Mathematical and Computational Applications

MDPI

Article

Variable Decomposition for Large-Scale Constrained Optimization Problems Using a Grouping Genetic Algorithm

Guadalupe Carmona-Arroyo *, Marcela Quiroz-Castellanos * and Efrén Mezura-Montes

Artificial Intelligence Research Institute, Universidad Veracruzana, Campus Sur, Calle Paseo Lote II, Sección Segunda 112, Nuevo Xalapa, Veracruz 91097, Mexico; emezura@uv.mx

* Correspondence: gcarmonaarroyo@gmail.com (G.G.-A.); maquiroz@uv.mx (M.Q.-C.)

Abstract: Several real optimization problems are very difficult, and their optimal solutions cannot be found with a traditional method. Moreover, for some of these problems, the large number of decision variables is a major contributing factor to their complexity; they are known as Large-Scale Optimization Problems, and various strategies have been proposed to deal with them. One of the most popular tools is called Cooperative Co-Evolution, which works through a decomposition of the decision variables into smaller subproblems or variables subgroups, which are optimized separately and cooperate to finally create a complete solution of the original problem. This kind of decomposition can be handled as a combinatorial optimization problem where we want to group variables that interact with each other. In this work, we propose a Grouping Genetic Algorithm to optimize the variable decomposition by reducing their interaction. Although the Cooperative Co-Evolution approach is widely used to deal with unconstrained optimization problems, there are few works related to constrained problems. Therefore, our experiments were performed on a test benchmark of 18 constrained functions under 100, 500, and 1000 variables. The results obtained indicate that a Grouping Genetic Algorithm is an appropriate tool to optimize the variable decomposition for Large-Scale Constrained Optimization Problems, outperforming the decomposition obtained by a state-of-the-art genetic algorithm.

Keywords: Grouping Genetic Algorithm; variable decomposition; Large-Scale Constrained Optimization

Citation: Carmona-Arroyo, G.; Quiroz-Castellanos, M.; Mezura-Montes, E. Variable Decomposition for Large-Scale Constrained Optimization Problems Using a Grouping Genetic Algorithm. *Math. Comput. Appl.* **2022**, *27*, 23. https://doi.org/10.3390/mca27020023

Academic Editor: Claudia Schillings

Received: 26 January 2022
Accepted: 1 March 2022
Published: 3 March 2022

1. Introduction

A constrained numerical optimization problem is defined by finding the vector $\mathbf{x} \in \mathbb{R}^D$ that minimizes the objective function $Obj(\mathbf{x})$ subject to inequality $g_j(\mathbf{x})$ and equality $h_k(\mathbf{x})$ constraints [1]. This is described by Equation (1).

$$\begin{aligned} &\text{minimize} \quad Obj(\mathbf{x}) \\ &\text{subject to} \\ &\qquad g_j(\mathbf{x}) \leq 0, \quad j = 1, \ldots, q \\ &\qquad h_k(\mathbf{x}) = 0, \quad k = 1, \ldots, r. \end{aligned} \tag{1}$$

where q and r represent the number of inequality and equality constraints, respectively, $\mathbf{x} = (x_1, \ldots, x_D)$, and the search space $\mathbb{S}$ is defined by the lower limits l and upper limits u ($l_i \leq x_i \leq u_i$), while the feasible region is defined as the subset of solutions that satisfy the constraints of the problem $\mathbb{F} \subset \mathbb{S}$.

To handle the constrained problem, the constraint violation sum cvs [2], is calculated by Equation (2).

$$cvs(\mathbf{x}) = \sum_{j}^{q} \max(0, g_j(\mathbf{x})) + \sum_{k}^{r} \max(0, |h_k(\mathbf{x})| - \epsilon) \tag{2}$$

where $|h_k(\mathbf{x})| - \epsilon$ is the transformation of equality constraints into inequality constraints $|h_k(\mathbf{x})| - \epsilon \leq 0$ with $\epsilon = 1 \times 10^{-4}$.

According to the specialized literature [3,4], a Large-Scale Optimization Problem consists of 100 or more variables, while benchmark functions for sessions and competitions on the field include thousands of decision variables. Algorithms that solve Large-Scale Optimization Problems are usually affected by the curse of dimensionality; i.e., these problems are more complex to solve when the number of decision variables increases. One of the best-known approaches to deal with these problems is the one proposed by Potter and De Jong called Cooperative Co-Evolution (CC) [5], which is based on the divide-and-conquer strategy. This CC approach works in three stages: (1) first, the problem is decomposed into subcomponents of less dimension and complexity; then, (2) each subproblem is optimized separately; and finally, (3) the solutions of each subproblem cooperate to create the solution of the original problem.

Although many of the approaches to solve Large-Scale Optimization Problems have implemented the CC approach, the first problem that arises is to find the adequate decomposition of the subgroups since the interaction among the variables must be taken into account to divide the problem. In other words, if two or more variables interact with each other, they must remain in the same subcomponent, just as the variables that do not interact with others must be part of subcomponents with just one variable. The decomposition of the subgroups can be evaluated considering definitions of problem separability and partial separability, as explained in Section 3.2. If the interacting variables are not grouped into the same subgroup, CC tends to find a solution that is not the optimum of the original problem but a local optimum introduced by an incorrect problem decomposition [6].

Several strategies have been proposed in the literature to deal with the problem of creating an adequate decision variable decomposition, ranging from random approaches to strategies that study the interaction among variables to optimize this decomposition. When the original problem is decomposed into subproblems, we aim for the interaction between them to be at minimum. For this reason, we can work with the decomposition through optimization strategies, where the objective is to group variables that interact with each other in the same subcomponent.

One of the first works related to the optimization of the variable decomposition for Large-Scale Constrained Problems was proposed by Aguilar-Justo et al. [7], who presented a Genetic Algorithm (GA) to handle the interaction minimization in the subcomponents. This GA and its operators, such as crossover and mutation, work under an integer genetic encoding, which is one of the most popular ways of representing a solution as a chromosome in this type of algorithm.

In this work, we resort to a Grouping Genetic Algorithm (GGA) to solve the decomposition problem, since these algorithms have proven to be some of the best when it comes to combinatorial optimization problems where the optimization of elements in groups is involved [8]. This proposal aims to show the benefits of using a GGA and its group-based representation, for the creation of subcomponents, compared against a genetic algorithm. In addition, to the best of the authors' knowledge, our proposal is the first GGA approach to handle decomposition in Large-Scale Constrained Optimization Problems.

We chose similar main operators and parameters to the genetic algorithm proposed by Aguilar-Justo et al. [7] in order to evaluate the impact of the representation schemes for the decomposition problem and to make a fair comparison of the performance.

Both algorithms were evaluated on a set of 18 test functions proposed by Sayed et al. [3], which are problems with 1, 2, and 3 constraints with 100, 500, and 1000 variables, respectively. Experimental results show that the proposed GGA obtains a suitable variable decomposition when compared against the GA of Aguilar-Justo et al. [7] for variable decomposition in Large-Scale Constrained Optimization Problems, especially where the separation is more complicated, such as in non-separable problems.

The work continues as follows: in the next section, we show related work regarding Decomposition Methods and Grouping Genetic Algorithms. In Section 3, we show our

proposed GGA and describe each of its components in detail. Section 4 contains the experiments and results of our algorithm compared to a genetic algorithm and a brief analysis of the performance of the GGA. Finally, in Section 5, we describe the conclusions and future work corresponding to our research.

2. Related Works

2.1. Decomposition Methods

According to Ma et al. [6], several variable grouping strategies have been proposed in the specialized literature for variable decomposition to deal with Large-Scale Unconstrained Optimization Problems. They classify them into the next classes: static variable grouping, random variable grouping, variable grouping based on interaction learning, variable grouping based on domain knowledge, overlap and hierarchical variable grouping, and finally, some hybrids of them. In the next paragraphs, we describe the works related to each class.

Static variable grouping methods do not rely on any intelligent procedure to create the variable decomposition. Instead, they preliminarily decompose the decision vector into a set of low-dimensional subcomponents and fix the variable grouping during the process of optimization. Among the works that perform static grouping of variables are the works of Potter and De Jong [5] and Liu et al. [9], which show good performance with fully separable problems. For non-separable problems, Van den Bergh and Engelbrecht [10] propose a static sequential decomposition. However, the decomposition process is dependent on an adequate number of subcomponents that must be adequate from the beginning of the strategy, and the static decomposition process has poor performance in many problems.

For that reason, several authors proposed *random variable grouping* strategies, such as the case of Yang et al. [11], who present random decomposition with a fixed number of subcomponents. Moreover, the same authors propose in [12] a random decomposition with a dynamic number of subcomponents. Omidvar et al. [13,14] improve these random strategies by integrating the probability of interaction between variables in the grouping technique.

In addition, if an algorithm can learn the structure of the problem and decompose it, the difficulty in solving the problem can be significantly reduced (*variable grouping based on interaction learning*). Many approaches have been proposed to detect variable interactions, and we can subdivide them into those based on perturbation, statistical models, distribution models, approximate models, and linkage adaptation. For example, in some cases, the interaction is captured by perturbing the decision variables and measuring the change in fitness caused by the perturbations, like in the work of Xu et al. [15]. Furthermore, Differential Grouping (DG) and Differential Grouping version 2 from Odmivar et al. [16,17], respectively, are based on perturbation as well, which are among the most popular decomposition algorithms and have been highly studied, which has led to various improvements, such as recursive decomposition [18–21]. Moreover, Delta Grouping [14] is classified as a decomposition method based on statistical models, where all variables and the objective functions are considered as random variables. Statistical analyses of variables or objective functions are performed first, and then the variables are grouped. In a distribution model, the set of promising solutions is first used to estimate the variable distributions and variable interactions, and then it is taken to generate new candidate solutions, based on the learned variable distributions and variable interactions: such is the case of Estimation of Distribution Algorithm (EDA), as is the case with the work of Sopov [22], where a genetic algorithm is combined with an EDA for collecting statistical data based on the past search experience to provide the problem decomposition by fixing genes in chromosomes, as well as other representatives of such methods [23,24]. As an example of an approximate model, the fitness evaluation of a Large-Scale Continuous Optimization Problem is converted to the evaluation of a simpler, partially separable problem in [25]. Linkage adaption methods use specially designed evolution operators, representations, and mechanisms to divide variables into groups [26].

When it comes to *overlap and hierarchy variable grouping*, which are usually interspersed in the decision vector, more elaborate strategies have to be used; Goh et al. [27,28] suggested assigning each variable to several subcomponents, each of which contains more than one variable, and the subcomponents compete with each other to represent shared variables. Furthermore, Strasser et al. [29] introduce some overlapping grouping strategies, including random overlapping grouping, neighbor overlapping grouping, centered overlapping grouping, and more.

Regarding the *variable grouping based on domain knowledge*, before CC is implemented to solve specific real-world problems, domain knowledge can be harnessed to reduce the complexity of the problems. The conflicting probability of two flights was used by Guan et al. [30] to learn the variable interaction in solving the flight conflicting avoidance problem, which is one of several examples of real optimization problems where domain knowledge can be a good tool.

All the works mentioned above focus on unconstrained problems. One of the first decomposition methods for solving Large-Scale Constrained Optimization Problems is an extension of the work of Sayed et al. [25]; this new version is known as the Variable Interaction Identification Technique for Constrained Problems (VIIC) [3]. This method can find the interaction between variables in problems of 100, 500, and 1000 dimensions with inequality constraints. Sayed et al. proposed to measure each decomposition of variables by minimizing the absolute difference between the full evaluation and the sum of each evaluated subgroup based on the definition of the separability and partial separability of a function. The approach was tested at a new benchmark and compared to Random Grouping (RG), and the results showed that VIIC outperformed RG. Later, Aguilar-Justo and Mezura-Montes [31] improved the performance of the VIIC to achieve adequate decomposition for a fixed number of subgroups. They transformed the constrained problem into an unconstrained problem and used a neighborhood heuristic to guide the search by their proposed decomposition and then optimize it (called VIICN). After that, they proposed an improvement to their work where the principles of VIIC and VIICN are used to build a genetic algorithm that performs a dynamic decomposition which they called DVIIC, without establishing a fixed number of subcomponents [7]. Recently, Vakhnin and Sopov [32] proposed a method based on CC that increases the size of groups of variables at the decomposition stage (called iCC), working with a fixed number of subcomponents.

Intending to improve the existing methods for the decomposition of variables, we propose a genetic algorithm with a genetic encoding based on groups, better known as the Grouping Genetic Algorithm, to optimize the variables decomposition. Experimental results demonstrate the benefits of using a group-based encoding scheme for this problem and its advantages over the genetic algorithm with an integer-based encoding scheme (DVIIC [7]).

2.2. Grouping Genetic Algorithms

As we have mentioned before, the variable decomposition problem is an optimization problem because we search for the best decomposition, in the sense of the variable interaction; that is, given a set $X = x_1, x_2, \ldots, x_D$ of D variables, we want to decompose said set into m disjoint groups, so that the variables within each group do not interact with the variables of the other groups. Therefore, we see our problem as a grouping problem.

According to the literature [8], grouping problems are a type of combinatorial optimization problem where a set X of D items is usually partitioned into a collection of m mutually disjoint subsets (groups) G_j, so that $X = \cup_{j=1}^{m} G_j$, and $G_j \cap G_k = \emptyset$, $j \neq k$. In this way, an algorithm designed to solve a grouping problem seeks the best possible distribution of the D items of the set X in m different groups ($1 \leq m \leq D$), such that each item is exactly in one group.

Kashan et al. [33] organized the grouping problems in three categories, using as criteria the number of groups, the type of groups, and the dependence on the order of the groups. First, using the number of groups as the criterion, grouping problems can be classified

as either constants or variables. In this sense, if the number of groups required is known, the problem is constant. On the other hand, if the number of groups is unknown, the problem is variable. Second, these problems can be divided into identical and non-identical groups, considering the characteristics of their groups. In this classification, if the quality of a solution is modified by exchanging all the items of two groups, that problem belongs to the non-identical grouping class. Otherwise, the problem is part of the identical category. Finally, grouping problems are called order-dependent when the solution quality depends on the groups' order. Consequently, grouping problems without this dependency belong to the not order-dependent class. Thus, we can say that the decomposition problem is a variable, identical, and not order-dependent grouping problem.

Grouping problems are very difficult to solve. Most of them are NP-hard, which implies there is no algorithm capable of finding an optimal solution for every instance in polynomial time [34]. There are several NP-hard grouping problems, such as the Bin Packing Problem, Job Shop Scheduling Problem, etc. Ramos-Figueroa et al. [8] studied in their work the strategies that work with most problems like the ones mentioned before. They concluded that the best and the most popular strategies to solve such problems are the Grouping Genetic Algorithms or GGAs.

The GGA was designed in 1992 by Falkenauer [35] and is an extension to the traditional GA with the difference of using a group-based solutions representation scheme and variation operators working together with such solution encoding. Ramos-Figueroa et al. [36] remark that the encoding of a grouping problem solution into a chromosome is a key issue for obtaining good GGA performance; the authors also comment on the importance of integrating crossover and mutation operators adapted to work at the group level. They present a survey of different variation operators designed to work with GGAs that use different types of encoding, as well as their advantages to solve grouping problems.

The state-of-the-art [8] indicates that some of the best results when solving NP-hard grouping problems have been obtained by GGAs that combine grouping encoding schemes and special operators adapted to work with these genetic encodings. Moreover, GGAs have been highly studied for grouping problems that have similarities with the variable decomposition problem, which is also due to the exploration and exploitation of the search space that is given by the nature of the elements of evolutionary algorithms [37]. In this work, we propose, to the best of the authors' knowledge, the first GGA for the variable decomposition problem in Large-Scale Constrained Optimization Problems. A comparative study is conducted to evaluate the performance of our Grouping Genetic Algorithm versus the genetic algorithm DVIIC [7] on the decomposition of variables for Large-Scale Constrained Optimization Problems. To promote a fair comparison, we implement similar operators and equivalent parameter settings. The experiments were carried out using 18 test functions each with 100, 500, and 1000 variables. The obtained results allow us to validate the advantages of the group-based encoding over the integer-based encoding.

3. A Grouping Genetic Algorithm for the Variable Decomposition Problem

The variable decomposition problem can be classified as a grouping problem. We seek to optimize the separation into groups of the decision variables of the Large-Scale Problem; that is, to create the best partition of the decision variables into a collection of m mutually disjoint groups so that the variables belonging to each group have no interaction with the variables of another group.

To study the importance of the solution encoding in a genetic algorithm to solve the variable decomposition problem, we decided to develop a GGA with operators and parameters with similar features to the genetic algorithm DVIIC (proposed by Aguilar-Justo et al. [7]) so that the comparison is as fair as possible. The main difference between the two algorithms is the genetic encoding. The proposal of Aguilar-Justo et al. [7] includes an integer-based representation, where a chromosome has a fixed length that is equal to the number of variables, and each gene represents a variable and indicates the group where the variable is set. On the other hand, our GGA includes a group-based representation, where a chromo-

some can have a variable length, equal to the number of subcomponents, and each gene represents a subcomponent and indicates the variables that belong to this subset.

In Algorithm 1, we show the general steps of the GGA proposed in this work. The precise details are shown in the following subsections. The process begins by generating an initial population P of pop_size individuals created by the population initialization strategy (Line 1). After that, each of the individuals in the population is evaluated, and the best solution for the population is obtained (Line 2). Then, we iterate through a max_gen number of generations or until we find a value equal to zero in the decomposition evaluation. Within this cycle, the individuals to be crossed will be selected, and the offspring will be created through the grouping crossover operator (Lines 4–5). Similarly, the population is updated by the mutation of some individuals in the population (Lines 6–7). Finally, the population is evaluated again to update the population and the best global solution found so far (Lines 8–9).

Algorithm 1: Grouping Genetic Algorithm for variable decomposition algorithm.

```
1  Generate an initial population P
2  Evaluate population and save best_solution
3  while generation ≤ max_gen and grps_diff ≠ 0 do
4      Select n pairs of individuals for crossover
5      Apply grouping crossover operator
6      Select n individuals for mutation
7      Apply grouping mutation operator
8      Evaluate offspring and update best_solution
9      Apply replacement strategy of the population
10 Return best_solution
```

In the following subsections, we detail the components and operators of our GGA.

3.1. Genetic Encoding

One of the most important decisions to make while implementing a genetic algorithm is to decide the representation to use to represent the solutions. It has been observed that improper representation can lead to poor performance of the GA. Our GGA works with group-based representation, which is the main characteristic of the GGAs.

Each individual in the population is represented by the groups of variables. Figure 1 shows an example of an individual that represents a problem with 10 variables numbered from 0 to 9 randomly assigned to four subcomponents or groups. The groups of variables (genes) according to this individual are the following: $grps_1 = \{x_3, x_6, x_8\}$, $grps_2 = \{x_1, x_2, x_7\}$, $grps_3 = \{x_0, x_4\}$, and $grps_4 = \{x_5, x_9\}$. Note that the number V of variables in each subcomponent is variable and can be between 1 and D; in addition, the number of subcomponents m is between 1 and D.

3, 6, 8	1, 2, 7	0, 4	5, 9

Figure 1. Group-based chromosome, where each gene represents a subcomponent (set of variables).

3.2. Decomposition Evaluation

Each individual is evaluated to determine its fitness and to discover which one is the best within the population.

Sayed et al. [3] proposed a decomposition evaluation inspired by the definitions of problem separability [38] and partial separability [39]. The definition of problem separability states that a fully separable problem that has D variables can be written in the form of a linear combination of subproblems of the decision variables, where the evaluation of the complete problem, $F(\mathbf{x})$, is the same as the aggregation of the evaluation of the subproblems, $f(x_i)$, which means $F(\mathbf{x}) = \sum_{i=1}^{D} f(x_i)$. Additionally, a partially separable

problem is defined as one which has D variables and that can be decomposed into m subproblems, where the summation of all subproblems equals the solution of the complete problem $F(x)$ such that $F(\mathbf{x}) = \sum_{k=1}^{m} f_k(x_v)$, $v = [1 + V \times (k-1), V \times k]$, where m is the number of subproblems and V is the number of variables in the k-th subproblem. Sayed et al. proposed to measure each decomposition of variables by minimizing the absolute difference between the full evaluation and the sum of each evaluated subgroup.

Algorithm 2 shows the decomposition evaluation procedure. First (in Line 1), we obtain $fit_{all_{c_1}}$ and $fit_{all_{c_2}}$ through the evaluations of Equations (3) and (4), where all the variables take the constant value c_1 and $c2$, respectively. After that, both evaluations are added and multiplied by the number of subgroups in the problem (m) to obtain $fit_{all_{c_1c_2}}$, and then we initialize $fit_{grps_{c_1c_2}}$ as 0 (Lines 2–3). Afterwards, we start a loop from $k = 1$ to the number of subgroups m in the individual (Lines 4–10), Within this loop, we create two arrangements of D variables in which each variable belonging to group k takes the value c_1, while the remaining variables take the value c_2 to evaluate this arrangement and obtain $fit_{grps_{k,c_1}}$. On the other hand, to obtain $fit_{grps_{k,c_2}}$, the variables in the k-th group take the value of c_2, and the remaining ones take the value of c_1 (Lines 5–8) according to Equations (5) and (6). After that, we calculate $fit_{grps_{k,c_1c_2}}$ as the sum of the previously calculated $fit_{grps_{k,c_1}}$ and $fit_{grps_{k,c_2}}$ (Line 9). Thus, to end the loop, we update $fit_{grps_{c_1c_2}}$ as the sum of $fit_{grps_{c_1c_2}}$ and $fit_{grps_{k,c_1c_2}}$. Finally, we obtain the evaluation of the decomposition by calculating the absolute difference $grps_{diff}$ shown in Line 11.

Algorithm 2: Decomposition evaluation.

Input: m, and $c_1 > 0$, $c_2 > 0$ random numbers
Output: $grps_{diff}$

1 Evaluate $fit_{all_{c_1}}$ and $fit_{all_{c_2}}$ according to Equations (3) and (4)
2 Calculate $fit_{all_{c_1c_2}} = m \times [fit_{all_{c_1}} + fit_{all_{c_2}}]$
3 $fit_{grps_{c_1c_2}} = 0$
4 **for** $k = 1$ *to* m **do**
5 Create arrangement $\mathbf{x}_{k,c_1}$ according to Equation (5)
6 Calculate $fit_{grps_{k,c_1}} = Obj(\mathbf{x}_{k,c_1}) + cvs(\mathbf{x}_{k,c_1})$
7 Create arrangement $\mathbf{x}_{k,c_2}$ according to Equation (6)
8 Calculate $fit_{grps_{k,c_2}} = Obj(\mathbf{x}_{k,c_2}) + cvs(\mathbf{x}_{k,c_2})$
9 Calculate $fit_{grps_{k,c_1c_2}} = fit_{grps_{k,c_1}} + fit_{grps_{k,c_2}}$
10 Update $fit_{grps_{c_1c_2}} = fit_{grps_{c_1c_2}} + fit_{grps_{k,c_1c_2}}$
11 Calculate $grps_{diff} = |fit_{all_{c_1c_2}} - fit_{grps_{c_1c_2}}|$

$$fit_{all_{c_1}} = Obj(\mathbf{x}) + cvs(\mathbf{x}),\ x_i = c_1,\ \forall i \in [1, D] \tag{3}$$

$$fit_{all_{c_2}} = Obj(\mathbf{x}) + cvs(\mathbf{x}),\ x_i = c_2,\ \forall i \in [1, D] \tag{4}$$

$$\mathbf{x}_{k,c_1} = \begin{cases} c_1 & \forall x_i \in grps_k \\ c_2 & \text{otherwise} \end{cases} \tag{5}$$

$$\mathbf{x}_{k,c_2} = \begin{cases} c_2 & \forall x_i \in grps_k \\ c_1 & \text{otherwise} \end{cases} \tag{6}$$

To clarify the decomposition evaluation procedure, we present an example below. Let $Obj(\mathbf{x}) + cvs(\mathbf{x}) = f(\mathbf{x}) = x_1x_2 + x_3x_4$ be the problem to decompose, and according to the arrangement decomposition given by $grps_1 = \{x_1\}$, $grps_2 = \{x_2, x_4\}$, and $grps_3 = \{x_3\}$, we have $m = 3$. Suppose $c_1 = 1$ and $c_2 = 2$. In the first step, we calculate $fit_{all_{c_1}}$ and $fit_{all_{c_2}}$. According to Equations (3) and (4),

$$fit_{all_{c_1}} = f(x_i = c_1) = 1 * 1 + 1 * 1 = 2$$

$$fit_{all_{c_2}} = f(x_i = c_2) = 2 * 2 + 2 * 2 = 8$$

Then, continuing with step 2,

$$fit_{all_{c_1c_2}} = m \times [fit_{all_{c_1}} + fit_{all_{c_2}}] = 3 \times [2 + 8] = 30$$

In step 3, we initialize $fit_{grps_{c_1c_2}} = 0$. Then, we start the for loop. At this point in the process, we have to create arrangement $\mathbf{x}_{k,c_1}$ according to Equation (5) in step 5 and evaluate it in step 6. Similarly, the process is performed for c_2 in steps 7 and 8. To calculate $fit_{grps_{k,c_1}}$ the variables of the k-th group will be evaluated in c_1, and the rest in c_2; for $k = 1$, the group is $grps_1 = \{x_1\}$, so $x_1 = 1$ and $x_2, x_3, x_4 = 2$. A similar calculation is performed with $fit_{grps_{k,c_2}}$, but evaluating the variables of the k-th group in c_2 and the rest in c_1. Therefore, following the steps into the loop,

For $k = 1$, $grps_1 = \{x_1\}$:

$$fit_{grps_{k,c_1}} = f_{grps_{k,c_1}} = 1 * 2 + 2 * 2 = 6$$

$$fit_{grps_{k,c_2}} = f_{grps_{k,c_2}} = 2 * 1 + 1 * 1 = 3$$

$$fit_{grps_{k,c_1c_2}} = fit_{grps_{k,c_1}} + fit_{grps_{k,c_2}} = 6 + 3 = 9$$

$$fit_{grps_{c_1c_2}} = fit_{grps_{c_1c_2}} + fit_{grps_{k,c_1c_2}} = 0 + 9 = 9$$

For $k = 2$, $grps_2 = \{x_2, x_4\}$:

$$fit_{grps_{k,c_1}} = f_{grps_{k,c_1}} = 2 * 1 + 2 * 1 = 4$$

$$fit_{grps_{k,c_2}} = f_{grps_{k,c_2}} = 1 * 2 + 1 * 2 = 4$$

$$fit_{grps_{k,c_1c_2}} = fit_{grps_{k,c_1}} + fit_{grps_{k,c_2}} = 4 + 4 = 8$$

$$fit_{grps_{c_1c_2}} = fit_{grps_{c_1c_2}} + fit_{grps_{k,c_1c_2}} = 9 + 8 = 17$$

For $k = 3$, $grps_3 = \{x_3\}$:

$$fit_{grps_{k,c_1}} = f_{grps_{k,c_1}} = 2 * 2 + 1 * 2 = 6$$

$$fit_{grps_{k,c_2}} = f_{grps_{k,c_2}} = 1 * 1 + 2 * 1 = 3$$

$$fit_{grps_{k,c_1c_2}} = fit_{grps_{k,c_1}} + fit_{grps_{k,c_2}} = 6 + 3 = 9$$

$$fit_{grps_{c_1c_2}} = fit_{grps_{c_1c_2}} + fit_{grps_{k,c_1c_2}} = 17 + 9 = 26$$

Finally,

$$grps_{diff} = |fit_{all_{c_1c_2}} - fit_{grps_{c_1c_2}}| = |30 - 26| = 4$$

The purpose of the problem decomposition is to create the best decomposition; that is, to create independent subcomponents, as well as to minimize the difference $grps_{diff}$. Therefore, we have adopted a similar evaluation to the one proposed by Aguilar-Justo et al. [7], which is defined next: to maximize the number of subproblems, the $grps_{diff}$ is updated as follows: (1) if the number of subgroups is one, then the $grps_{diff}$ takes an extreme greater value; (2) if the $grps_{diff}$ is zero, this means that the decomposition is perfect, and it is rewarded by subtracting the number of subgroups (m) of the individual; (3) in another case, the $grps_{diff}$ value does not change. Since the use of the previous evaluation function

benefits a decomposition with a high number of subcomponents, in some cases, a complete decomposition (as many groups as numbers of variables) would be presented as optimal when in reality it is not the case. For this reason, a modification in the evaluation function is necessary (avoiding the benefit to a high number of groups). Therefore, the evaluation function has been modified in our algorithm. This change is summarized in Equation (7).

$$grps_{diff} = \begin{cases} infinite & \text{if } m = 1 \\ grps_{diff} & \text{otherwise} \end{cases} \tag{7}$$

3.3. Population Initialization

The initial population in most GGAs is generally generated by obtaining random partitions of the elements to group. In our GGA, to create a new chromosome, a random number between 1 and the dimension of the problem (D) is generated—i.e., $m \in [1, D]$—which represents the number of subcomponents m, and then each variable is randomly assigned to one of these groups. First, we ensure that each group contains at least one variable, and this is done by shuffling the variables and assigning the first m of them to each group. After that, the remaining variables are randomly assigned to one of the created groups. This is done because in the genetic algorithm DVIIC [7], a random number is chosen to determine the number of groups, and each variable is randomly assigned to a group (under the integer-based representation).

3.4. Grouping Crossover Operator

After choosing the individuals that are subject to the crossover operator, each pair of these individuals, called parents, will create two new individuals (offspring) through a mating strategy. There are several crossover operators for GGAs; however, for comparison purposes, we have chosen the two-point crossover operator that is analogous to the one used in the genetic algorithm DVIIC [7]. This operator works as follows: two crossing points (a and b) between 1 and the number of genes in the individual minus one ($m - 1$) are selected randomly to define the crossing section of both parents (P_1 and P_2). In this way, the first child (C_1) is generated with a copy of P_1, injecting and replacing the groups between the crossing points (a and b) of P_2. Next, the groups copied from P_1 with duplicated items are identified, removing the groups and releasing the remaining variables (missing variables), among which are also those elements that were lost when eliminating the groups from the crossing section and were not in the inserted groups. It is important to note that the injected groups remain intact. Finally, the missing variables are re-inserted into a random number of new groups (between 1 and the number of missing variables) to form the complete individual. The second child (C_2) is generated with the same process but changing the role of the parents.

In Figure 2, we can see an example of crossover for two individuals with 10 variables. The crossing points a and b are marked in step (1); then, in step (2), the section between a and b of parent P_1 is inserted and replaced in the other parent (P_2) and vice versa. In step (3), we have the free variables that result from the groups eliminated for having repeated variables, such as, in the first child, the free element 8 that was in the group with 3 and 6, which were repeated elements, and the elements 2 and 4 that were lost variables (elements). Finally, in step (4), we have the offspring with the free elements re-inserted.

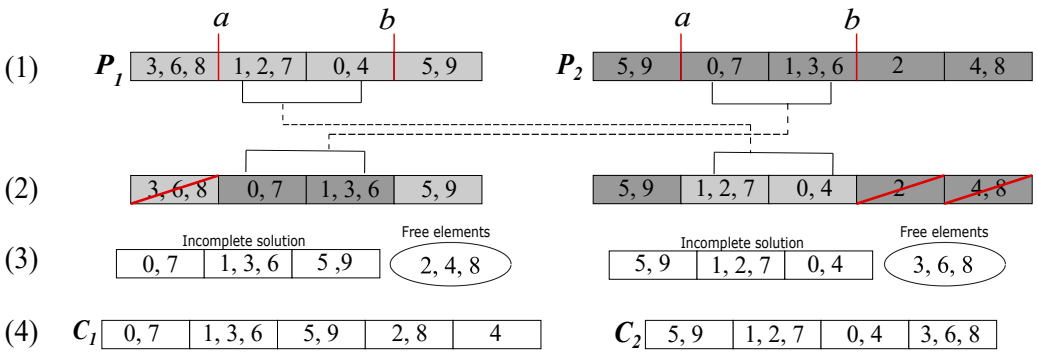


Figure 2. Two-point crossover operator.

3.5. Grouping Mutation Operator

The mutation operator used in the genetic algorithm DVIIC [7] is called uniform mutation, in which once an individual is selected to mutate, one of its genes is randomly selected and is changed from the group to which it belongs. Therefore, to take a similar operator, we have chosen to implement the group-oriented elimination mutation operator for GGAs. This operator works by eliminating a random group of the individual. Later, the deleted elements are re-inserted by adding a random number of groups between 1 and the number of free variables, with the variables randomly assigned to them (similar to how an individual is created). Figure 3 shows an example of the elimination operator. In step (1), the group marked in gray is the eliminated one; then, their elements pass to the free group of elements shown in step (2). Finally, the elements are re-inserted in step (3) with the aforementioned strategy.

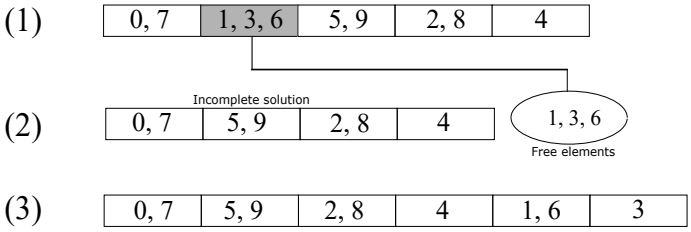


Figure 3. Elimination operator.

3.6. Selection and Replacement Strategies

In a Genetic Algorithm, we have to select the members of the population that will be candidates for crossover and mutation. A selection scheme decides which individuals are allowed to pass on their genes to the next generation, either through cloning, crossover, or mutation. Generally, selection schemes from the literature can be classified into three classes: proportional selection, tournament selection, and ranking selection. Usually, the selection is according to the relative fitness using the best or random individuals [40,41].

Several strategies have been proposed for the parent selection (individuals for crossover). In our GGA, we use a selection scheme similar to that included in the genetic algorithm DVIIC [7], and we carry out a shuffling of the population; for each pair of parents, a random number between 0 and 1 is created. This number determines if the pair of individuals is subject to crossover. That is, the crossover of both individuals is applied when the number is less than or equal to p_c.

In the same way, the selection of individuals to mutate has been studied, and there are various selection techniques for mutation. In this case, the selection method for mutation is similar to the selection method of the genetic algorithm DVIIC [7]. Given a mutation probability p_m, for each individual in the population, a random number between 0 and 1 is generated, and when this number is less or equal than p_m, the individual will be mutated.

In addition to the selection scheme, there must also be a criterion under which the population will be replaced in each generation. Generally, the replacement strategies can

be split into three classes: age-based, fitness-based, and random-based (deleting the oldest, worst, or random individuals, respectively) [42]. Similar to the strategy of the genetic algorithm DVIIC [7], in our GGA, after crossover, the offspring replace the parents, and after the mutation, the mutated individuals replace the original ones. Elitism is adopted to always maintain the best solution of the population, replacing the worst individual of the new population.

4. Experiments and Results

In order to study the benefits of using a group-based against an integer encoding in a genetic algorithm, we compared our proposal with the decomposition strategy proposed by Aguilar-Justo et al. [31]. Therefore, we chose the same set of test functions the authors used. It is the first set for Large-Scale Constrained Optimization Problems and it was proposed by Sayed et al. in 2015 [3]. This test set has different separability complexity degrees, which are described in Table 1. It can be tested over three numbers of variables (100, 500, and 1000). These 18 functions were created by combining 6 objective functions with 1, 2, or 3 constraints. The 6 objective functions are based on 2 problems in the literature that have been used, for example, in the CEC 2008 benchmark problems [4], which are the Rosenbrock's function, which is multimodal and nonseparable, and the Sphere function, which is unimodal and separable. In addition, in Table 2, we can see the components of these 18 test functions; that is, the objective function and the constraints that make up each function. The details of the mathematical expression of each function can be consulted in the work of Sayed et al. [3].

Table 1. Characteristics of the objective functions and constraints.

	Description
Obj_1	Completely separable
Obj_2	Partially nonseparable
Obj_3	Partially nonseparable
Obj_4	Partially nonseparable and overlapping variables
Obj_5	Spliced nonseparable and overlapping variables
Obj_6	Spliced nonseparable and overlapping variables
g_1	Separable groups of 5 variables
g_2	Nonseparable groups of 3 variables
g_3	Spliced nonseparable pairs

We have compared the results of our GGA against the Dynamical Variable Interaction Identification Technique for Constrained Problems (DVIIC), in which Aguilar et al. [7] proposed a genetic algorithm for the decomposition of the 18 test functions. We computed 25 independent runs per each benchmark function, in 3 different numbers of variables (100, 500, and 1000). The parameters of our algorithm were set similarly as in the DVIIC work, to compare under equal conditions and perform the same number of function evaluations. These are as follows:

- Population size of 100 individuals;
- Crossover probability $p_c = 0.9$;
- Mutation probability $p_m = 0.1$;
- 10,000 function evaluations—i.e., 100 generations.

Such a configuration implies that the same number of evaluations is carried out by having 100 individuals in each generation for 100 generations, which is equal to 10,000 evaluations. These experiments were conducted on an Intel(R) Core(TM) i5 CPU with 2.50 GHz, Python 3.4, and Microsoft Windows 10.

In the following tables, we show the results of the execution of our proposed GGA and the genetic algorithm DVIIC. Both were executed for each of the 18 functions, 25 times in each dimension. Furthermore, each of the tables shows the results of the Wilcoxon Rank Sum test for each of the functions (column W). A checkmark (✓) means that there are significant differences in favor of the GGA; in addition, an equality symbol (=) represents there are no significant differences between both algorithms.

Table 2. Components of the 18 test functions.

Function	Objective	g_1	g_2	g_3
F_1		×		
F_2	Obj_1	×	×	
F_3		×	×	×
F_4		×		
F_5	Obj_2	×	×	
F_6		×	×	×
F_7		×		
F_8	Obj_3	×	×	
F_9		×	×	×
F_{10}		×		
F_{11}	Obj_4	×	×	
F_{12}		×	×	×
F_{13}		×		
F_{14}	Obj_5	×	×	
F_{15}		×	×	×
F_{16}		×		
F_{17}	Obj_6	×	×	
F_{18}		×	×	×

First, Table 3 contains the results according to the evaluation of the best individual for the 25 runs in the 18 functions under 100 variables. The best, median, and the standard deviation registered of the evaluation function ($grps_{diff}$) value are shown. We can observe that the GGA improves the decomposition evaluation function value in all of the cases compared to DVIIC. As we can see, unlike DVIIC, the GGA reaches the value of 0 in the best result in most cases. Furthermore, the median and standard deviation values obtained by the GGA are smaller in all cases. Such values equal to zero indicate that our algorithm found the best value for the evaluation function ($grps_{diff} = 0$) in the 25 runs for functions 1 to 12. Finally, the Wilcoxon Rank Sum Test reveals that there are significant differences in favor of the GGA in all cases.

Second, Table 4 shows the results of our algorithm to solve the same 18 functions, now with 500 variables. The GGA obtained the smallest values in most cases for the best, median, and standard deviation when compared against DVIIC. In a similar way as in Table 3, the standard deviation and median values equal to zero indicate that our algorithm found the best value for the evaluation function ($grps_{diff} = 0$) in the 25 runs for functions 1 to 12. Moreover, in the other test functions, the best, median, and standard deviation values are smaller when compared to DVIIC. On the other hand, the Wilcoxon Rank Sum test shows that there are no significant differences between the two algorithms in function 1 and shows significant differences in favor of the GGA in the remaining 17 functions. According to this test, in functions 2 to 18, the Wilcoxon Rank Sum test rejects the hypothesis that the DVIIC approach is as effective as the proposed GGA approach, and F_1 is trivial to solve by the two algorithms.

Table 3. Statistical results in dimension 100. Best results shown in boldface.

	GGA			DVIIC			
D = 100	**Best**	**Median**	**Std**	**Best**	**Median**	**Std**	**W**
F_1	$\mathbf{0.00 \times 10^{0}}$	$\mathbf{0.00 \times 10^{0}}$	$\mathbf{0.00 \times 10^{0}}$	$\mathbf{0.00 \times 10^{0}}$	2.18×10^{-11}	2.96×10^{-11}	✓
F_2	$\mathbf{0.00 \times 10^{0}}$	$\mathbf{0.00 \times 10^{0}}$	$\mathbf{0.00 \times 10^{0}}$	$\mathbf{0.00 \times 10^{0}}$	2.35×10^{4}	1.20×10^{4}	✓
F_3	$\mathbf{0.00 \times 10^{0}}$	$\mathbf{0.00 \times 10^{0}}$	$\mathbf{0.00 \times 10^{0}}$	1.38×10^{5}	1.38×10^{5}	6.77×10^{4}	✓
F_4	$\mathbf{0.00 \times 10^{0}}$	$\mathbf{0.00 \times 10^{0}}$	$\mathbf{0.00 \times 10^{0}}$	4.29×10^{4}	8.57×10^{4}	1.69×10^{4}	✓
F_5	$\mathbf{0.00 \times 10^{0}}$	$\mathbf{0.00 \times 10^{0}}$	$\mathbf{0.00 \times 10^{0}}$	1.27×10^{4}	1.78×10^{4}	2.48×10^{3}	✓
F_6	$\mathbf{0.00 \times 10^{0}}$	$\mathbf{0.00 \times 10^{0}}$	$\mathbf{0.00 \times 10^{0}}$	8.97×10^{5}	1.44×10^{6}	2.36×10^{5}	✓
F_7	$\mathbf{0.00 \times 10^{0}}$	$\mathbf{0.00 \times 10^{0}}$	$\mathbf{0.00 \times 10^{0}}$	2.51×10^{5}	1.01×10^{6}	2.62×10^{5}	✓
F_8	$\mathbf{0.00 \times 10^{0}}$	$\mathbf{0.00 \times 10^{0}}$	$\mathbf{0.00 \times 10^{0}}$	6.70×10^{5}	8.38×10^{5}	9.84×10^{4}	✓
F_9	$\mathbf{0.00 \times 10^{0}}$	$\mathbf{0.00 \times 10^{0}}$	$\mathbf{0.00 \times 10^{0}}$	2.13×10^{3}	3.20×10^{3}	3.58×10^{2}	✓
F_{10}	$\mathbf{0.00 \times 10^{0}}$	$\mathbf{0.00 \times 10^{0}}$	$\mathbf{0.00 \times 10^{0}}$	5.36×10^{5}	1.07×10^{6}	2.05×10^{5}	✓
F_{11}	$\mathbf{0.00 \times 10^{0}}$	$\mathbf{0.00 \times 10^{0}}$	$\mathbf{0.00 \times 10^{0}}$	4.12×10^{2}	5.84×10^{2}	9.04×10^{1}	✓
F_{12}	$\mathbf{0.00 \times 10^{0}}$	$\mathbf{0.00 \times 10^{0}}$	$\mathbf{0.00 \times 10^{0}}$	9.28×10^{3}	1.76×10^{4}	3.20×10^{3}	✓
F_{13}	$\mathbf{0.00 \times 10^{0}}$	$\mathbf{0.00 \times 10^{0}}$	$\mathbf{2.31 \times 10^{4}}$	6.31×10^{5}	1.09×10^{6}	1.08×10^{5}	✓
F_{14}	$\mathbf{0.00 \times 10^{0}}$	$\mathbf{0.00 \times 10^{0}}$	$\mathbf{5.35 \times 10^{2}}$	1.55×10^{7}	1.93×10^{7}	1.72×10^{6}	✓
F_{15}	$\mathbf{4.66 \times 10^{-10}}$	$\mathbf{4.66 \times 10^{-10}}$	$\mathbf{2.60 \times 10^{2}}$	5.42×10^{6}	8.72×10^{6}	9.33×10^{5}	✓
F_{16}	$\mathbf{1.79 \times 10^{4}}$	$\mathbf{5.38 \times 10^{4}}$	$\mathbf{1.72 \times 10^{4}}$	4.93×10^{5}	6.81×10^{5}	7.26×10^{4}	✓
F_{17}	$\mathbf{1.07 \times 10^{5}}$	$\mathbf{4.28 \times 10^{5}}$	9.52×10^{4}	4.83×10^{5}	6.99×10^{5}	$\mathbf{8.07 \times 10^{4}}$	✓
F_{18}	$\mathbf{1.63 \times 10^{5}}$	$\mathbf{4.88 \times 10^{5}}$	$\mathbf{1.40 \times 10^{5}}$	1.80×10^{6}	2.62×10^{6}	2.94×10^{5}	✓

Table 4. Statistical results in dimension 500. Best results shown in boldface.

	GGA			DVIIC			
D = 500	**Best**	**Median**	**Std**	**Best**	**Median**	**Std**	**W**
F_1	$\mathbf{0.00 \times 10^{0}}$	$\mathbf{0.00 \times 10^{0}}$	$\mathbf{0.00 \times 10^{0}}$	$\mathbf{0.00 \times 10^{0}}$	$\mathbf{0.00 \times 10^{0}}$	$\mathbf{0.00 \times 10^{0}}$	=
F_2	$\mathbf{0.00 \times 10^{0}}$	$\mathbf{0.00 \times 10^{0}}$	$\mathbf{0.00 \times 10^{0}}$	4.66×10^{-10}	4.43×10^{4}	1.16×10^{4}	✓
F_3	$\mathbf{0.00 \times 10^{0}}$	$\mathbf{0.00 \times 10^{0}}$	$\mathbf{0.00 \times 10^{0}}$	3.59×10^{4}	7.19×10^{4}	1.23×10^{4}	✓
F_4	$\mathbf{0.00 \times 10^{0}}$	$\mathbf{0.00 \times 10^{0}}$	$\mathbf{0.00 \times 10^{0}}$	1.17×10^{6}	1.24×10^{6}	2.98×10^{4}	✓
F_5	$\mathbf{0.00 \times 10^{0}}$	$\mathbf{0.00 \times 10^{0}}$	$\mathbf{0.00 \times 10^{0}}$	7.20×10^{6}	8.73×10^{6}	5.68×10^{5}	✓
F_6	$\mathbf{0.00 \times 10^{0}}$	$\mathbf{0.00 \times 10^{0}}$	$\mathbf{0.00 \times 10^{0}}$	3.34×10^{6}	4.12×10^{6}	2.09×10^{5}	✓
F_7	$\mathbf{0.00 \times 10^{0}}$	$\mathbf{0.00 \times 10^{0}}$	$\mathbf{0.00 \times 10^{0}}$	5.19×10^{4}	1.13×10^{5}	1.94×10^{4}	✓
F_8	$\mathbf{0.00 \times 10^{0}}$	$\mathbf{0.00 \times 10^{0}}$	$\mathbf{0.00 \times 10^{0}}$	8.79×10^{5}	9.75×10^{5}	4.19×10^{4}	✓
F_9	$\mathbf{0.00 \times 10^{0}}$	$\mathbf{0.00 \times 10^{0}}$	$\mathbf{0.00 \times 10^{0}}$	5.58×10^{4}	1.67×10^{5}	3.46×10^{4}	✓
F_{10}	$\mathbf{0.00 \times 10^{0}}$	$\mathbf{0.00 \times 10^{0}}$	$\mathbf{0.00 \times 10^{0}}$	9.69×10^{6}	1.19×10^{7}	5.03×10^{5}	✓
F_{11}	$\mathbf{0.00 \times 10^{0}}$	$\mathbf{0.00 \times 10^{0}}$	$\mathbf{0.00 \times 10^{0}}$	1.95×10^{6}	2.58×10^{6}	1.80×10^{5}	✓
F_{12}	$\mathbf{0.00 \times 10^{0}}$	$\mathbf{0.00 \times 10^{0}}$	$\mathbf{0.00 \times 10^{0}}$	3.50×10^{6}	3.84×10^{6}	1.10×10^{5}	✓
F_{13}	$\mathbf{0.00 \times 10^{0}}$	$\mathbf{0.00 \times 10^{0}}$	$\mathbf{2.56 \times 10^{4}}$	9.61×10^{5}	1.26×10^{6}	7.07×10^{4}	✓
F_{14}	$\mathbf{0.00 \times 10^{0}}$	$\mathbf{0.00 \times 10^{0}}$	$\mathbf{6.86 \times 10^{4}}$	9.10×10^{5}	1.35×10^{6}	1.08×10^{5}	✓
F_{15}	$\mathbf{0.00 \times 10^{0}}$	$\mathbf{5.96 \times 10^{-8}}$	$\mathbf{1.27 \times 10^{5}}$	7.97×10^{6}	9.88×10^{6}	4.48×10^{5}	✓
F_{16}	$\mathbf{8.13 \times 10^{2}}$	$\mathbf{3.25 \times 10^{3}}$	$\mathbf{7.04 \times 10^{2}}$	1.38×10^{7}	1.69×10^{7}	7.77×10^{5}	✓
F_{17}	$\mathbf{7.96 \times 10^{4}}$	$\mathbf{1.59 \times 10^{5}}$	$\mathbf{1.75 \times 10^{4}}$	2.26×10^{7}	2.43×10^{7}	1.48×10^{7}	✓
F_{18}	$\mathbf{3.83 \times 10^{5}}$	$\mathbf{7.66 \times 10^{5}}$	$\mathbf{1.46 \times 10^{5}}$	2.78×10^{7}	3.63×10^{7}	2.05×10^{6}	✓

Finally, Table 5 contains the results for the 18 functions with both algorithms implemented on 1000 variables. In this experiment, we observed that, in a similar way to the experiment with 500 variables, our GGA obtained the smallest best, median, and standard deviation values in most cases. On the other hand, the behavior of the median and standard deviation values allows us to see that we obtained the zero value in the 25 independent runs for the first 12 functions. Moreover, the Wilcoxon Rank Sum test results show that there are no significant differences between the two algorithms in function 1 and indicate significant differences in favor of the GGA in the remaining 17 functions. In a similar way as in the results with 500 variables, the Wilcoxon Rank Sum test determines that F_1 is a trivial case and rejects the hypothesis that the DVIIC approach is as effective as the proposed GGA approach for the other 17 remaining functions.

Table 5. Statistical results in dimension 1000. Best results shown in boldface.

	GGA			DVIIC			
D = 1000	**Best**	**Median**	**Std**	**Best**	**Median**	**Std**	**W**
F_1	$\mathbf{0.00 \times 10^0}$	$\mathbf{0.00 \times 10^0}$	$\mathbf{0.00 \times 10^0}$	$\mathbf{0.00 \times 10^0}$	$\mathbf{0.00 \times 10^0}$	$\mathbf{0.00 \times 10^0}$	=
F_2	$\mathbf{0.00 \times 10^0}$	$\mathbf{0.00 \times 10^0}$	$\mathbf{0.00 \times 10^0}$	4.18×10^3	1.25×10^4	5.39×10^3	✓
F_3	$\mathbf{0.00 \times 10^0}$	$\mathbf{0.00 \times 10^0}$	$\mathbf{0.00 \times 10^0}$	2.42×10^2	2.90×10^2	1.98×10^1	✓
F_4	$\mathbf{0.00 \times 10^0}$	$\mathbf{0.00 \times 10^0}$	$\mathbf{0.00 \times 10^0}$	3.93×10^6	4.86×10^6	2.33×10^5	✓
F_5	$\mathbf{0.00 \times 10^0}$	$\mathbf{0.00 \times 10^0}$	$\mathbf{0.00 \times 10^0}$	1.33×10^6	1.73×10^6	1.19×10^5	✓
F_6	$\mathbf{0.00 \times 10^0}$	$\mathbf{0.00 \times 10^0}$	$\mathbf{0.00 \times 10^0}$	1.05×10^6	1.36×10^6	7.85×10^4	✓
F_7	$\mathbf{0.00 \times 10^0}$	$\mathbf{0.00 \times 10^0}$	$\mathbf{0.00 \times 10^0}$	1.10×10^6	1.61×10^6	1.28×10^5	✓
F_8	$\mathbf{0.00 \times 10^0}$	$\mathbf{0.00 \times 10^0}$	$\mathbf{0.00 \times 10^0}$	4.16×10^6	4.91×10^6	1.82×10^5	✓
F_9	$\mathbf{0.00 \times 10^0}$	$\mathbf{0.00 \times 10^0}$	$\mathbf{0.00 \times 10^0}$	4.56×10^6	4.92×10^6	1.09×10^5	✓
F_{10}	$\mathbf{0.00 \times 10^0}$	$\mathbf{0.00 \times 10^0}$	$\mathbf{0.00 \times 10^0}$	2.16×10^6	2.47×10^6	8.18×10^4	✓
F_{11}	$\mathbf{0.00 \times 10^0}$	$\mathbf{0.00 \times 10^0}$	$\mathbf{0.00 \times 10^0}$	1.99×10^5	2.13×10^5	4.33×10^3	✓
F_{12}	$\mathbf{0.00 \times 10^0}$	$\mathbf{0.00 \times 10^0}$	$\mathbf{0.00 \times 10^0}$	2.93×10^5	3.18×10^5	8.43×10^3	✓
F_{13}	$\mathbf{0.00 \times 10^0}$	$\mathbf{0.00 \times 10^0}$	$\mathbf{3.51 \times 10^4}$	3.36×10^7	3.98×10^7	1.49×10^6	✓
F_{14}	$\mathbf{0.00 \times 10^0}$	$\mathbf{0.00 \times 10^0}$	$\mathbf{5.49 \times 10^3}$	4.60×10^7	1.76×10^8	6.74×10^7	✓
F_{15}	$\mathbf{0.00 \times 10^0}$	$\mathbf{0.00 \times 10^0}$	$\mathbf{2.00 \times 10^5}$	1.02×10^5	3.49×10^7	1.55×10^7	✓
F_{16}	$\mathbf{4.07 \times 10^5}$	$\mathbf{8.15 \times 10^5}$	$\mathbf{1.27 \times 10^5}$	7.71×10^7	8.72×10^7	2.37×10^6	✓
F_{17}	$\mathbf{4.32 \times 10^4}$	$\mathbf{8.63 \times 10^4}$	$\mathbf{2.52 \times 10^4}$	7.71×10^6	1.15×10^7	1.05×10^6	✓
F_{18}	$\mathbf{4.20 \times 10^5}$	$\mathbf{8.41 \times 10^5}$	$\mathbf{1.20 \times 10^5}$	3.03×10^7	4.39×10^7	3.07×10^6	✓

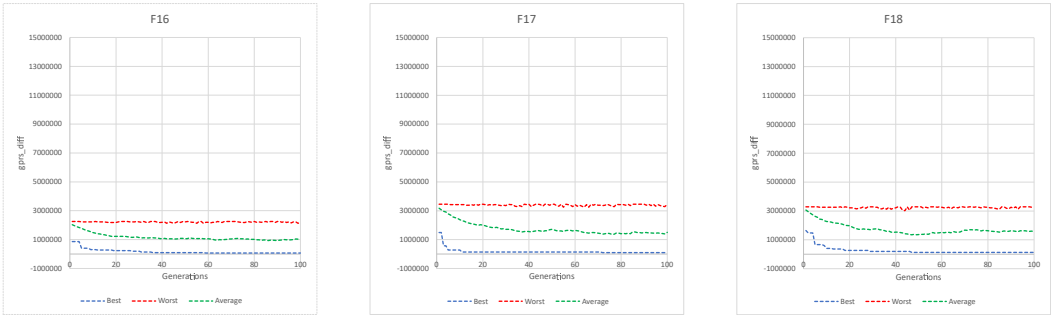
Given the previous tables, we observe that our algorithm presents better performance than DVIIC, obtaining better $grps_{diff}$ values in all cases (in comparison with the mentioned algorithm). An interesting behavior is observed in these experiments; it seems to be more difficult for our algorithm to find the minimum decomposition evaluation in the 18 test functions as the dimension decreases. Zero best, median, and standard deviation values of the 25 independent runs indicate a stable behavior of our algorithm in each execution of the first 12 functions of the benchmark (in the three experiments). However, these values increase with the complexity of the functions, and in the end, functions 16, 17, and 18 do not reach the minimum in any of the experiments.

Analyzing the Performance of the GGA

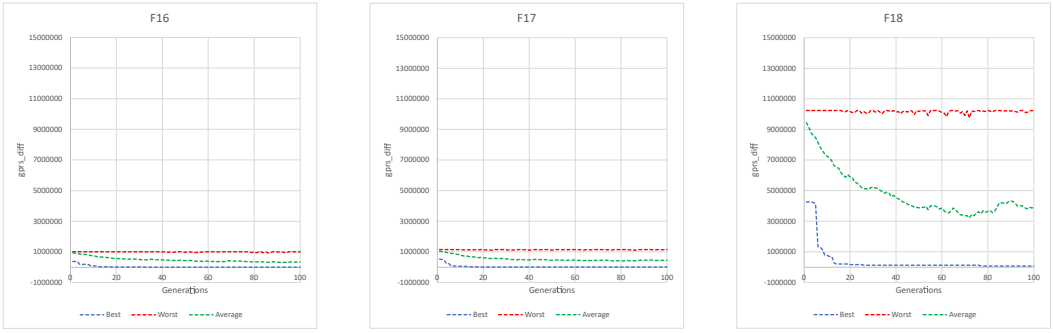
Due to the behavior observed in the previous experiments, a detailed study of the algorithm is necessary to improve it in future work. For this reason, we decided to make a brief study of the convergence of our algorithm.

In order to understand the on-line behavior of our algorithm, we carried out some plots of the GGA convergence for the most difficult functions of the benchmark. Figure 4

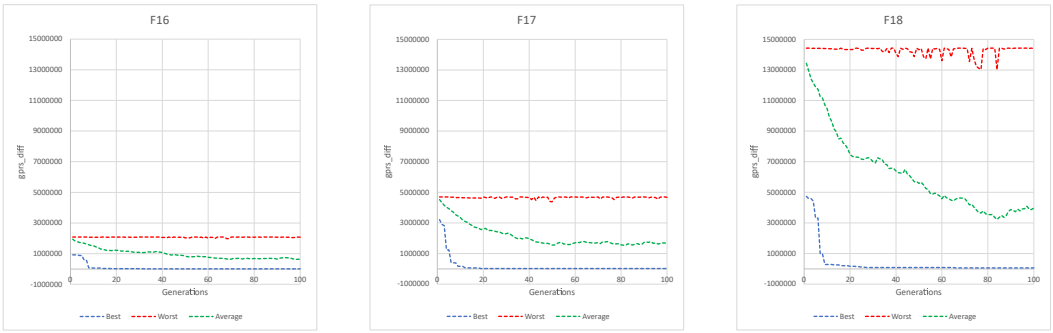
shows the convergence in functions F_{16}, F_{17}, and F_{18} through 100 generations for three dimension values.



(**a**) 100D-F_{16} (**b**) 100D-F_{17} (**c**) 100D-F_{18}



(**d**) 500D-F_{16} (**e**) 500D-F_{17} (**f**) 500D-F_{18}



(**g**) 1000D-F_{16} (**h**) 1000D-F_{17} (**i**) 1000D-F_{18}

Figure 4. Convergence plots of 100 generations for functions F_{16}, F_{17}, and F_{18} with 100, 500, and 1000 variables.

Three convergence behaviors are shown in each of the graphs within Figure 4. First, we shown the convergence of the worst individual in the population—that is, the individual with the highest decomposition evaluation value (red color). After that, we show the behavior across the 100 generations of the average decomposition evaluation of the 100 individuals in the population (green color). Finally, we show the convergence across the 100 generations of the best individual in the population in terms of their decomposition evaluation value (blue color).

Figure 4a–c shows the convergence in the experiment with 100 variables from one of the 25 GGA runs. We can observe similar behavior in the three functions, with decomposition evaluation values below 4.0×10^7 in all three cases (best, worst, and average). It is important to note that the behavior of the best individual presents an early convergence in the three functions and how the decomposition evaluation of the worst individuals remains stable over the generations.

Regarding the convergence of the functions with 500 variables (Figure 4d–f), we can observe that function F_{18} shows the highest values of the decomposition evaluation for the three values (best, worst, and average), unlike the other functions. Similar to those functions with 100 variables, we see that this value does not converge to zero in any of the cases, and the best individual has a quick convergence. We can also induce, according to the graphs, that several individuals of the population do not converge in the neighborhood of the best solution.

In the case of the functions evaluated with 1000 variables (Figure 4g–i), we see a fast convergence of the best individual in the population. As in the previous graphs, the value of the decomposition evaluation in the worst individual has a continuous behavior, without converging to zero during the 100 generations, and the best value has a quick convergence.

The above discussed best, worst, and average values' convergence behaviors in the functions with spliced nonseparable and overlapping variables suggest that the included strategies in the GGA do not appear to lead to better solutions. We can see from the plots in Figure 4 that not the entire population converges to the neighborhood of the best solution, due to the low selective pressure of the selection and replacement strategies. We also observe that the GGA produces good solutions in the early stages but leads to the premature convergence of the best individual. This behavior can be related to the crossover and mutation operators that promoted the creation of new groups, which does not seem to be a suitable strategy for nonseparable functions. All these observations indicate that, although our algorithm performs well, it can still be further improved by analyzing its components.

5. Conclusions and Future Work

In this paper, we have proposed a Grouping Genetic Algorithm (GGA) to deal with the decomposition of variables in Large-Scale Constrained Optimization Problems to create subproblems of the original problem and thus reduce the dimension. To evaluate the impact of the representation scheme on the performance of a genetic algorithm, our GGA was designed in a similar way to a state-of-the-art genetic algorithm that works with the decomposition of variables and which includes an integer-based representation. The main difference between the two algorithms was the genetic encoding. The experiments were carried out in a benchmark of 18 functions with different complexity characteristics, and these functions were tested in 100, 500, and 1000 dimensions.

The obtained results confirm that the use of a group-based genetic encoding allows our GGA to obtain good and robust decompositions on test functions with different features and separability complexity degrees, outperforming in all the benchmark functions the results obtained by a genetic algorithm with an integer-based encoding.

We are aware that there are still test functions with spliced nonseparable and overlapping variables that show a high degree of difficulty; for these functions, the included strategies in the GGA do not appear to lead to better solutions. However, the GGA presented in this work does not include the state-of-the-art grouping genetic operators.

Future work will consist of studying the parameters of the GGA as well as the effect of each of the methods used in the crossover and mutation operators to identify the best strategies that work together with the grouping encoding scheme and the features of the functions. Furthermore, it is necessary to implement an efficient reproduction technique with a balance in selective pressure and population diversity to avoid the premature convergence of the best individuals and increase the algorithm's performance.

The introduction of a new decomposition method opens up an interesting range of possibilities for future research. Currently, we are working on including our GGA in the decomposition step of two Cooperative Co-Evolution methods that include different strategies for the optimization and cooperation of the subcomponents, with the respective feasibility and computational complexity analysis.

Finally, although the set of test functions analyzed in this work is varied concerning the characteristics of the functions, we would explore the proposal in other Large-Scale Constrained Optimization benchmarks.

Author Contributions: Conceptualization, G.C.-A., M.Q.-C. and E.M.-M.; methodology, M.Q.-C.; software, G.C.-A.; validation, M.Q.-C. and E.M.-M.; formal analysis, M.Q.-C.; investigation, G.C.-A., M.Q.-C. and E.M.-M.; resources, G.C.-A.; writing—original draft preparation, G.C.-A.; writing—review and editing, G.C.-A., M.Q.-C. and E.M.-M.; visualization, G.C.-A. and M.Q.-C.; supervision, M.Q.-C.; project administration, M.Q.-C. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Conflicts of Interest: The authors declare no conflict of interest. The funders had no role in the design of the study; in the collection, analyses, or interpretation of data; in the writing of the manuscript, or in the decision to publish the results.

References

1. Deb, K. *Optimization for Engineering Design: Algorithms and Examples*; PHI Learning Pvt. Ltd.: New Delhi, India, 2012.
2. Smith, A.E.; Coit, D.W.; Baeck, T.; Fogel, D.; Michalewicz, Z. Penalty functions. In *Handbook of Evolutionary Computation*; CRC Press: Boca Raton, FL, USA, 1997.
3. Sayed, E.; Essam, D.; Sarker, R.; Elsayed, S. Decomposition-based evolutionary algorithm for large-scale constrained problems. *Inf. Sci.* **2015**, *316*, 457–486. [CrossRef]
4. Tang, K.; Yáo, X.; Suganthan, P.N.; MacNish, C.; Chen, Y.P.; Chen, C.M.; Yang, Z. *Benchmark Functions for the CEC'2008 Special Session and Competition on Large Scale Global Optimization*; Nature Inspired Computation and Applications Laboratory, USTC: Hefei, China, 2007; pp. 1–18.
5. Potter, M.A.; De Jong, K.A. A cooperative coevolutionary approach to function optimization. In *International Conference on Parallel Problem Solving from Nature*; Springer: Berlin/Heidelberg, Germany, 1994; pp. 249–257.
6. Ma, X.; Li, X.; Zhang, Q.; Tang, K.; Liang, Z.; Xie, W.; Zhu, Z. A survey on cooperative co-evolutionary algorithms. *IEEE Trans. Evol. Comput.* **2018**, *23*, 421–441. [CrossRef]
7. Aguilar-Justo, A.E.; Mezura-Montes, E.; Elsayed, S.M.; Sarker, R.A. Decomposition of large-scale constrained problems using a genetic-based search. In Proceedings of the 2016 IEEE International Autumn Meeting on Power, Electronics and Computing (ROPEC), Ixtapa, Mexico, 9–11 November 2016; IEEE: Piscataway, NJ, USA, 2016; pp. 1–6.
8. Ramos-Figueroa, O.; Quiroz-Castellanos, M.; Mezura-Montes, E.; Schütze, O. Metaheuristics to solve grouping problems: A review and a case study. *Swarm Evol. Comput.* **2020**, *53*, 100643. [CrossRef]
9. Liu, Y.; Yao, X.; Zhao, Q.; Higuchi, T. Scaling up fast evolutionary programming with cooperative coevolution. In Proceedings of the 2001 Congress on Evolutionary Computation (IEEE Cat. No. 01TH8546), San Francisco, CA, USA, 7–11 July 2001; IEEE: Piscataway, NJ, USA, 2001; Volume 2, pp. 1101–1108.
10. Van den Bergh, F.; Engelbrecht, A.P. A cooperative approach to particle swarm optimization. *IEEE Trans. Evol. Comput.* **2004**, *8*, 225–239. [CrossRef]
11. Yang, Z.; Tang, K.; Yao, X. Differential evolution for high-dimensional function optimization. In Proceedings of the 2007 IEEE Congress on Evolutionary Computation, Singapore, 25–28 September 2007; IEEE: Piscataway, NJ, USA, 2007; pp. 3523–3530.
12. Yang, Z.; Tang, K.; Yao, X. Large-scale evolutionary optimization using cooperative coevolution. *Inf. Sci.* **2008**, *178*, 2985–2999. [CrossRef]
13. Omidvar, M.N.; Li, X.; Yang, Z.; Yao, X. Cooperative co-evolution for large-scale optimization through more frequent random grouping. In Proceedings of the IEEE Congress on Evolutionary Computation, Barcelona, Spain, 18–23 July 2010; IEEE: Piscataway, NJ, USA, 2010; pp. 1–8.
14. Omidvar, M.N.; Li, X.; Yao, X. Cooperative co-evolution with delta grouping for large-scale non-separable function optimization. In Proceedings of the IEEE Congress on Evolutionary Computation, Barcelona, Spain, 18–23 July 2010; IEEE: Piscataway, NJ, USA, 2010; pp. 1–8.
15. Xu, B.; Zhang, Y.; Gong, D.; Guo, Y.; Rong, M. Environment sensitivity-based cooperative co-evolutionary algorithms for dynamic multi-objective optimization. *IEEE/ACM Trans. Comput. Biol. Bioinform.* **2017**, *15*, 1877–1890. [CrossRef] [PubMed]
16. Omidvar, M.N.; Li, X.; Mei, Y.; Yao, X. Cooperative co-evolution with differential grouping for large-scale optimization. *IEEE Trans. Evol. Comput.* **2013**, *18*, 378–393. [CrossRef]
17. Omidvar, M.N.; Yang, M.; Mei, Y.; Li, X.; Yao, X. DG2: A faster and more accurate differential grouping for large-scale black-box optimization. *IEEE Trans. Evol. Comput.* **2017**, *21*, 929–942. [CrossRef]
18. Sun, Y.; Kirley, M.; Halgamuge, S.K. A recursive decomposition method for large-scale continuous optimization. *IEEE Trans. Evol. Comput.* **2017**, *22*, 647–661. [CrossRef]

19. Sun, Y.; Omidvar, M.N.; Kirley, M.; Li, X. Adaptive threshold parameter estimation with recursive differential grouping for problem decomposition. In Proceedings of the Genetic and Evolutionary Computation Conference, Ser. GECCO '18, Kyoto, Japan, 15–19 July 2018; ACM: New York, NY, USA, 2018; pp. 889–896.
20. Sun, Y.; Li, X.; Ernst, A.; Omidvar, M.N. Decomposition for large-scale optimization problems with overlapping components. In Proceedings of the 2019 IEEE Congress on Evolutionary Computation (CEC), Wellington, New Zealand, 10–13 June 2019; pp. 326–333.
21. Yang, M.; Zhou, A.; Li, C.; Yao, X. An Efficient Recursive Differential Grouping for Large-Scale Continuous Problems. *IEEE Trans. Evol. Comput.* **2020**, *25*, 159–171. [CrossRef]
22. Sopov, E. *Large-Scale Global Optimization Using a Binary Genetic Algorithm with EDA-Based Decomposition*; Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics); Springer: Cham, Switzerland, 2016; Volume 9712; pp. 619–626.
23. Valdez, S.I.; Hernández, A.; Botello, S. A Boltzmann based estimation of distribution algorithm. *Inf. Sci.* **2013**, *236*, 126–137. [CrossRef]
24. Mühlenbein, H.; Paaß, G. From recombination of genes to the estimation of distributions I. Binary parameters. In *International Conference on Parallel Problem Solving from Nature*; Springer: Berlin/Heidelberg, Germany, 1996; pp. 178–187.
25. Sayed, E.; Essam, D.; Sarker, R. Dependency identification technique for large-scale optimization problems. In Proceedings of the 2012 IEEE Congress on Evolutionary Computation, Philadelphia, PA, USA, 7–11 July 2012; IEEE: Piscataway, NJ, USA, 2012; pp. 1–8.
26. Schaffer, J.D.; Morishima, A. An adaptive crossover distribution mechanism for genetic algorithms. In *Genetic Algorithms and Their Applications: Proceedings of the Second International Conference on Genetic Algorithms*; Lawrence Erlbaum Associates, Inc.: Hillsdale, NJ, USA, 1987; pp. 36–40.
27. Goh, C.K.; Tan, K.C. A competitive-cooperative coevolutionary paradigm for dynamic multiobjective optimization. *IEEE Trans. Evol. Comput.* **2008**, *13*, 103–127.
28. Goh, C.K.; Tan, K.C.; Liu, D.S.; Chiam, S.C. A competitive and cooperative co-evolutionary approach to multi-objective particle swarm optimization algorithm design. *Eur. J. Oper. Res.* **2010**, *202*, 42–54. [CrossRef]
29. Strasser, S.; Sheppard, J.; Fortier, N.; Goodman, R. Factored evolutionary algorithms. *IEEE Trans. Evol. Comput.* **2016**, *21*, 281–293. [CrossRef]
30. Guan, X.; Zhang, X.; Wei, J.; Hwang, I.; Zhu, Y.; Cai, K. A strategic conflict avoidance approach based on cooperative coevolutionary with the dynamic grouping strategy. *Int. J. Syst. Sci.* **2016**, *47*, 1995–2008. [CrossRef]
31. Aguilar-Justo, A.E.; Mezura-Montes, E. Towards an improvement of variable interaction identification for large-scale constrained problems. In Proceedings of the IEEE Congress on Evolutionary Computation, Vancouver, BC, Canada, 24–29 July 2016; IEEE Press: Piscataway, NJ, USA, 2016.
32. Vakhnin, A.; Sopov, E. Investigation of the iCC framework performance for solving constrained LSGO problems. *Algorithms* **2020**, *13*, 108. [CrossRef]
33. Kashan, A.H.; Akbari, A.A.; Ostadi, B. Grouping evolution strategies: An effective approach for grouping problems. *Appl. Math. Model.* **2015**, *39*, 2703–2720. [CrossRef]
34. Garey, M.R. *A Guide to the Theory of NP-Completeness. Computers and Intractability*; WH Freeman and Company: New York, NY, USA, 1979.
35. Falkenauer, E. A new representation and operators for genetic algorithms applied to grouping problems. *Evol. Comput.* **1994**, *2*, 123–144. [CrossRef]
36. Ramos-Figueroa, O.; Quiroz-Castellanos, M.; Mezura-Montes, E.; Kharel, R. Variation Operators for Grouping Genetic Algorithms: A Review. *Swarm Evol. Comput.* **2020**, *60*, 100796. [CrossRef]
37. Eiben, A.E.; Schippers, C.A. On evolutionary exploration and exploitation. *Fundam. Inform.* **1998**, *35*, 35–50. [CrossRef]
38. Mosk-Aoyama, D.; Shah, D. Fast distributed algorithms for computing separable functions. *IEEE Trans. Inf. Theory* **2008**, *54*, 2997–3007. [CrossRef]
39. Ray, T.; Yao, X. A cooperative coevolutionary algorithm with correlation-based adaptive variable partitioning. In Proceedings of the 2009 IEEE Congress on Evolutionary Computation, Trondheim, Norway, 18–21 May 2009; IEEE: Piscataway, NJ, USA, 2009; pp. 983–989.
40. Blickle, T.; Thiele, L. A comparison of selection schemes used in evolutionary algorithms. *Evol. Comput.* **1996**, *4*, 361–394. [CrossRef]
41. Zhang, B.T.; Kim, J.J. Comparison of selection methods for evolutionary optimization. *Evol. Optim.* **2000**, *2*, 55–70.
42. Smith, J. On replacement strategies in steady state evolutionary algorithms. *Evol. Comput.* **2007**, *15*, 29–59. [CrossRef]