An Abstract Interface for Large-Scale Continuous Optimization Decomposition Methods

Rodolfo A. Lopes
Federal University of Ouro Preto
Ouro Preto, Brazil
rodolfoayala@ufop.edu.br

Rodrigo C. P. Silva
Federal University of Ouro Preto
Ouro Preto, Brazil
rodrigo.silva@ufop.edu.br

Alan R. R. de Freitas
Federal University of Ouro Preto
Ouro Preto, Brazil
alandefreitas@ufop.edu.br

ABSTRACT

Decomposition methods are valuable approaches to support the development of divide-and-conquer metaheuristics. When the problem structure is unknown, such as in black-box problems, this structure can be inferred through several decomposition mechanisms. In the context of continuous optimization, the most efficient metaheuristics to deal with a large number of decision variables involve decomposition methods. However, choosing a suitable decomposition method is not a trivial task since each strategy requires an appropriate set of parameters. In this context, this paper proposes a C++ library called Continuous Optimization Problem Decomposition (COPD) that provides the most recent decomposition methods, interfaces for new methods, and integration with solvers. Furthermore, the proposed library can aid related studies since the decomposition for continuous optimization problems can be easily applied with different methods. The experimental results demonstrate a high grouping accuracy for most methods on large-scale problems.

CCS CONCEPTS

• **Software and its engineering → Software libraries and repositories**; • **Mathematics of computing → Continuous functions**.

KEYWORDS

Decomposition Methods, Decomposition Library, Continuous Optimization

ACM Reference Format:
Rodolfo A. Lopes, Rodrigo C. P. Silva, and Alan R. R. de Freitas. 2021. An Abstract Interface for Large-Scale Continuous Optimization Decomposition Methods. In *2021 Genetic and Evolutionary Computation Conference Companion (GECCO '21 Companion), July 10–14, 2021, Lille, France.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3449726.3463188

1 INTRODUCTION

The performance of traditional optimization methods tends to deteriorate as the number of decision variables increases. The research field that studies methodologies to solve problems with many decision variables is known as Large-Scale Global Optimization (LSGO). There are two primary reasons for this performance deterioration [15]: (i) the dimensionality of decision vectors exponentially increases the search space, and (ii) the landscape complexity of the objective function also increases with the problem dimensionality.

Decomposition-based metaheuristics are recognized as successful approaches to solving LSGO problems [15]. For example, the winner of the 2019 *Competition on Large-Scale Global Optimization* [19] is a decomposition-based metaheuristic [26], and evidence is shown in [16] and [25] that decomposition-based methods tend to perform better than non-decomposition-based methods in large-scale problems.

In divide-and-conquer metaheuristics, decomposition methods investigate the decision variables interactions to find ways to split an optimization problem into a set of smaller independent subproblems. In addition, decomposition methods are helpful when the algebraic decomposition of the optimization problem is not known, e.g., black-box optimization problems. There exist several decomposition methods in the literature, and they aim at near-optimal optimization decomposition. Thus, the decomposition methods directly influence the performance of these metaheuristics [15].

Considering the importance of the decomposition strategies, we propose a library, called COPD - Continuous Optimization Problem Decomposition, to aid applications and metaheuristics based on decomposition approaches. The proposed library facilitates the implementation of optimization problems and the investigation of decision variable interactions. It is available in the project repository[1].

COPD provides the most recent decomposition algorithms, an interface to new decomposition methods, and integration with solvers. COPD also addresses important recommendations made to the optimization research community [2], for instance: (i) it provides standard benchmarking for the implemented methods; (ii) it makes the automated algorithm design easier once the selection of decomposition procedure can be automated; (iii) inspiration for theoretical studies brought by the empirical results; and (iv) a transparent and standard tool for the validation of new decomposition methods.

In the following sections, we introduce the main concepts necessary to successfully apply decomposition methods in continuous optimization problems. Section 2 provides a short definition of a continuous optimization problem and introduces general definitions for continuous optimization separability. We then present a brief overview of the main metaheuristics and their strategies to solve continuous optimization problems.

Section 3 introduces the COPD library and its main components. Section 4 describes details about the different decomposition methods available. Section 5 presents a comparison of these different

[1] https://github.com/RodolfoALopes/decomposition_library

GECCO '21 Companion, July 10–14, 2021, Lille, France

ACM ISBN 978-1-4503-8351-6/21/07…$15.00
https://doi.org/10.1145/3449726.3463188

decomposition methods on a well-known large-scale problem suite. Lastly, Section 6 summarizes the topics presented throughout this paper.

2 CONTINUOUS OPTIMIZATION

Optimization is a systematic search procedure for the best solution given an optimization problem. Without losing generality, the optimization process searches for candidate solutions that satisfy all the constraints and generate the minimum function value. A continuous optimization problem is formulated as in Equation 1.

$$\begin{aligned} \min \quad & f(\mathbf{x}) \\ s.t. \quad & g_i(\mathbf{x}) \leq 0, \ i = 1, ..., p \\ & h_j(\mathbf{x}) = 0, \ j = 1, ..., q \\ & l_k \leq x_k \leq u_k, k = 1, ..., D \\ & \mathbf{x} \in \mathbb{R}^D \end{aligned} \tag{1}$$

where $\mathbf{x}$ is a decision vector of D dimensions, the function $f(\cdot) : \mathbb{R}^D \rightarrow \mathbb{R}$ is the problem objective function, and the functions $g_i(\cdot)$ and $h_j(\cdot)$ define inequality and equality constraints, respectively. l_k is the lower bound and u_k is the upper bound of variable x_k.

2.1 Large-Scale Continuous Optimization Problems

A continuous optimization problem (Equation 1) with a high number of decision variables ($D \geq 1000$) might also be considered a Large-Scale Global Optimization (LSGO) problem [15, 18, 19]. Evolutionary Algorithms (EAs), such as the Covariance Matrix Adaptation - Evolution Strategy (CMA-ES) [10] and the Differential Evolution (DE) algorithm [23], are metaheuristics commonly used to tackle LSGO problems [15, 18]. However, their performance deteriorates with an increase in the number of decision variables [15, 18].

The major non-decomposition strategies to improve EAs are alternative initialization methods [6, 17]; adaptation/self-adaptation of operators and parameter control [1, 3, 22]; surrogate models [9, 14]; hybridization [8, 11]; and parallelization [4, 5]. However, these strategies can be implemented independently from decomposition strategies and are often not sufficient to deal with all the difficulties of LSGO problems, as they do not directly address the curse of dimensionality.

Decomposition approaches apply a divide-and-conquer procedure in order to improve the EAs performance [15, 18]. Solving lower-dimensional subproblems is faster than optimizing the complete model due to the curse of dimensionality.

2.2 Optimization Separability

Broadly speaking, an optimization problem is separable if we can divide it into two or more subproblems and solve these components independently. The following definition can describe this idea:

Definition 2.1. An optimization problem defined over the objective function, $f(\mathbf{x})$, is partially separable with m independent subcomponents if:

$$\arg\min_{\mathbf{x} \in \mathcal{F}} f(\mathbf{x}) = \left\langle \arg\min_{\mathbf{s}_1 \in \mathcal{F}_1} f(\mathbf{s}_1), \ \ldots, \ \arg\min_{\mathbf{s}_m \in \mathcal{F}_m} f(\mathbf{s}_m) \right\rangle \tag{2}$$

where m represents the number of independent subcomponents, $\mathbf{x} = \langle x_1, ..., x_D \rangle \in \mathbb{R}^D$ is a decision vector of D dimensions, $\mathbf{s}_1 \in \mathbb{R}^{D_1}, ..., \mathbf{s}_m \in \mathbb{R}^{D_m}$ are disjoint sub-vectors of $\mathbf{x}$, $2 \leq m \leq D$, and $D_1 + ... + D_m = D$. $\mathcal{F}_i \subset \mathbb{R}^{D_i}$ is the feasible set of the subproblem defined of over $\mathbf{s}_i$ such that $\mathcal{F}_1, ..., \mathcal{F}_m$ are disjoint sets and $\mathcal{F}_1 \cup ... \cup \mathcal{F}_m = \mathcal{F}$.

Notice that a special case of Definition 2.1 is when m is equal to D. In this case, the optimization problem is called fully separable, meaning that all decision variables can be searched independently.

Another useful definition is one of function additive separability, given below.

Definition 2.2. A $f(\mathbf{x})$ is partially additively separable function if:

$$f(\mathbf{x}) = \sum_{i=1}^{m} f\_sub_i(\mathbf{s}_i) \tag{3}$$

where $\mathbf{x} = \langle x_1, ..., x_D \rangle \in \mathbb{R}^D$ is the global decision variable vector of D dimensions, $\mathbf{s}_i$ are disjoint sub-vectors from $\mathbf{x}$, and m is the number of independent subcomponents.

When an objective function is additively separable, the unconstrained optimization problem defined over it is also separable according to Definition 2.1.

Definition 2.2 has been commonly used to find subproblems in several related works [12, 16, 20, 21, 24–27]. However, this definition does not represent all possible optimization separability types implied by Definition 2.1. Therefore, methods that are solely based on additive separability may not be able to find all subcomponents in a problem.

2.3 Decomposition Strategies

Decomposition strategies aim to provide procedures able to analyze and decompose an optimization problem into subcomponents. These decomposition strategies are essential to allow divider-and-conquer EAs approaches to work with black-box optimization problems. Cooperative Coevolutive approaches are one of the most popular of these divider-and-conquer EAs [26, 27, 30].

Decomposition strategies are only used when the problem structure is unknown and needs to be inferred. These strategies aim to achieve near-optional subcomponents in terms of optimization separability, as presented in Definition 2.1. Decomposition methods found in the literature only require representing a candidate solution and objective function to compute the solution quality. Thus, a possible decomposition of the optimization problem is provided at the end of processing.

Besides being an essential alternative to decompose black-box optimization problems, the decomposition methods can be fundamental to the development of gray-box operators since the structure of the problem is not available. For instance, crossover and local search gray-box operators have been proposed in the literature over the past years [7, 28, 29]. However, these operators are only applicable if the structure of the optimization problem is available; therefore, decomposition methods can provide a way to support it for black-box problems.

Different decomposition methods have been proposed since the 2010s, given the importance of these strategies. An important point concerning decomposition methods is choosing the most suitable

method is not a trivial task. This fact is due to the different characteristics and requirements of each method. Moreover, there is a lack of a unique repository that provides easy usage of the major decomposition methods.

Thus, this paper proposes a C++ library implementing decomposition algorithms to provide an essential part of the decomposition methods. Lastly, a repository with several decomposition methods can help develop new related studies since the decomposition of optimization problems can easily apply with different methods.

3 THE DECOMPOSITION LIBRARY

This section describes the design of the proposed decomposition library called Continuous Optimization Problem Decomposition (COPD). COPD provides several decomposition methods to analyze the decision variables interactions to split the original continuous optimization problem into smaller ones. The main goals of this library are to (i) support the decomposition analysis of the continuous optimization problems from different decomposition methodologies; (ii) allow straightforward extension of the decomposition methods; (iii) allow user-provided or third-party solvers to integrate the library; and (iv) allow the development of gray-box operators for black-box continuous optimization problems.

COPD has two major C++ classes: *optimization_problem* and *decomposition_algorithm* and two additional classes, *options* and *criteria*. The *options* class allows the specification of the parameters of the methods by the user. The *criteria* class keeps a set of possible stopping criteria which are updated by the *solver* and *decomposition_algorithm* classes and used by *solver* to halt the search.

The class diagram presented in Figure 1 describes these classes.

Figure 1 also depicts a standard interface for solvers, *solver*. It helps users implement their own solvers, but it is not part of our design goals to offer sophisticated algorithms for continuous optimization problems.

The *optimization_problem* abstract class is responsible for representing a continuous optimization problem. The *optimization_problem* abstract class has attributes to define the dimensionality, lower and upper boundaries, as well as, attributes to store information from the problem structure. The function of the problem class called *value* is used to calculate the objective function value given a candidate solution. Thus, for each continuous optimization problem that the user needs to represent, the user should inherit the *optimization_problem* class and implement the virtual function *value*.

The *optimization_problem* class also provides a function named *get_problem_structure*, which aims to get the problem structure identified by the decomposition algorithms. Thus, a data structure formed by sets of integers inside a vector is declared to store the problem subcomponents. The set of integers represents the indexes of the decision variables of the same subcomponent. Each set from the vector data structure represents a subcomponent in the problem.

The function *set_problem_structure* from the *optimization_problem* class is used to provide a way to assign the data structure that contains the problem structure. Every time this function is invoked, the protected attribute *problem_structure_is_known* is also assigned as true in addition to the problem structure is assigned.

Whenever a new optimization problem needs to be represented, a new class should be created, and it must inherit from the *optimization_problem* class. For each new decomposition algorithm implemented, a new class inheriting from the *decomposition_algorithm* class should be created. The pure virtual function called *decompose* must be implemented. In this scenario, the function *decompose* receives as parameters: (i) an object from the *optimization_problem* class representing an optimization problem to be decomposed; (ii) an object from the *options* class where has all parameters required by the decomposition method; and (iii) an object from the *criteria* class to account for the number of evaluation functions spent.

Lastly, this proposed library implemented eight different decomposition methods that the users can easily use, and details are available at the project repository[2].

4 DECOMPOSITION METHODS

The COPD library implements eight decomposition methods found in the literature. The set of methods is comprised of six well-established and well-studied methods [12, 16, 20, 21, 24, 25], each with at least 50 citations, and two updates [26, 27] of the method presented in [25].

Following the proposed project design shown in Figure 1, the decomposition methods inherit from the abstract class *decomposition_algorithm*.

4.1 Differential Grouping

Proposed by [20], the Differential Grouping (DG) method is one of the first and main decomposition strategies to examine and decompose continuous optimization problems automatically. Based on the partial additive separability definition, defined in Equation 3, DG uses the following theorem to identify an interaction between two decision variables:

THEOREM 4.1. *Let $f(\mathbf{x})$ be an additively separable function. $\forall\ a,\ b_1 \neq b_2,\ \sigma \in \mathbb{R},\ \sigma \neq 0$, if the following condition holds:*

$$\Delta_{\sigma,x_p}[f](\mathbf{x})|_{x_p=a,x_q=b_1} \neq \Delta_{\sigma,x_p}[f](\mathbf{x})|_{x_p=a,x_q=b_2} \tag{4}$$

implies,

$$\Delta_{\sigma,x_p}[f](\mathbf{x})|_{x_p=a,x_q=b_1} - \Delta_{\sigma,x_p}[f](\mathbf{x})|_{x_p=a,x_q=b_2} \neq 0 \tag{5}$$

then x_p and x_q interact with each other, i.e., they are non-separable, where

$$\Delta_{\sigma,x_p}[f](\mathbf{x}) = f(\ldots, x_p + \sigma, \ldots) - f(\ldots, x_p, \ldots) \tag{6}$$

means the difference of $f(\mathbf{x})$ with respect to x_p with the interval σ.

Broadly speaking, Theorem 4.1 means that there is an interaction between decision variables x_p and x_q if Equation 5 yields a value different from 0. For more details and the proof of Theorem 4.1, see [20].

The DG method analyzes pairs of decision variables, x_p, and x_q. It starts with the first decision variable, x_p, and computes its interaction with all other decision variables, x_q. If no decision variable interaction has been detected, it means that the decision variable x_p can be searched independently. Then, x_p is removed from the set of variables. Otherwise, if interactions are identified, all decision variables involved are grouped in the same set. Decision variables

[2] https://github.com/RodolfoALopes/decomposition_library

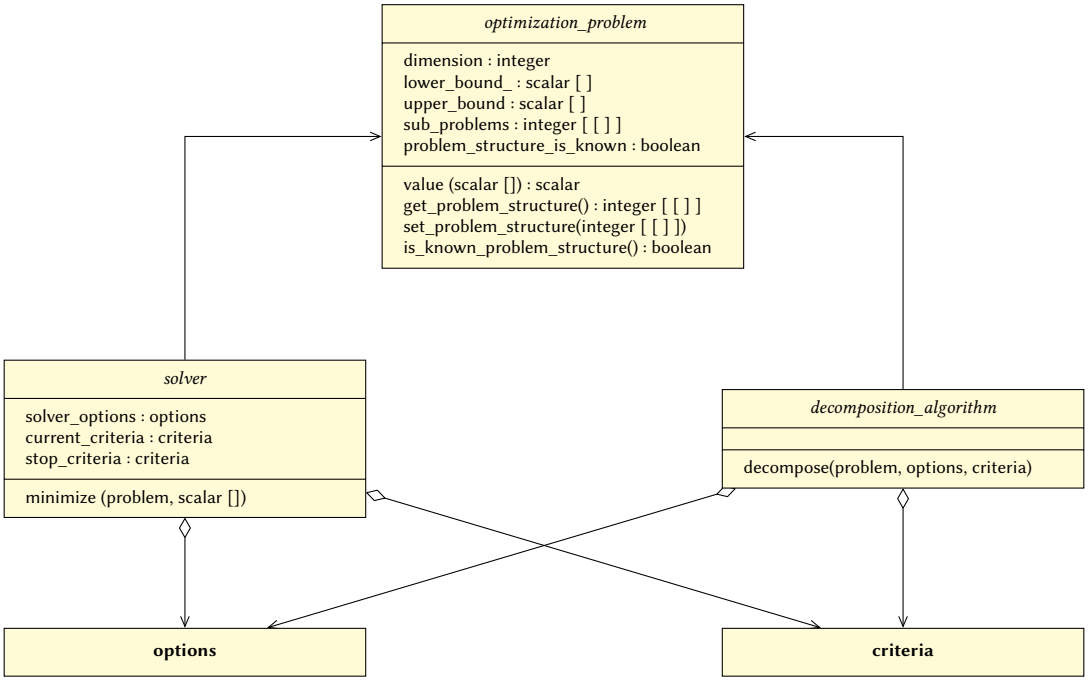

Figure 1: Class Diagram of Continuous Optimization Problem Decomposition (COPD) library.

in the same set indicate that there is an independent subproblem defined over them. This process continues until there is no pair of decision variables to analyze.

In the DG method, Theorem 4.1 is used to evaluate whether any pair of decision variables interact. To do that, the algorithm initializes two vectors, $\mathbf{x}$ and $\mathbf{x}'$, with the lower bound of the decision variables. To verify the interaction between the decision variables x_p and x_q, the variable x_p from the vector $\mathbf{x}'$ is changed to its upper bound, then, the value of Δ_1, is computed as follows:

$$\Delta_1 = f(\mathbf{x}) - f(\mathbf{x}') \tag{7}$$

After that, q-th variables from $\mathbf{x}$ and $\mathbf{x}'$ are set to their middle value. The new points generated from these changes will be called $\mathbf{y}$ and $\mathbf{y}'$, respectively. Then, a new Δ is computed, as follows:

$$\Delta_2 = f(\mathbf{y}) - f(\mathbf{y}') \tag{8}$$

Thus, an interaction between x_p and x_q is detected when the condition described in Equation 9 is true.

$$|\Delta_1 - \Delta_2| > \epsilon \tag{9}$$

where ϵ is a parameter of the DG algorithm which has to be defined by the user.

Notice that the value of ϵ directly influences on the interactions detected by this decomposition method, and its choice is not a trivial task. Another disadvantage of this algorithm is that it may incorrectly identify subproblems since it skips through the decision variables for which one interaction has already been identified, ignoring higher-order interactions. For instance, suppose $f(\mathbf{x}) = x_1x_2 + x_2x_3x_4 \mid -10.0 < x_k < 10.0$, in which the DG algorithm incorrectly split the whole problem into groups formed by (i) x_1 and x_2; and (ii) x_3 and x_4.

Even though DG is not the most used algorithm due to more function evaluations and its accuracy to identify the subproblems compared to the other decomposition strategies, its theoretical foundation is widely applied by other methods.

4.2 Differential Grouping - 2

An improved variant of the DG decomposition, the Differential Grouping - 2 (DG-2), was proposed in 2017 by, [21]. The DG2 method is also based on Theorem 4.1 however, to deal with the disadvantages as mentioned earlier of its predecessor, DG-2 does the following:

(1) Computes the raw interaction matrix which stores the values from $|\Delta_1 - \Delta_2|$ (see Equation 9) for all pairs of decision variables;
(2) Defines ϵ based on the raw interaction matrix by estimating the magnitude of round-off errors;
(3) Defines an adjacency matrix based on the raw interaction matrix and the epsilon values;
(4) Extracts the subproblems by analyzing the adjacency matrix.

The DG-2 method has shown to be more efficient and presented greater grouping accuracy than its predecessor. This version can reuse sample points generated for detecting interactions and automatically define a suitable threshold value (ϵ), respectively. Furthermore, the adjacency matrix available makes this algorithm useful for gray-box operators, given that the problem structure is mapped.

4.3 Extended Differential Grouping

The Extended Differential Grouping (XDG) method is also a variant of the DG method [24]. The main aim of this method is to identify direct and indirect interactions between the decision variables. Chronologically, XDG was proposed after the DG method and before its improved version, DG-2.

The XDG algorithm works similarly to the DG method in which computing the interaction between a pair of decision variables according to Equation 9. This method evaluates the interaction between all pairs of decision variables. If any decision variable interaction exists, it is mapped in an adjacency matrix kept by the XDG method. Afterward, the XDG can extract all subproblems just by analyzing the adjacency matrix.

Although the XDG method can be more accurate than DG, it stills requires a previous investigation of the suitable value for the ϵ parameter. On the other side, it can help develop the gray-box operator, given that the problem structure is known and available on the adjacency matrix.

4.4 Fast Interdependency Identification

The Fast Interdependency Identification (FII) algorithm is another well-known decomposition method proposed in 2017 by [12]. Its main aim was to reduce the function evaluations spent to decompose a continuous optimization problem keeping the grouping accuracy ratio.

Basically, the FII algorithm works in two stages to decompose an optimization problem. The first stage identifies and divides the set of decision variables into two groups: separable ($\mathbf{x}_{seps}$) and non-separable ($\mathbf{x}_{nonseps}$). Note that in this stage, the non-separable variables can still be separable into small problems. Then, given a decision variable x_p and a sample solution $\mathbf{x}$, its difference value $\Delta\mathbf{x}_p$ is calculated by Equation below:

$$\Delta\mathbf{x}_p = f(\mathbf{x}') - f(\mathbf{x}) \tag{10}$$

where $\mathbf{x}'$ is the solution $\mathbf{x}$ incremented by the value σ in x_p.

According to the authors, perturbing all other decision variables from the solution $\mathbf{x}$ produces a new solution $\mathbf{y}$. In this way, a new difference value ($\Delta\mathbf{y}_p$) is calculated, according to Equation 11:

$$\Delta\mathbf{y}_p = f(\mathbf{y}') - f(\mathbf{y}) \tag{11}$$

where $\mathbf{y}'$ is the solution $\mathbf{y}$ and the variable $x_p$equal the original value of the solution $\mathbf{x}$.

It classifies the decision variable x_p as separable if the Equation below is true:

$$|\Delta\mathbf{x}_p - \Delta\mathbf{y}_p| \leq \epsilon_1 \tag{12}$$

where ϵ is a parameter defined by the user that controls the interaction between the decision variables. Therefore, x_p is included on the set of separable variables, $\mathbf{x}_{seps}$. Otherwise, it is included on the non-separable set, $\mathbf{x}_{nonseps}$.

The second stage of the FII method is to identify possible subproblem from the non-separable decision variables. Thus, this method proposes perturbations on the remainder variables ($\mathbf{x}_{nonseps}$) for a solution $\mathbf{x}$, increasing a δ value on these variables and generating a solution $\mathbf{x}'$. Then, if a variable x_p interact with another variable x_q, the following Equation yields true results:

$$|\Delta\mathbf{x}_q - \Delta\mathbf{x}'_q| > \epsilon_2 \tag{13}$$

where ϵ_2 is a predefined value that is defined by the user.

Computational experiments [12] have shown that FII is more efficient concerning the number of function evaluations than previous methods. Notice, however, that four parameters must be set before the decomposition process can start. Thus, this setup step may require additional experiments.

The FII method may not be helpful to the development of gray-box operators since the adjacency matrix is not readily available. Therefore, extra computing is also required to extract the adjacency matrix.

4.5 Global Differential Grouping

Proposed by [16], the Global Differential Grouping (GDG) algorithm is another extension of the DG method. Also, based on Theorem 4.1, it was proposed to improve grouping accuracy compared to its predecessor.

The GDG algorithm does not immediately exclude a variable x_q from the set of all decision variables when an interaction is detected with variable x_p. With this characteristic, the GDG algorithm does not miss mapping all interactions compared to the DG method. Furthermore, this algorithm uses an adjacency matrix to map the interaction between the decision variables found.

To deal with the necessity to define a suitable value for the ϵ parameter, the authors propose computing it according to the following Equation:

$$\epsilon = \alpha \times \min\{|f(\mathbf{x}_1)|, \ldots, |f(\mathbf{x}_n)|\} \tag{14}$$

where n is the number of points randomly selected, and α is the controlling coefficient proposed by the authors. The user should define both parameters, n and α before the decomposition process started.

Setting the GDG parameters is not a trivial task. These parameters can directly influence the number of evaluation functions spent and the grouping accuracy of the algorithm. Concerning the use of this decomposition strategy to support the development of gray-box operators, how the adjacency matrix is automatically available at the process end, no extra process is required.

4.6 Recursive Differential Grouping

Proposed in 2018 by [25], the rationale of the Recursive Differential Grouping (RDG) method is recursively analyzing the decision variable interaction placing all interacting decision variables into the same group. For that, it based on the following Corollary:

Corollary 4.2. *Let f(**x**), $\mathbf{x}_1$ and $\mathbf{x}_1$ mutually exclusive subsets from the of decision variables where $\mathbf{x}_1 \cap \mathbf{x}_2 = \emptyset$. If there two unit vectors $\mathbf{u}_1$ and $\mathbf{u}_2$, two values l_1 and l_2 greater than 0, such that:*

$$f(\mathbf{x} + l_1\mathbf{u}_1 + l_2\mathbf{u}_2) - f(\mathbf{x} + l_2\mathbf{u}_2) \neq f(\mathbf{x} + l_1\mathbf{u}_1) - f(\mathbf{x}) \tag{15}$$

there is some interaction of at least one pair of decision variables between sets $\mathbf{x}_1$ and $\mathbf{x}_2$. Moreover, $U_{\mathbf{x}_1}$ is a subset of $U_{\mathbf{x}}$ such that any unit vector $\mathbf{u}_1 = (u_1, \ldots, u_D) \in U_{\mathbf{x}}$, where:

$$u_i = 0, \quad if\ x_i \notin \mathbf{x}_1 \tag{16}$$

and the same idea is applied to $U_{\mathbf{x}_2}$.

According to [25], the interaction between $\mathbf{x}_1$ and $\mathbf{x}_2$ subsets can be calculated setting a point $\mathbf{x}$ with the lower bound of all decision variables. Then, $\mathbf{x}'$ is defined by setting all decision variables with the lower bound and perturbing the variables from the subset $\mathbf{x}_1$ with the upper bound. After, the objective function value difference (δ_1) is calculated by:

$$\delta_1 = f(\mathbf{x}) - f(\mathbf{x}') \tag{17}$$

Perturbing the decision variables of the subset $\mathbf{x}_2$ from points $\mathbf{x}$ and $\mathbf{x}'$ with the middle value between the lower and upper bounds, we will generate the points $\mathbf{y}$ and $\mathbf{y}'$. Thus, the objective function value difference is described by:

$$\delta_2 = f(\mathbf{y}) - f(\mathbf{y}') \tag{18}$$

If $|\delta_1 - \delta_2| > \epsilon$, it represents that there is some interaction between the decision variables from subsets $\mathbf{x}_1$ and $\mathbf{x}_2$ where ϵ represents a control parameter from the RDG method to defined variable interaction. Similar to the GDG method, in the RDG algorithm, the ϵ is defined by Equation 14.

Unlike the other decomposition methods, the RDG algorithm works recursively identifying the interaction between the decision variables. It starts from the first variable from the subset $\mathbf{x_1}$, and if there is no interaction, it is classified as a separable decision variable. If some interaction is detected, the remainder of the decision variables and the recursive identification of the interaction process continue.

Lastly, the RDG method achieves a competitive grouping accuracy rate compared to the other decomposition methods [25]. However, the RDG requires a suitable parameter setting for the parameters n and α necessaries interaction threshold (ϵ). The RDG method for the development of gray-box operators is not fully compatible since the adjacency matrix is not known.

4.7 Recursive Differential Grouping - 2

A second version of the RDG decomposition method proposed by [27] in 2018 is Recursive Differential Grouping - 2 (RDG-2). Its predecessor, RDG [25], requires a parameter to estimate a threshold value used to verify if two subsets of variables interact between them or not. However, the RGD-2 method proposes a formulation used to compute values automatically for the ϵ parameter.

So, the formulation used by the RDG-2 to define suitable values for the ϵ parameter is:

$$\epsilon = \gamma \times (|f(\mathbf{x})| + |f(\mathbf{x}')| + |f(\mathbf{y})| + |f(\mathbf{y}')|) \tag{19}$$

where γ is defined by Equation below:

$$\gamma = (v \times \mu)/(1 - (v \times \mu)) \tag{20}$$

where $v = \sqrt{D} + 2$, μ represents machine epsilon[3], and D is the dimensionality of the optimization problem.

Even though the RDG-2 proposes an automatic definition of the set parameters required by its predecessor, it still incompatible with gray-box operator development due to the adjacency matrix is not known.

[3]It represents the difference between 1.0 and the next value representable by the floating-point. For C++, see epsilon function on numeric_limits.

4.8 Recursive Differential Grouping - 3

A new version of RDG-2, called Recursive Differential Grouping - 3 (RDG-3), was proposed in 2019 by [26]. The main difference between the RDG-3 from its predecessors is its capability to decompose overlapping decision variables. Overlapping optimization problems share a set of decision variables with two or more subproblems. Figure 2 presents a problem example in which a decision variable, x_3, is shared by two subproblems.

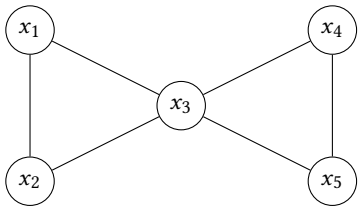

Figure 2: Variable Interaction Graph for an optimization problem example with a shared decision variable.

To deal with overlapping decision variables, the authors propose a parameter ϵ_n to limit the group size of the decision variables for overlapping problems. Therefore, this method only includes indirect interactions in a group $\mathbf{x}_1$ if the predefined limit for the group sizes (ϵ_n) is still respected.

The computational experiments proposed by [26] showed a similar grouping accuracy rate than their predecessors, RDG and RDG-2. However, how the RDG-3 does still not automatically offer an adjacency matrix; it is not compatible with the development of gray-box operators.

5 COMPARISON

This section summarizes the characteristics of the decomposition methods implemented in the COPD library. They are compared from the following perspectives:

(1) Support for the development of gray-box operators, i.e., whether the decomposition methods produce a variable interaction graph (VIG) that gray-box methods can exploit;
(2) The number of parameters settings that influence the decomposition methods accuracy; and
(3) The accuracy of the methods.

Table 1 shows which of the methods compute a Variable Interaction Graph (VIG) automatically. As mentioned before, VIG is essential for the gray-box operators recently proposed in the literature. From all methods of COPD library, DG-2, XDG, and GDG are the only method from the COPD library able to provide the VIG.

Table 2 relates the decomposition methods and the number of parameters necessary to their working. Only two methods are parameter-free, DG-2 and RDG-2. DG, XDG, and RDG-3 require that one parameter must be set. GDG and RDG-2 require two setting parameters. Lastly, the FII algorithm requires the setting of four parameters.

Table 1: Variable Interaction Graph Available Analysis

Variable Interaction Graph Available	Methods
Yes	DG-2, XDG, GDG
No	DG, FII, RDG, RDG-2, RDG-3

Table 2: Decomposition Method Parameters

Number of Parameters	Methods
1	DG, XDG, RDG-3
2	GDG, RDG
4	FII
Parameter-Free	DG-2, RDG-2

Decomposition methods are best suited to handle large-scale optimization problems, consisting of a combination of independent subproblems. In order to analyze the accuracy of the decomposition methods in finding these subproblems, we run them in the best-known set of large-scale problems [13].

The parameters used for the computational experiments are proposed in related works, such as (i) DG, $\epsilon = 0.1$ [20]; (ii) XDG, $\epsilon = 0.1$ [24], (iii) FII, $\epsilon_1 = \epsilon_2 = 0.01$ and $\sigma = \delta = 10$ [12], (iv) GDG, $k = 10$ and $\alpha = 1 \times 10^{-10}$ [16], (v) RDG, $k = 10$ and $\alpha = 1 \times 10^{-12}$ [25], and (vi) RDG-3, $\epsilon_n = 50$ [26].

In addition to comparing the number of function evaluations spent to analyze each optimization problem, we also compare the grouping identification accuracy according to the ratio defined in Equation 21. The accuracy ratio is based on $correct\_identification\_groups$ and $ideal\_groups$, which represents the number of subproblems correctly identified and the number of real groups, respectively.

$$accuracy = \frac{correct\_identification\_groups}{ideal\_groups} \tag{21}$$

Table 3 presents the computational experiment results of decomposition methods on the CEC'2013 benchmark functions. These results compare the accuracy, and the number of function evaluations spent to analyze the optimization problems.

RDG-2 was the most accurate, 79.3%, and second in the average number of function evaluations, 14612. Its success can be explained by the adaptive ϵ selection mechanism and the recursive decomposition procedure, which saves several function evaluations. The RDG-3 has similar mechanisms, but its linkage breaking mechanism tends to over-decompose the problems, as seen in problems f_{12}, f_{13}, and f_{14}.

From the methods which produce a VIG, important in gray-box optimizers, DG-2 presented the highest accuracy, 72.7%. Its success can be explained again by the adaptive ϵ selection mechanism shared with RDG-2 and RDG-3.

Nevertheless, the comparative performance of the decomposition methods can be influence by the choice of parameters and the set of benchmark problems. Having this in mind, it is possible to see the relevance of a decomposition methods library to the development of related studies.

6 SUMMARY AND CONCLUSIONS

In this paper, we have presented a C++ library called Continuous Optimization Problem Decomposition. The decomposition strategies try to break the optimization problem into independent subproblems that the optimizer can solve separately. This paper focuses only on decomposition methods, commonly used to find the subcomponents from a problem before the optimization process starts.

Our library allows users to easily describe continuous optimization problems and analyze their structure through the main decomposition methods found in the literature. Furthermore, it facilitates the application of decomposition, providing the user with implementations of the main methods speeding up the solution of complex optimization problems.

To improve the library and promote its use for both designing new decomposition techniques and benchmarking large-scale optimization algorithms, in the future, it should provide a more natural interface for the exportation of the performance metrics computed over the search course.

ACKNOWLEDGMENTS

This work has been supported by the Brazilian Agencies State of Minas Gerais Research Foundation - FAPEMIG (APQ-00040- 14); Coordination for the Improvement of Higher Level Personnel - CAPES, Brazil; National Council of Scientific and Technological Development - CNPq, Brazil (402956/2016-8); and UFOP, Brazil.

REFERENCES

[1] A. Banitalebi, M. Aziz, and Z. Aziz. 2016. A self-adaptive binary differential evolution algorithm for large scale binary optimization problems. *Information Sciences* 367-368 (2016), 487–511.
[2] Thomas Bartz-Beielstein, Carola Doerr, Jakob Bossek, Sowmya Chandrasekaran, Tome Eftimov, Andreas Fischbach, Pascal Kerschke, Manuel López-Ibáñez, Katherine Mary Malan, Jason H. Moore, Boris Naujoks, Patryk Orzechowski, Vanessa Volz, Markus Wagner, and Thomas Weise. 2020. Benchmarking in Optimization: Best Practice and Open Issues. *CoRR* abs/2007.03488 (2020). arXiv:2007.03488 https://arxiv.org/abs/2007.03488
[3] J. Brest, B. Bošković, S. Greiner, V. Žumer, and M. Maučec. 2007. Performance comparison of self-adaptive and adaptive differential evolution algorithms. *Soft Computing* 11 (2007), 617–629.
[4] A. Cano and C. García-Martínez. 2016. 100 Million dimensions large-scale global optimization using distributed GPU computing. In *2016 IEEE Congress on Evolutionary Computation (CEC)*. IEEE – Institute of Electrical and Electronic Engineers, Vancouver, BC, Canada, 3566–3573.
[5] S. Das and P. N. Suganthan. 2011. Differential Evolution: A Survey of the State-of-the-Art. *IEEE Transactions on Evolutionary Computation* 15, 1 (2011), 4–31.
[6] W. Gao, S. Liu, and L. Huang. 2012. Particle swarm optimization with chaotic opposition-based population initialization and stochastic search technique. *Communications in Nonlinear Science and Numerical Simulation* 17, 11 (2012), 4316 – 4327.
[7] T. M. Gomes, A. R. R. de Freitas, and R. A. Lopes. 2019. Multi-Heap Constraint Handling in Gray Box Evolutionary Algorithms. In *Proceedings of the Genetic and Evolutionary Computation Conference* (Prague, Czech Republic). Association for Computing Machinery, New York, NY, USA, 829–836.
[8] P. Guo, W. Cheng, and Y. Wang. 2016. Hybrid evolutionary algorithm with extreme machine learning fitness function evaluation for two-stage capacitated facility location problem. *Expert Systems with Applications* 71 (2016), 57–68.
[9] R. Haftka, D. Villanueva, and A. Chaudhuri. 2016. Parallel surrogate-assisted global optimization with expensive functions – a survey. *Structural and Multidisciplinary Optimization* 54 (2016), 3–13.
[10] N. Hansen. 2006. *The CMA Evolution Strategy: A Comparing Review.* Springer Berlin Heidelberg, Berlin, Heidelberg, 75–102.
[11] W. E. Hart, N. Krasnogor, and J. E. Smith. 2005. *Memetic Evolutionary Algorithms.* Springer Berlin Heidelberg, Berlin, Heidelberg, 3–27.

Table 3: Comparison of the Decomposition Results on CEC'2013 Benchmark Functions

	Seps	Non-Seps	Groups	Accuracy (%)	Function Evaluations
				DG / DG-2 / XDG / FII / GDG / RDG / RDG-2 / RDG-3	
f_1	1000	0	1000	100 / 100 / 100 / 100 / 100 / 100 / 100 / 100	1998000 / 500501 / 1001000 / 3001 / 501511 / 3008 / 2998 / 2998
f_2	1000	0	1000	100 / 100 / 100 / 100 / 100 / 100 / 100 / 100	1998000 / 500501 / 1001000 / 3001 / 501511 / 3008 / 2998 / 2998
f_3	1000	0	1000	100 / 0 / 100 / 100 / 0.1 / 0 / 0 / 0	1998000 / 500501 / 1001000 / 3001 / 501511 / 6005 / 5992 / 5992
f_4	700	300	707	5.94 / 100 / 1.27 / 62.6 / 100 / 100 / 100 / 100	31640 / 500501 / 1001000 / 4523 / 501511 / 9842 / 9832 / 9823
f_5	700	300	707	99.7 / 100 / 99.8 / 99.8 / 99.8 / 100 / 100 / 99.8	1323408 / 500501 / 1001000 / 4216 / 501511 / 10145 / 9895 / 9814
f_6	700	300	707	99.2 / 0.99 / 99.4 / 99.4 / 99.7 / 4.38 / 0.99 / 1.13	1471716 / 500501 / 1001000 / 3599 / 501511 / 13574 / 11587 / 12019
f_7	700	300	707	9.05 / 100 / 0.70 / 79.6 / 100 / 95.1 / 100 / 100	22852 / 500501 / 1001000 / 4432 / 501511 / 8219 / 9814 / 9805
f_8	0	1000	20	70.0 / 90.0 / 0 / 80.0 / 55.0 / 75.0 / 90.0 / 85.0	45248 / 500501 / 1001000 / 25360 / 501511 / 19412 / 19405 / 19342
f_9	0	1000	20	75.0 / 100 / 90.0 / 95.0 / 85.0 / 100 / 100 / 100	36560 / 500501 / 1001000 / 21442 / 501511 / 19247 / 19156 / 19126
f_{10}	0	1000	20	25.0 / 100 / 60.0 / 55.0 / 85.0 / 85.0 / 100 / 95.0	220408 / 500501 / 1001000 / 15419 / 501511 / 19142 / 19879 / 19735
f_{11}	0	1000	20	30.0 / 100 / 0 / 50.0 / 40.0 / 0 / 100 / 100	22124 / 500501 / 1001000 / 14409 / 501511 / 11078 / 19429 / 19399
f_{12}	0	1000	1	0 / 100 / 100 / 100 / 100 / 100 / 100 / 0	1000000 / 500501 / 1001000 / 503500 / 501511 / 50876 / 50866/49891
f_{13}	0	905	1	0 / 0 / 0 / 0 / 0 / 0 / 0 / 0	12412 / 409966 / 819930 / 6786 / 410881 / 8315 / 15187 / 15988
f_{14}	0	905	1	0 / 0 / 0 / 0 / 0 / 100 / 100 / 0	26620 / 409966 / 819930 / 8467 / 410881 / 9272 / 16150 / 16288
f_{15}	0	1000	1	100 / 100 / 100 / 100 / 0 / 100 / 100 / 100	3996 / 500501 / 1001000 / 4001 / 501511 / 6071 / 5992 / 5992
		Mean		54.2 / 72.7 / 56.7 / 74.7 / 64.3 / 70.6 / 79.3 / 72.0	680732 / 488429 / 976857 / 74844 / 489427 / 13147 / 14612 / 14614

[12] X. Hu, F. He, W. Chen, and J. Zhang. 2017. Cooperation coevolution with fast interdependency identification for large scale optimization. *Information Sciences* 381 (2017), 142–160.

[13] Xiaodong Li, Ke Tang, Mohammmad Nabi Omidvar, Zhenyu Yang, and Kai Qin. 2013. Benchmark Functions for the CEC'2013 Special Session and Competition on Large-Scale Global Optimization. (01 2013).

[14] D. Lim, Y. Ong, Y. Jin, and B. Sendhoff. 2007. A Study on Metamodeling Techniques, Ensembles, and Multi-Surrogates in Evolutionary Computation. In *Proceedings of the 9th Annual Conference on Genetic and Evolutionary Computation* (London, England). Association for Computing Machinery, New York, NY, USA, 1288–1295.

[15] S. Mahdavi, M. Shiri, and S. Rahnamayan. 2015. Metaheuristics in Large-Scale Global Continues Optimization. *Information. Sciences* 295 (2015), 407–428.

[16] Y. Mei, M. Omidvar, X. Li, and X. Yao. 2016. A Competitive Divide-and-Conquer Algorithm for Unconstrained Large-Scale Black-Box Optimization. *ACM Trans. Math. Softw.* 42, 2 (2016), 24 pages.

[17] V. Melo and A. Delbem. 2012. Investigating Smart Sampling as a population initialization method for Differential Evolution in continuous problems. *Information Sciences* 193 (2012), 36–53.

[18] D. Molina. 2016. Evolutionary algorithms for large-scale global optimisation: a snapshot, trends and challenges. *Progress in Artificial Intelligence* 5 (2016), 85–89.

[19] D. Molina and A. La Torre. 2020. *IEE CEC 2019 - Special Session and Competition on Large-Scale Global Optimization.* https://www.tflsgo.org/special_sessions/cec2019.html

[20] M. N. Omidvar, X. Li, Y. Mei, and X. Yao. 2014. Cooperative Co-Evolution With Differential Grouping for Large Scale Optimization. *IEEE Transactions on Evolutionary Computation* 18, 3 (2014), 378–393.

[21] M. N. Omidvar, M. Yang, Y. Mei, X. Li, and X. Yao. 2017. DG2: A Faster and More Accurate Differential Grouping for Large-Scale Black-Box Optimization. *IEEE Transactions on Evolutionary Computation* 21, 6 (2017), 929–942.

[22] R. C. P. Silva, R. A. Lopes, A. R. R. Freitas, and F. G. Guimaraes. 2014. A study on self-configuration in the differential evolution algorithm. In *2014 IEEE Symposium on Differential Evolution (SDE)*. IEEE Symposium on Differential Evolution (SDE), Orlando, FL, USA, 1–8.

[23] R. Storn and K. Price. 1997. Differential Evolution – A Simple and Efficient Heuristic for GloSbal Optimization over Continuous Spaces. *Journal of Global Optimization* 11, 4 (1997), 341–359.

[24] Y. Sun, M. Kirley, and S. Halgamuge. 2015. Extended Differential Grouping for Large Scale Global Optimization with Direct and Indirect Variable Interactions. In *Proceedings of the 2015 Annual Conference on Genetic and Evolutionary Computation* (Madrid, Spain). Association for Computing Machinery, New York, NY, USA, 313–320.

[25] Y. Sun, M. Kirley, and S. K. Halgamuge. 2018. A Recursive Decomposition Method for Large Scale Continuous Optimization. *IEEE Transactions on Evolutionary Computation* 22, 5 (2018), 647–661.

[26] Y. Sun, X. Li, A. Ernst, and M. N. Omidvar. 2019. Decomposition for Large-scale Optimization Problems with Overlapping Components. In *2019 IEEE Congress on Evolutionary Computation (CEC)*. IEEE Congress on Evolutionary Computation (CEC), Wellington, New Zealand, 326–333.

[27] Y. Sun, M. Omidvar, M. Kirley, and X. Li. 2018. Adaptive Threshold Parameter Estimation with Recursive Differential Grouping for Problem Decomposition. In *Proceedings of the Genetic and Evolutionary Computation Conference* (Kyoto, Japan). Association for Computing Machinery, New York, NY, USA, 889–896.

[28] R. Tinós, D. Whitley, and F. Chicano. 2015. Partition Crossover for Pseudo-Boolean Optimization. In *Proceedings of ACM Conference on Foundations of Genetic Algorithms* (Aberystwyth, United Kingdom). Association for Computing Machinery, New York, NY, USA, 137–149.

[29] D. Whitley and W. Chen. 2012. Constant Time Steepest Descent Local Search with Lookahead for NK-Landscapes and MAX-KSAT. In *Proceedings of the 14th Annual Conference on Genetic and Evolutionary Computation* (Philadelphia, Pennsylvania, USA). Association for Computing Machinery, New York, NY, USA, 1357–1364.

[30] Z. Yang, K. Tang, and X. Yao. 2008. Large scale evolutionary optimization using cooperative coevolution. *Information Sciences* 178 (2008), 2985–2999.