01 | 2.16E-01 | 1.97E-01 | 1.97E-01 |
| Std   | Std   | Std   | Std   | Std     |
| 0.00E+00 | 0.00E+00 | 0.01E+00 | 0.01E+00 | 0.01E+00 |

| $f_{11}$ | $f_{12}$ | $f_{13}$ | $f_{14}$ |
|--------|--------|--------|--------|
| Avg    | Avg    | Avg    | Avg    |
| 9.16E-19 | 7.41E-19 | 8.57E-19 | 8.85E-19 |
| Std    | Std    | Std    | Std    |
| 3.35E-19 | 1.90E-19 | 2.77E-19 | 3.07E-19 |
| Med    | Med    | Med    | Med    |
| 2.93E-03 | 2.33E-03 | 2.33E-03 | 2.33E-03 |

| $f_{15}$ | $f_{16}$ |
|---------|---------|
| Avg     | Avg     |
| 2.12E-06 | 2.12E-06 |
| Std     | Std     |
| 2.35E-05 | 1.91E-05 |
that belong to more than one subproblem) may be optimized for each subproblem separately, i.e., the optimal value of a shared variable is the same for all subproblems the shared variable belongs to. If subproblems are conflicting, optimizing one subproblem collides with the optimization of other subproblems that share the same variables. Note that the considered CC-based optimizers process disjoint components and each shared variable must belong to only one component. Therefore, if subproblems are concurring, it could be better to use smaller components like optimizing each subproblem separately. If subproblems contradict, then it may be better to use larger components to optimize many conflicting subproblems at the same time. In general, RDG3 tends to create more groups containing fewer variables than IRRG due to the $\epsilon$ threshold. Therefore, RDG3 seems suitable for $f_{13}$ (RDG3-embedded CCs report approximately 100 times better results than IRGG-embedded ones). The situation is opposite for $f_{14}$. Finally, for $f_{12}$, which contains neither conforming nor contradicting subproblems, both variable grouping ways seem equally useful. Thus, the differences in the optimization results for $f_{13}$ and $f_{14}$ arise from the differences of the underlying problem structure and the employed grouping manner.

In Tables S-I and S-II (in the supplementary material, Section S-II), we report the results for the modified CEC’2013 sets. As expected, for these functions, RDG3-embedded CCs are significantly less effective when compared to IRGG-embedded ones. The results remain similar for $f_{15}^2$ and $\sqrt{f_{15}}$, which is expected because these functions are fully nonseparable. RDG3-embedded CCs are no longer dominating for $f_{13}^2$ and $\sqrt{f_{13}}$, because RDG3 detects more (nonexisting) dependencies. Therefore, it proposes larger groups, which may be advantageous for overlapping problems with conforming subproblems. However, despite increasing the size of the groups, RDG3-embedded CCs do not improve their effectiveness for $f_{14}^2$ and $\sqrt{f_{14}}$. Indeed, RDG3 proposes larger groups for these functions, but since they contain false linkage, these groups mix variables from different subfunctions. Such low-quality groups do not improve the quality of the search.

Functions from $F_5$, i.e., $f_5$, $(f_5)^2$, and $\sqrt{f_5}$, contain separable variables. However, differently to other functions of this type, the quality of the results reported for $F_5$ is similar for all IRGG- and RDG3-embedded CCs. Based on the reported results, it seems that for $F_5$, the influence of the fitness of separable variables is significantly higher than for the other partially separable functions with separable subproblems ($F_4$, $F_6$, $F_7$). Therefore, the quality of the decomposition does not play an important role in the optimization of $f_5$, $(f_5)^2$, and $\sqrt{f_5}$. Note that from all the considered optimization methods, SHADE-ILS is the most effective in optimizing fully separable problems. Hence, SHADE-ILS outperformed the others for $F_5$.

VI. CONCLUSION

In this article, we propose a new decomposition strategy, namely IRGG, which derives from monotonicity checking, never reports false linkage, and significantly mitigates the risk of missing linkage. Although IRGG is more expensive than RDG3, its typical time complexity is also $O(n \log(n))$. Additionally, the FFE cost of IRGG is low when compared to the overall optimization cost. IRGG- and RDG3-embedded CCs report similar quality results for the standard CEC’2013 functions, which are largely suitable for RDG3. However, for both proposed modified CEC’2013 sets with functions with nonadditively separable subfunctions, the effectiveness of IRGG-embedded CCs remains the same, while for RDG3-embedded CCs deteriorates significantly. Moreover, IRGG-embedded CCs outperform RDG3-embedded CCs for the considered real-world optimization problem with nonadditively separable subproblems.

The comparison of IRGG-embedded CCs and SHADE-ILS shows that CCs may choose different component optimizers depending on the component size and the existence of separable variables. Another promising future work direction is adjusting the $\epsilon$ value during the IRGG run and further improvements in the procedure of automatic $\epsilon$ estimation. Finally, proposing new benchmark sets with nonhomogeneous functions may be helpful for future research.

ACKNOWLEDGMENT

The authors wish to express their gratitude to Roza Goscien for valuable discussions on real-world optimization problems.

REFERENCES

[1] 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. 1, pp. 729–745, 2020.

[2] Y.-H. Jia, Y. Mei, and M. Zhang, “A Memetic level-based learning swarm Optimizer for large-scale water distribution network optimization,” in Proc. Genet. Evol. Comput. Conf., 2020, pp. 1107–1115.

[3] A. Yaman, D. C. Mocanu, G. Iacca, G. Fletcher, and M. Pechenizkiy, “Limited evaluation cooperative co-evolutionary differential evolution for large-scale Neuroevolution,” in Proc. Genet. Evol. Comput. Conf., 2018, pp. 569–576.

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

[5] W. Dong, T. Chen, P. Tiño, and X. Yao, “Scaling up estimation of distribution algorithms for continuous optimization,” IEEE Trans. Evol. Comput., vol. 17, no. 6, pp. 797–822, Dec. 2013.

[6] D. Molina, A. LaTorre, and F. Herrera, “SHADE with iterative local search for large-scale global optimization,” in Proc. IEEE Congr. Evol. Comput. (CEC), 2018, pp. 1–8.

[7] M. A. Potter and K. A. De Jong, “A cooperative coevolutionary approach to function optimization,” in Parallel Problem Solving From Nature, Heidelberg, Germany: Springer, 1994, pp. 249–257.

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

[9] M. Yang, A. Zhou, C. Li, J. Guan, and X. Yan, “CCFR2: A more efficient cooperative co-evolutionary framework for large-scale global optimization,” Inf. Sci., vol. 512, pp. 64–79, Feb. 2020.

[10] Y. Sun, M. Kirley, and X. Li, “ Cooperative co-evolution with online Optimizer selection for large-scale optimization,” in Proc. Genet. Evol. Comput. Conf., 2018, pp. 1079–1086.

[11] 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.

6See the supplementary material.
[12] Y. Sun, M. Kirley, and S. K. Halgamuge, “Extended differential grouping for large scale global optimization with direct and indirect variable interactions,” in *Proc. Genet. Evol. Comput. Conf.*, 2015, pp. 313–320.

[13] 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.

[14] 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.

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

[16] 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. (CEC)*, 2019, pp. 326–333.

[17] M. W. Przewozniczek and M. M. Komarnicki, “Empirical linkage learning,” *IEEE Trans Evol. Comput.*, vol. 24, no. 6, pp. 1097–1111, Dec. 2020.

[18] M. W. Przewozniczek, B. Frej, and M. M. Komarnicki, “On measuring and improving the quality of linkage learning in modern evolutionary algorithms applied to solve partially additively separable problems,” in *Proc. Genet. Evol. Comput. Conf.*, 2020, pp. 742–750.

[19] M. W. Przewozniczek and N. Weicker, “On the improvement of coevolutionary optimizers by learning variable interdependencies,” in *Proc. IEEE Congr. Evol. Comput.*, vol. 3, 1999, pp. 1627–1632.

[20] W. Chen, T. Weise, Z. Yang, and K. Tang, “Large-scale global optimization using cooperative coevolution with variable interaction learning,” in *Parallel Problem Solving from Nature*. Heidelberg, Germany: Springer, 2010, pp. 1057–1066.

[21] 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. School Comput. Sci. Inf. Technol., RMIT Univ., Melbourne VIC, Australia, Rep., 2013.

[22] M. N. Omidvar, X. Li, and K. Tang, “Designing benchmark problems for large-scale continuous optimization,” *Inf. Sci.*, vol. 316, pp. 419–436, Sep. 2015.

[23] F. van den Bergh and A. P. Engelbrecht, “A cooperative approach to particle swarm optimization,” *IEEE Trans. Evol. Comput.*, vol. 8, no. 3, pp. 225–239, Jun. 2004.

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

[25] M. N. Omidvar, X. Li, and X. Yao, “Cooperative co-evolution with delta grouping for large scale non-separable function optimization,” in *Proc. IEEE Congr. Evol. Comput. (CEC)*, 2010, pp. 1–8.

[26] S. Mahdavi, M. E. Shir, and S. Ramananuyan, “Cooperative co-evolution with a new decomposition method for large-scale optimization,” in *Proc. IEEE Congr. Evol. Comput. (CEC)*, 2014, pp. 1285–1292.

[27] Y. Wang, H. Liu, F. Wei, T. Zong, and X. Li, “Cooperative Coevolution with formula-based variable grouping for large-scale global optimization,” *Evol. Comput.*, vol. 26, no. 4, pp. 569–596, 2018.

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

[29] IEEE Standard for Floating-Point Arithmetic, IEEE Standard 754-2008, 2008.

[30] A. LaTorre, S. Muelas, and J.-M. Peña, “A comprehensive comparison of large scale global optimizers,” *Inf. Sci.*, vol. 316, pp. 517–549, Sep. 2015.

[31] A. Hadi, A. Wagdy, and K. Jambi, “LSHADE-SPA memetic framework for solving large-scale optimization problems,” *Complex Intell. Syst.*, vol. 5, pp. 25–40, Dec. 2018.

[32] R. Tanabe and A. Fukunaga, “Success-history based parameter adaptation for differential evolution,” in *Proc. IEEE Congr. Evol. Comput. (CEC)*, 2013, pp. 71–78.

[33] L.-Y. Tseng and C. Chen, “Multiple trajectory search for large scale global optimization,” in *Proc. IEEE World Congr. Comput. Intell.*, 2008, pp. 3052–3059.

[34] J. L. Morozes and J. Nocedal, “Remark on ‘algorithm 778: L-BFGS-B: Fortran subroutines for large-scale bound constrained optimization’,” *ACM Trans. Math. Softw.*, vol. 38, no. 1, p. 7, 2011.

[35] O. Mersmann, M. Preuss, and H. Trautmann, “Benchmarking evolutionary algorithms: Towards exploratory landscape analysis,” in *Parallel Problem Solving from Nature*. Heidelberg, Germany: Springer, 2010, pp. 73–82.

[36] A. Engelbrecht, P. Bosman, and K. M. Malan, “The influence of fitness landscape characteristics on particle swarm optimisers,” *Nat. Comput.*, vol. 21, pp. 335–345, 2021.

[37] P. Kerschke and H. Trautmann, “Automated algorithm selection on continuous black-box problems by combining exploratory landscape analysis and machine learning,” *Evol. Comput.*, vol. 27, no. 1, pp. 99–127, 2019.

[38] S. Chen, A. Bouluf-Röhler, J. Montgomery, and T. Hendllass, “An analysis on the effect of selection on exploration in particle swarm optimization and differential evolution,” in *Proc. IEEE Congr. Evol. Comput. (CEC)*, 2019, pp. 3037–3044.

[39] N. Hansen and A. Ostermeier, “Completely derandomized self-adaptation in evolution strategies,” *Evol. Comput.*, vol. 9, no. 2, pp. 159–195, 2001.

[40] R. Biedrzycki, “On equivalence of algorithm’s implementations: The CMA-ES algorithm and its five implementations,” in *Proc. Genet. Evol. Comput. Conf.*, 2019, pp. 247–248.

[41] M. Piere and D. Medhi, *Routing, Flow, and Capacity Design in Communication and Computer Networks*. San Francisco, CA, USA: Morgan Kaufmann, 2004.

[42] I. Cidon, R. Rom, and Y. Shavitt, “Analysis of multi-path routing,” *IEEE/ACM Trans. Netw.*, vol. 7, no. 6, pp. 885–896, Dec. 1999.

[43] R. Banner and A. Orda, “Multipath routing algorithms for congestion minimization,” *IEEE/ACM Trans. Netw.*, vol. 15, no. 2, pp. 413–424, Apr. 2007.

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

[45] M. W. Przewozniczek, M. M. Komarnicki, and B. Frej, “Direct linkage discovery with empirical linkage learning,” in *Proc. Genet. Evol. Comput. Conf.*, 2021, pp. 690–617.

[46] R. Ros and N. Hansen, “A simple modification in CMA-ES achieving linear time and space complexity,” in *Parallel Problem Solving From Nature*. Heidelberg, Germany: Springer, 2008, pp. 296–305.

[47] 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.

Marcin Michał Komarnicki received the B.S. and M.S. degrees in computer science from the Wrocław University of Science and Technology, Wrocław, Poland, in 2013 and 2014, respectively, where he is currently pursuing the Ph.D. degree.

His current research interests include large-scale global optimization, linkage learning techniques, and multiobjective optimization.

Michal Witold Przewozniczek received the Ph.D. and D.Sc. degrees in computer science from the Wrocław University of Science and Technology, Wrocław, Poland, in 2012 and 2021, respectively. He is an Associate Professor with the Wrocław University of Science and Technology. He is also a co-owner of the company implementing recent scientific achievements in the industry field.

His research interests concentrate on evolutionary optimization development and its application in practice. This includes problem decomposition via linkage learning techniques, using problem structure decomposition in gray- and black-box optimization, multipopulation approaches, and dynamic management of subpopulations. More details at www.przewozniczek.eu.
Halina Kwasnicka received the Ph.D. and D.Sc. (Habilitation) degrees from the Wrocław University of Science and Technology (WUST), Wrocław, Poland, in 1980 and 2000, respectively. She is currently a Full Professor with the Department of Artificial Intelligence, WUST. In 2009, she was nominated by the President of Poland as a full professor (the highest scientific title in Poland). She gave several invited lectures. She coauthored over 200 scientific papers, five books (in Polish) and was co-editor of two books (Springer). Her research interest focuses on a broad understanding of Artificial Intelligence, from nature-inspired methods, data mining, and Explainable Artificial Intelligence.

Prof. Kwasnicka was awarded as TOP of the TOP Women in AI 2022 by Perspektywy Women in Tech and received the “Distinguished Mentor in AI” award.

Krzysztof Walkowiak (Senior Member, IEEE) received the Ph.D. and D.Sc. (Habilitation) degrees from the Wrocław University of Science and Technology (WroclawTech), Wrocław, Poland, in 2000 and 2008, respectively. He is currently a Full Professor with the Department of Systems and Computer Networks, WroclawTech. He has coauthored above 300 papers, 70 of them published in JCR journals. His research interest focuses on optimization of communication networks, machine learning, metaheuristics, and intent-based networks.

Prof. Walkowiak received the 2014 Fabio Neri Best Paper Award and Best Paper Awards of DRCN 2009 and RNDM 2015.