Universidad Veracruzana

ARTIFICIAL INTELLIGENCE RESEARCH INSTITUTE

A GROUPING GENETIC ALGORITHM FOR VARIABLE DECOMPOSITION IN LARGE-SCALE CONSTRAINED OPTIMIZATION PROBLEMS

Submitted by

Guadalupe Carmona-Arroyo

AS THE FULFILLMENT OF REQUIREMENT FOR THE DEGREE OF
Ph.D. in Artificial Intelligence

Advisor: Ph.D. Marcela Quiroz-Castellanos
Supervisor: Ph.D. Efrén Mezura-Montes

December 10, 2024

Abstract

In this thesis, a Grouping Genetic Algorithm is proposed to perform the decomposition of D-dimensional vectors into sub-vectors of smaller size than D in optimization problems with hundreds or thousands of variables, i.e., D is an integer greater than or equal to 100. These problems are known as large-scale optimization problems, which are difficult to solve with traditional optimization approaches, so various ways to attack them have been proposed in the literature.

Some of these strategies include what is known as variable decomposition, in which, first, the decision vector is decomposed into m smaller sub-vectors. Each subset is optimized, and finally, the complete solution to the problem is created by the collaboration of the m solutions found for each subproblem. However, there are problems in which the decomposition of variables must be carried out carefully because the function or problem to be optimized usually presents variables that have different degrees of interaction between them. In other words, we have problems that are not completely separable, so performing the decomposition implies that the variables interacting with each other remain together in a subvector. This means that we can see the decomposition of variables as an optimization problem, particularly a discrete optimization problem since we seek to find the best possible grouping of variables.

Several methods have been found in the literature to create the variable decomposition. However, many of these methods have encountered difficulties with problems different from those studied, where new characteristics of separability of functions or interaction between variables are presented. Therefore, we propose an algorithm based on group optimization to deal with the problem of variable decomposition. This Genetic Algorithm has elements and operators based on groups, allowing us to study the search space appropriately and work on the problem discretely. It is also called the Grouping Genetic Algorithm.

The experimental results show that the decomposition created by the proposed algorithm helps to obtain better solutions to optimize the problem and make the

complete solution. In addition, the analysis shows that the decision vector decomposition results strongly influence the final optimization results. This confirms the hypothesis of the work, which indicates that the optimization results of LSO problems can be improved if the decomposition is performed under a discrete optimization approach, such as a grouping Genetic Algorithm.

Resumen

En este trabajo de tesis se propone un algoritmo genético de agrupación para realizar la descomposición de vectores D-dimensionales en sub-vectores de tamaño menor a D en problemas de optimización con cientos o miles de variables, es decir, D es un número entero mayor o igual a 100. Estos problemas se conocen como problemas de alta dimensionalidad, los cuales son difíciles de resolver con los enfoques tradicionales de optimización por lo que en la literatura se han propuesto diversas maneras de atacarlos. Algunas de estas estrategias incluyen lo que se conoce como descomposición de variables, en donde primero se descompone el vector de decisión en m sub-vectores de menor tamaño, después cada subconjunto es optimizado y finalmente la solución completa del problema es creada por la colaboración de las m soluciones encontradas para cada sub-problema.

Sin embargo, existen problemas en que la descomposición de variables debe ser cuidadosa debido a que la función o el problema a optimizar presenta variables que interactúan entre sí, dicho de otra manera, tenemos problemas que no son completamente separables por lo que realizar la descomposición implica que las variables que interactúan entre ellas permanezcan juntas en un sub-vector. Esto provoca que podamos ver a la descomposición de variables como un problema de optimización, particularmente de optimización discreta, ya que buscamos encontrar la mejor agrupación posible de variables.

En la literatura, se han encontrado diversos métodos para crear dicha descomposición de variables, sin embargo, muchos de estos métodos se han encontrado dificultades con problemas diferentes a los estudiados en donde se presentan nuevas características de separabilidad de funciones o de interacción entre las variables. Por lo anterior, proponemos un algoritmo basado en la optimización de grupos para lidiar con el problema de descomposición de variables. Este algoritmo es un algoritmo genético de agrupación cuyos elementos y operadores se basan en grupos por lo que se permite estudiar el espacio de búsqueda de una manera adecuada trabajando el problema de manera discreta.

Los resultados experimentales arrojan que la descomposición creada por el algoritmo genético de agrupación ayuda en obtener mejores soluciones al optimizar el problema y crear la solución completa. Además, se demostró que existe una fuerte influencia de los resultados de descomposición del vector de decisión en los resultados finales de optimización. Lo que confirma la hipótesis del trabajo, la cual indica que los resultados de optimización de problemas con alta dimensionalidad pueden ser mejorados si la descomposición es realizada bajo un enfoque de optimización discreta, particularmente, un algoritmo genético de agrupación.

Acknowledgements

I deeply thank to Consejo Nacional de Humanidades, Ciencias y Tecnologías (CONAHCYT) for its financial support in developing this work.

To the Universidad Veracruzana and the Instituto de Investigaciones en Inteligencia Artificial (IIIA) for the academic tools provided throughout my PhD studies.

To the Sociedad Matemática Mexicana for the financial support to finish this thesis.

To my advisor, Ph.D Marcela Quiroz Castellanos, for her unconditional support and wisdom in guiding me through the process. Their insightful feedback was instrumental in helping me achieve my goals.

To Ph.D Efrén Mezura Montes for their invaluable guidance, encouragement, and expertise throughout this research.

To my committee members for their time, constructive feedback, and suggestions that significantly improved the quality of this work.

To Javier for his invaluable love and for being my constant strength.

To my parents Lucía and Félix, my sister and brother for always being there for me, and to my beloved nephew and niece, who are my daily motivation.

To the friends who accompany me and encourage me to continue, and those who are no longer physically here.

Table of Contents

List of Figures

List of Tables

List of Acronyms

LSO	Large-scale Optimization
EA	Evolutionary Algorithm
EP	Evolutionary Programming
ES	Evolutionary Strategie
GA	Genetic Algorithm
DE	Differential Evolution
CC	Cooperative Co-evolution
CCGA	Cooperative Co-evolution Genetic Algorithm
CCEA	Cooperative Co-evolution Evolutionary Algorithm
GGA	Grouping Genetic Algorithm
GGA-VD	Grouping Genetic Algorithm for Variable Decomposition
VIIC	Variable Interaction Identification Technique for Constrained Problems
VIICN	Neighbor Variable Interaction Identification Technique for Constrained Problems
DVIIC	Dynamic Variable Interaction Identification Technique for Constrained Problems
CCVIL	Cooperative Co-evolution with Variable Interaction Learning
DECC-G	Differential Evolution Cooperative Coevolution with Grouping Decomposition
SaNSDE	Self-adaptive Differential Evolution with Neighborhood Search
DECC-CIG	Cooperative Co-evolutionary Differential Evolution Algorithm with

	Correlation Identification Grouping
CCVF	Cooperative Co-evolution with Variable Grouping and Filled Function
RG	Random Grouping
DG	Differential Grouping
DG2	Differential Grouping version 2
MLCC	Multilevel Cooperative Co-evolution
CMA-ES	Covariance Matrix Adaptation Evolution Strategy
CC-CMA-ES	Cooperative Co-evolution Covariance Matrix Adaptation Evolution Strategy
DECC-D	Differential Evolution with Cooperative Co-evolution using Delta-Grouping
DECC-CIG	Co-evolutionary Differential Evolution algorithm with Correlation Identification Grouping
TPLSO	Two-phase Learning-based Swarm Optimizer
DCCC	Difficulty and Contribution-based Cooperative Co-evolution
GPU	Graphics Processing Unit
CUDA	Compute Unified Device Architecture
COCC	Constraint-objective Cooperative Co-evolution
MA	Memetic Algorithm
PSO	Particle Swarm Optimization
ACO	Ant Colony Optimization
ABC	Artificial Bee Colony
EDA	Estimation of Distribution Algorithm
MTS	Multiple Offspring Sampling
DE-LoCoS	Differential Evolution Local Cooperative Search
CEC	Competitions on Large Scale Global Optimization
HC	Hill Climbing
VNS	Variable Neighborhood Search

SA	Simulated Annealing
TS	Tabu Search

Chapter 1

Introduction

1.1 Background

Given the number of variables in the decision vector and the individual characteristics of the functions to be optimized, such as non-separability conditions, one of the main challenges when using decomposition methods is creating subgroups with less dimension than the decision vector. In the specialized literature, this creation of subgroups is known as variable decomposition or variable grouping. It is well known that poor grouping of variables causes optimization algorithms to obtain local optima, or worse still, not even approach an optimum [1]. To these complications, we also add that optimization problems become even more complex when they have constraints, which are the problems studied in this thesis.

Although various algorithms exist in the literature for variable decomposition in Large-scale Optimization (LSO) problems, some present complex calculations and require prior information about the issue or function to be optimized; therefore, we determined that a new simple and fast element optimization proposal is necessary.

In this work, we propose and study the performance of an algorithm to optimize the decomposition, called GGA-VD. We analyze the influence of the algorithm components on the obtained solutions and their aptitude. In addition, the decomposition obtained is evaluated through a cooperative co-evolution strategy and a

memetic algorithm.

Experimental results showed that the GGA-VD is an adequate tool for decomposing variables in methods dedicated to solving large-scale problems.

In the present Section, we describe the problem statement, motivation, and objectives for the present thesis.

1.2 Problem statement

This work seeks to propose an alternative and novel solution to improve the algorithms that solve Large-scale Optimization problems, mainly focusing on decomposition-based methods that integrate the divide-and-conquer strategy. In these methods, during the first step, the decision vector $\mathbf{x} \in \mathbb{R}^D$ is divided into disjoint sub-vectors of dimension less than D. These sub-vectors are optimized separately by some optimization method, usually an Evolutionary Algorithm (EA). Finally, in the third and final step, these solutions to the sub-problems cooperate to create the complete solution to the problem.

As mentioned, the variables are grouped or decomposed in the first stage. This is the main problem that needs to be solved in this thesis. The objective is to create an approach that does not require a priori information on the functions or issues to be optimized and can provide a good decomposition so that good results can be obtained by optimizing the sub-problems separately.

According to the literature, finding a suitable decomposition of the decision vector greatly influences the solution of the complete problem. Therefore, our approach is based on optimizing said decomposition and then measuring its efficiency in steps two and three of the decomposition-based approaches.

To the above, we add that optimization problems usually have constraints, which decreases the performance of the algorithms that solve optimization problems. That said, we face issues with high dimensionality, with constraints that are usually among the most complicated in the specialized literature, so we consider that effective and quick strategies should be proposed for their solution.

1.3 Justification

This work arises from the need to generate knowledge to develop solution strategies for Large-scale Constrained Optimization problems. One of the main strategies for solving these problems is decomposing the decision vector of the problem in an optimized way so that the sub-component optimization algorithms work correctly. Said decomposition is not trivial and requires adequate strategies to be optimized, so finding the appropriate variable grouping is also an optimization problem that we solve through an innovative algorithm inspired by the solutions for the well-known grouping problems.

We treat the decomposition problem analogously to grouping problems, which are classic in the specialized literature on combinatorial optimization. Thus, we seek to adapt a classic algorithm, the Grouping Genetic Algorithm, to solve variable decomposition and obtain competent results of the optimization strategies.

One of the main benefits of this type of algorithm is the ease of handling solutions that involve groupings of elements, which motivates us to test its performance in variable grouping.

1.4 Thesis objectives

1.4.1 Thesis main objective

Develop a variable decomposition algorithm focused on the variable decomposition problem to efficiently solve Large-scale Constrained Optimization problems by analyzing the performance of the algorithm and its contribution to reducing the complexity of the large-scale problems and improve the optimization results.

1.4.2 Thesis specific objectives

- Study of large-scale optimization problems to identify the factors that determine their complexity.

- Development of heuristic strategies for variable decomposition that improve the performance of large-scale optimization techniques.

- Experimental application to generate knowledge about the behavior of variable decomposition algorithms and how they relate to the solution algorithm and its result.

1.5 Thesis hypothesis

It is possible to develop a variable decomposition algorithm based on group representation to solve the variable decomposition into strategies that solve large-scale optimization problems that compete with the results in the literature concerning the quality of the found solutions measured by the fitness function of each problem obtaining smaller values than those reported in the literature.

1.6 Research Contributions

- Conference talks:

  - International Workshop on Numerical and Evolutionary Optimization, 2020
  - International Workshop on Numerical and Evolutionary Optimization, 2021
  - International Workshop on Numerical and Evolutionary Optimization, 2022
  - Escuela Latinoamericana de Verano de Investigación de operaciones, 2022 - Tecnológico de Monterrey, Mexico

- Journal papers:

  - [2] G. Carmona-Arroyo, M. Quiroz-Castellanos, and E. Mezura-Montes, "Variable decomposition for large-scale constrained optimization problems using a grouping genetic algorithm," *Mathematical and Computational Applications*, vol. 27, no. 2, p. 23, 2022.

- Book chapters:

  - [3] G. Carmona-Arroyo, J. B. Vázquez-Aguirre, and M. Quiroz-Castellanos, "One-dimensional bin packing problem: An experimental study of instances difficulty and algorithms performance," *Fuzzy Logic Hybrid Extensions of Neural and Optimization Algorithms: Theory and Applications*, pp. 171–201, 2021.

1.7 Document structure

- Chapter 2: In this chapter, we define Large-scale Optimization Problems and the optimization algorithms used to solve them, mentioning the most commonly used in the literature. We also define problems with constraints and the method used in this work to solve constrained large-scale problems. Finally, we describe the test functions on which this work is evaluated and their particular characteristics.

- Chapter 3: This chapter presents the problem of variable decomposition and its importance in large-scale problems. It also reviews the specialized literature and classifies the methods used in this review to deal with decomposition in various methods used to solve these problems. Finally, a method proposed in the literature is presented based on optimizing the decomposition through a genetic algorithm.

- Chapter 4: Here, we present the main proposal of this work, a Grouping Genetic Algorithm based on optimizing groups to solve the variable decomposition problem in large-scale problems. First, we make an analogy between

this problem and other discrete optimization grouping problems, mentioning the methods used in the specialized literature to solve them. Then, we list all the elements of this algorithm formally and how they were implemented, as well as examples of its operation.

- Chapter 5: This chapter presents the experiments performed in this work. First, we compare our proposal against another well-known decomposition algorithm in the literature and compare it against the first genetic algorithm proposed for the decomposition of variables, showing the results in comparative tables. Then, the experiments performed to improve our proposal using different algorithm components and the results obtained are shown. Next, the results of the function optimization process evaluating our decomposition proposal are shown. Finally, some essential characteristics found in the decomposition and optimization process are shown, identifying relationships and patterns of the functions, their particularities, and results.

- Chapter 6: Finally, in this chapter, we present our general discussion and conclusions on the performance of our proposal and its operation in optimizing the decomposition of variables to optimize large-scale problems. We identify areas of opportunity for study in the proposal and mention work to be done in the future.

Chapter 2

Large-scale Constrained Optimization

To begin with, it is necessary to show the importance of studying large-scale problems, how they have been solved in the specialized literature, and why they are essential for solving them. This chapter discusses the strategies used to show the panorama of methods that focus on solving these problems. We also note the importance of addressing large-scale constrained problems and present the benchmark set with which our proposal is evaluated.

2.1 Large-scale Optimization Problems

Optimization problems have been defined as large-scale when they have a large number of variables in the decision vector [4, 5, 6]. The research in the literature for unconstrained large-scale problems consider problems of 100 dimensions or more large-scale [5, 7]. Another perspective considers an optimization problem large when it is difficult to optimize and has a complex structure [8].

We are considering the first perspective in this thesis, and the problems we use to test our algorithm are constrained; an essential fact of choice to work with constrained problems is that most specialized works focus on strategies to solve

unconstrained problems. However, it is well-known that a constraint problem is more difficult to solve, regardless of the algorithm used.

In Figure 2.1, we present a bar graph to compare the number of articles that have been published each year, from January 2000 to October 2024, using the terms ”large-scale optimization” versus the term ”large-scale constrained optimization” in the article according to Google Scholar. This graph aims to demonstrate the importance of generating more knowledge along the lines of algorithms that solve problems with constraints. The second term produced a significantly lower number of publications than the first one, and this number does not seem to be increasing.

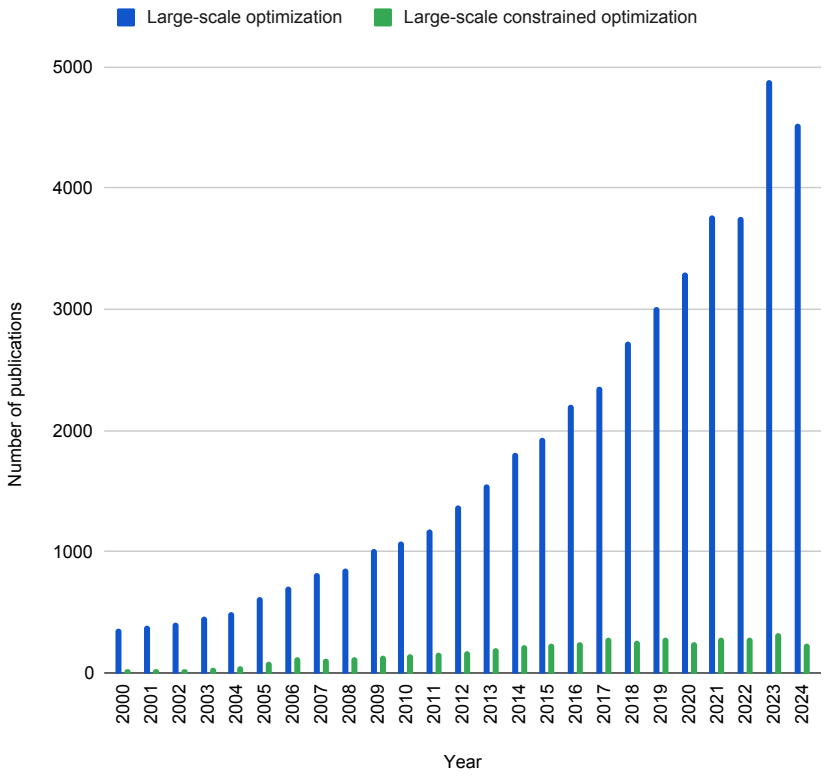


Figure 2.1: Number of yearly published articles with the term ”large-scale optimization” and the term ”large-scale constrained optimization” from January 2000 until October 2024.

Given the above, we formally define the type of problems that we will use in this work: a constrained numerical optimization problem is determined 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 [9]. This is described by Equation (2.1).

$$\begin{aligned} &\text{minimize} \quad Obj(\mathbf{x}) \\ &\text{subject to} \\ &\qquad g_j(\mathbf{x}) \leq 0, \quad j = 1, \ldots, q \\ &\qquad h_k(\mathbf{x}) = 0, \quad k = 1, \ldots, r. \end{aligned} \tag{2.1}$$

where q and r represent the number of inequality and equality constraints, respectively, $\mathbf{x} = (x_1, ..., x_D)$, 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 $\mathbb{F} \subset \mathbb{S}$. Furthermore, the vector $\mathbf{x}$ is D-dimensional with $D \geq 100$.

Generally, when an optimization problem involves many variables in the decision vector is solved, the algorithms that solve it have difficulty finding a good solution, known as the *curse of dimensionality*. However, there are well-known state-of-the-art approaches to working with these problems.

The state-of-the-art strategies for LSO are based on decomposition-based and nondecomposition-based algorithms. Most of them also use evolutionary algorithms to optimize. The following sections describe these approaches in a general way.

2.2 Large-scale Optimization Techniques

Two main approaches to tackling LSO problems can be found in the specialized literature: decomposition-based algorithms and non-decomposition-based methods. These last methods solve LSO problems as a whole, and they are designed with specific effective operators or combined with another optimization method to further enhance their performance in exploring complex search spaces.

Decomposition methods are based on the divide-and-conquer approach, which decomposes LSO problems into single-variable or multiple low-dimensional disjoint sub-components. Figure 2.2 summarizes these methods and their approaches according to the specialized literature [10, 11] in a summarized way.

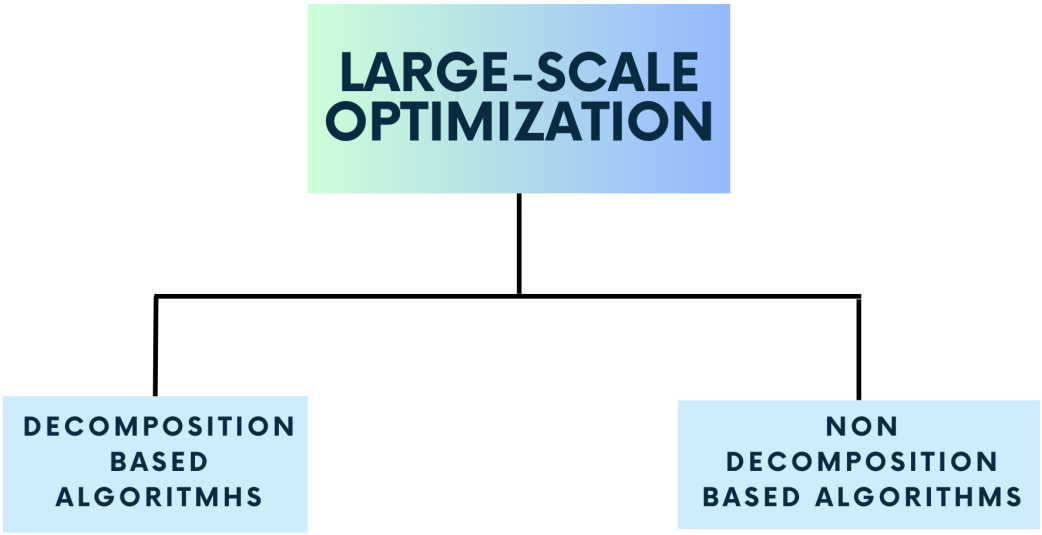


Figure 2.2: A classification of the Large-scale Optimization techniques.

This classification is current; however, several particular mechanisms have been proposed since LSO is an essential challenge in the science and engineering fields. Since some classical optimization methods require a starting point and many evaluations, and fast ones require information on the function derivative, which may not be present or be complex to calculate, other strategies, such as heuristics and meta-heuristics, have been developed to alleviate the disadvantages of these methods.

These strategies have presented problems when trying to solve LSO problems. Still, they have been used as a basis for algorithms that focus on large-scale. In the following sections, we describe the Evolutionary Algorithms that have been the most used for LSO approaches, as well as the algorithms that are part of the

classification in Figure 2.2 and have been developed over the years by the specialized scientific community.

2.2.1 Evolutionary Algorithms

Evolutionary Algorithms are meta-heuristics; they are search processes that can be applied to different issues. These algorithms are directly inspired by Darwinian evolution; in general, they share the following characteristics [12]:

- Individuals from one or more populations compete for resources.
- Populations change over time.
- Individuals can reproduce, passing their genetic load to future generations.
- There are selection processes according to the aptitude of the individuals.

According to the specialized literature, three main paradigms are often discussed and used to solve optimization problems:

- Evolutionary Programming (EP)
- Evolutionary Strategies (ESs)
- Genetic Algorithms (GAs)

Evolutionary programming [13] was initially designed to evolve finite state machines by emphasizing the phenotype space, utilizing mutation as the sole evolution operator, and employing an elitist replacement scheme based on individual fitness scores. There is no initial selection: every individual in the current population generates one offspring. Finally, the best P individuals among parents and offspring become the next generation's parents.

On the other hand, Evolutionary Strategies were developed to solve highly complex hydrodynamic problems by a group of engineering students [14]. ESs are characterized by the impartial selection of individuals for recombination and a deterministic process. They differ from the rest of the Evolutionary Algorithms, mainly

in the form of the mutation operator. They are primarily applied in continuous optimization problems where the representation is through vectors of real numbers.

Finally, Genetic Algorithms are perhaps the best-known of all the evolutionary computation approaches. These were proposed by Goldberg and Holland [15] in the 1960s.

Genetic algorithms are population-based, and unlike the algorithms mentioned above, they base their search on crosses (reproduction), i.e., they consider all individuals part of the same species and can generate children by combining their information. That is not why they leave aside the mutation process, since every time a child is generated, it is exposed to said process, but it is not always fulfilled. A GA is the base for the algorithm proposed in this work. Therefore, we present a general algorithm in Algorithm 1.

Algorithm 1: Genetic Algorithm

```
1 Generate an initial population
2 Evaluate population for each generation do
3     Selection of parents
4     Apply crossover to parents operator to generate offspring
5     Apply mutation operators to offspring
6     Evaluate new individuals
7     Replace population with a new individual
```

Initially, GAs work under extinction; the new generation directly replaces the previous generation. However, elitism can be added to keep the best solution within the current population.

Another important algorithm to tackle LSO is Differential Evolution (DE). DE was proposed by Storn and Price in 1995 [16], to deal with continuous search spaces. The canonical algorithm is DE/rand/1/bin, ”rand” means that it uses a random

selection of the base vector that is used in the mutation, "1" means that the operator is based on a difference, and "bin" means that the crossover operator is binomial. The Differential Evolution process can be described in four steps listed below:

- Initialization: An initial Pop population of NP solutions $x_{i,0}$, $i = 1, ..., NP$ is randomly generated. For each decision variable $x_{j,i,0}$, a random value is generated between the limits of the function, as shown in Equation 2.2.

$$x_{j,i,0} = l_j + rand_j(0,1) \times (u_j - l_J), \tag{2.2}$$

  where $rand_j(0,1)$ is a random number generated with a uniform distribution between 0 and 1.

- Mutation operator: in each generation g, for each of the vectors in the population (called target) $x_{i,g}$, a mutant solution $v_{i,g+1}$ is generated according to Equation 2.3.

$$v_{i,g+1} = x_{r_0,g} + F(x_{r_1,g} - x_{r_2,g}), \tag{2.3}$$

  where $r_0, r_1, r_2 \in 1, 2...NP$ are randomly selected indices from the population, different from each other and different from i (i.e., $r_0 \neq r_1 \neq r_2 \neq i$). $F > 0$ is a scale factor that controls the amplitude of the difference vector, made up of $x_{r_1,g}$ and $x_{r_2,g}$. $x_{r_0,g}$ is the so-called basis vector.

- Crossover operator: after generating the mutant vector, the vector target $x_{i,g}$ is combined with the mutant vector $v_{i,g+1}$ to obtain the trial vector $u_{i,g+1}$. This crossover operator is performed according to Equation 2.4.

$$u_{j,i,g+1} = \begin{cases} v_{j,i,g+1} & if\ rand[0,1) \leq CR\ or\ j = j_r \\ \\ x_{j,i,g} & in\ other\ case \end{cases} \tag{2.4}$$

  Where $j = 1, 2, ..., D$, $rand[0,1)$ is a random number with a uniform distribution between 0 and 1, and j_r is an index of a decision variable selected

randomly, with the objective that at least one value of the mutant vector belongs to the vector trial and avoid copies of the parent vector. $CR \in [0, 1]$ is the crossover rate.

- Selection: according to the fitness value of the objective function, if $u_{i,g+1}$ is better than the target vector $x_{i,g}$, then $u_{i,g+1}$ will be selected to pass to the next generation. Otherwise, $x_{i,g}$ is maintained in the next generation, assuming minimization, according to Equation 2.5.

$$x_{i,g+1} = \begin{cases} u_{i,g+1} & if\ f(u_{i,g+1}) \leq f(x_{i,g}) \\ \\ x_{i,g} & in\ other\ case \end{cases} \tag{2.5}$$

It is essential to note that there are several versions of DE. In 2020, Pant et al. [17] published a survey with a review of them. They studied papers that have proposed an improvement of DE for approximately 25 years. They present 283 research articles that have been covered, and the journey of DE is shown through its essential aspects like population generation, mutation schemes, crossover schemes, variation in parameters, and hybridized variants, along with various successful applications of this strategy.

2.2.2 Decomposition-based Algorithms

Another way to call decomposition-based algorithms is algorithms based on the Cooperative Co-evolution approach, which Potter and De Jong proposed [18]. They developed Cooperative Co-evolution Genetic Algorithm (CCGA), taking as inspiration the classical techniques that optimize each one of the variables of the problem separately, leaving the rest with a constant value. This way, as many species as variables in the situation are created, the Genetic Algorithm is applied to each variable. As they mention in their work, this algorithm verified that the Cooperative Co-evolution (CC) scheme was significantly better than the traditional Genetic Algorithm in some popular benchmark functions.

When applying the CC framework to solve a particular problem, a standard approach is to decompose the problem into sub-components and assign each sub-component to a sub-population. A particular EA evolves these sub-populations and co-evolved simultaneously. The scheme proposed by Potter and De Jong [18] can be generalized as follows:

1. Problem decomposition: A specific decomposition method divides the D-dimensional decision vector into k sub-groups.

2. Optimization process: Each sub-group is separately evolved with a particular EA.

3. Cooperative process: All sub-groups are combined to form the whole solution and exchange the information among subgroups.

The steps mentioned above are repeated for a predefined number of iterations.

Algorithm 2: Cooperative Co-evolution

```
1  gen = 0
2  for each sub-component S do
3      Population_s(gen) = randomly initialized population
4      evaluation fitness of each individual in Population_s(gen)
5  while not terminated do
6      gen = gen + 1
7      for each sub-component S do
8          Select Representative_s(gen) from Population_s(gen − 1) based on
             fitness
9      for each sub-component S do
10         apply mutation, crossover and selection operator of DE to
             Population_s(gen)
11         evaluate fitness of individual in Population_s(gen) using
             Representative_s(gen)
```

The authors mention that the CC scheme is significantly better than the traditional Genetic Algorithm in solving some classical state-of-the-art functions. However, they also concluded that, like the classical methods, its performance degrades when the problem variables interact. Algorithm 2 shows the CC general procedure.

The advantages of CC using EAs over the traditional evolutionary algorithm result mainly from its divide-and-conquer decomposition strategy.

These algorithms have mostly four advantages. First, the decomposition of the problem allows parallelism to speed up the optimization process. Second, each sub-problem is solved with a separate sub-population, maintaining good solution diversity. Third, decomposing a system into sub-problems increases the robustness against module errors and failures and thus enhances the reusability in dynamic environments. Finally, the "curse of dimensionality," i.e., the rapid deterioration in performance with increased decision variables, can be alleviated if the problem is appropriately decomposed [19].

Cooperative Co-evolution Evolutionary Algorithms (CCEAs) have been successfully applied to various optimization problems across real-world application areas. The Potter and De Jong framework has been adopted with many other metaheuristic algorithms, thus injecting vitality into classic optimization solutions.

In 2008, Yang et al. [20] proposed a CC-based algorithm called Differential Evolution Cooperative Coevolution with Grouping Decomposition (DECC-G), which uses a predefined group size to decompose the decision vector into multiple subcomponents. Each sub-component is optimized separately using Self-adaptive Differential Evolution with Neighborhood Search (SaNSDE) [21]. Instead of using Potters's greedy collaboration method, they used the adaptive weighting strategy to co-adapt the sub-components. The adaptive weighting strategy applies a weight to each sub-component after every cycle and then evolves the weight vector with Differential Evolution. The basic steps of the DECC-G algorithm are as follows:

1. Start a cycle

2. Randomly split the D-dimensional decision vector into subcomponents of size

s, i.e., $D = m \times s$, where m is the number of sub-components.

3. Optimize each sub-component separately using SaNSDE.

4. Apply the weight vector to each optimized sub-component and then optimize the weight vector for the current population's best, random, and worst members.

5. If stop criteria are met, stop the iterations; otherwise, go to step 1.

Yang et al. proposed Multilevel Cooperative Co-evolution (MLCC) [22] in the same year, which considers a pool of group size, also called a decomposer pool. Corresponding to each group size, a performance record is maintained and updated at the end of every cycle. At the start of a new cycle, a decomposer or group size is selected from the decomposer pool based on its past performance. Then, the problem is decomposed into small components using this decomposer. Each sub-component evolves in a round-robin fashion for specific fitness evaluations, and then, at the end of a cycle, group-size performance is stored. The process is repeated until termination criteria are reached.

Later, MLCC [22] was further improved by Yang et al. [23]. They introduced three techniques to improve its performance. These techniques included frequent random grouping, removing adaptive weighting strategy for co-adaption, and self-adaption of sub-component sizes. The authors realized that the Adaptive weighting strategy is computationally expensive and wastes a lot of fitness evaluation. So, they suggested that instead of using many fitness evaluations in adaptive weighting, if these fitness evaluations are used to increase the number of cycles, the frequency of random grouping will increase, improving algorithm performance.

Liu et al. proposed Cooperative Co-evolution Covariance Matrix Adaptation Evolution Strategy (CC-CMA-ES) for LSO problems in 2013 [24]. They are improving an evolutionary algorithm. In CC-CMA-ES, the authors have used two new decomposition strategies based on the diagonal of the Covariance Matrix Adaptation Evolution Strategy (CMA-ES): a min-variance decomposition strategy and a

Max-variance decomposition strategy. These new decomposition strategies balance exploration and exploitation; adaptive and appropriate decomposition strategies complement this.

Dependency identification techniques were necessary to find the optimal group size by identifying the interacting variables. However, work done in this area is minimal. Wicker [25] proposed a simple technique for identifying interacting variables based on Potter's non-greedy collaboration strategy [26] proposed in his Ph.D.thesis.

Later, Chen et al. [27] proposed Cooperative Co-evolution with Variable Interaction Learning (CCVIL). They have used Wicker's [25] dependency identification technique to identify interacting variables to place them in one group for the subsequent evolutionary cycles. CCVIL dynamically finds interacting variables through the algorithm's evolution. It is one of the state-of-the-art algorithms in the LSO field. Though CCVIL successfully found optimal group sizes for most problems. However, it is computationally expensive. Around 60 percent of total FEs were consumed in the learning stage, which leaves only less than 40 percent for the optimization stage.

That same year, Omidvar et al. [23] proposed Differential Evolution with Cooperative Co-evolution using Delta-Grouping (DECC-D), which uses delta value to identify interacting variables. Delta value corresponding to a decision variable is calculated by measuring the amount of change in it in successive iterations. DECC-D sorts the delta values, and decision variables corresponding to the two smallest delta values are merged in one subcomponent.

Omidvar et al. have used a gradient approximation technique in one of the most famous strategies for LSO, which is Differential Grouping (DG) [28] proposed in 2013. This gradient approximation was used to find the interacting variables in the same sub-component. It successfully identifies direct interaction among decision variables but fails to identify indirect interaction among them.

In 2015, Sun et al. [29] improved the DG technique and proposed Extended Differential Grouping. This new technique successfully identifies direct and indirect

interaction among decision variables. Mei et al. [30] also proposed an improvement of Differential Grouping in 2016. They adopt a modified CMA-ES as the base optimizer for solving sub-problems.

Some other CC-based algorithms for LSO are the Co-evolutionary Differential Evolution algorithm with Correlation Identification Grouping (DECC-CIG) [31], Cooperative Co-evolution with Variable Grouping and Filled Function (CCVF) [32], and Variable Grouping Differential Evolution algorithm [33] all of them proposed between 2013 and 2014.

Since 2014, several proposals have been made to improve the proposals of Omidvar et al.; Delta and Differential Grouping Cooperative Co-evolution are two of the most popular strategies to solve LSO, mainly for their proposed decomposition strategies. All these improvements will be described in Section 3.2 tacking, given that they improve the variable decomposition strategies.

Lan et al. [34] propose a Two-phase Learning-based Swarm Optimizer (TPLSO) for large-scale optimization. Inspired by cooperative learning behavior, mass learning, and elite learning are involved in TPLSO. In the mass learning phase, the algorithm randomly selects three particles to form a study group and then adopts a competitive mechanism to update the current members. Then, the particles are sorted in the swarm, picking out the elite particles with better fitness values. In the elite learning phase, the elite particles learn from each other to search for more promising areas.

Xu et al. [35] analyze the nature of the sub-problems from their difficulty and contribution to LSO problems. They propose a method to quantify the optimization difficulty of the functions during the evolution process, which considers the fitness landscape's difficulty and the optimization algorithm's behaviors. Then, they propose a difficulty and contribution-based CC framework, called Difficulty and Contribution-based Cooperative Co-evolution (DCCC), which encourages allocating the computational resources to more contributing and difficult sub-problems.

Vakhnin et al. [36] proposed a method for selecting the number of sub-components and the population size during the optimization process from a predefined pool of

parameters based on their performance in the previous optimization steps. The main contribution is the improvement of co-evolutionary decomposition-based algorithms for solving LSO problems.

Kelkawi et al. [37] proposed a parallel implementation of the cooperative co-evolution framework for solving LSO problems, using the Graphics Processing Unit (GPU) and Compute Unified Device Architecture (CUDA) platform to expose a degree of parallelism. Features of the GPU parallel technology and CUDA platform, such as shared and global memories, are used to optimize the sub-components of the problem in parallel.

Recently, Kelkawi et al. [38] proposed a distributed implementation of the cooperative co-evolution framework for solving large-scale global optimization problems on the Apache Spark distributed computing platform. They use a distributed variant of the cooperative co-evolution framework, and features of the Spark platform are utilized to enhance the computational speed of the algorithm.

After that, Duan et al. [39] propose parallelizing the covariance matrix adaptation evolution strategy and present a multilevel learning-based framework for distributed limited memory to do so efficiently.

Tacking of constrained LSO Xu et al. [40] proposed a Constraint-objective Cooperative Co-evolution (COCC) framework for large-scale continuous constrained optimization problems, which is based on the nature of the objective and constraint functions: modular and imbalanced components. The COCC framework allocates computing resources to different sub-components according to the impact of objective values and constraint violations.

2.2.3 Non-decomposition-based algorithms

In addition to decomposition-based algorithms, some authors have explored the usage of strategies based on a different framework, like the Memetic Algorithms (MAs), which are the combination of a global search with local search. Some examples are the works of Zhang and Li [41], where a local optimizer was adopted

as a second optimization stage. Cao et al. [1] developed an MA with an improved DE version called SaNSDE [21] as a global optimizer, and Solis and Wets [42] algorithm as a local search.

The specialized literature reports that MAs avoid the usage of CC to solve large-scale optimization problems. One of the first proposals in this area was the one proposed by Hong-qi Li and Li in 2007 [43], who designed a Particle Swarm Optimization (PSO) algorithm, adding the Harmony Search algorithm into the updating process of PSO.

Another MA based on PSO was developed by Zhao et al. in 2008 [44], where they tried to improve the best solution by a Quasi-Newton method every certain number of iterations. After that, in 2009, Muelas et al. proposed [45] an algorithm based on Differential Evolution with a Multiple Offspring Sampling (MTS) [46] as a local search method.

Molina et al. [47] presented an MA based on local search chains. LaTorre et al. [48] developed a hybrid algorithm using the Multiple Offspring Sampling framework, where they combined the Solis and Wets [42] method and MTS [46]. The algorithm showed a high performance in non-separable problems. An improved simplex crossover for an improved DE version called SaDE was designed by Zhang et al. [49].

As we studied, this particular class of algorithms is population-based meta-heuristics that use a principal search motor and a set of local search algorithms activated within the generation cycle of the external algorithm, in most cases, an evolutionary algorithm. In general, the Memetic Algorithm procedure consists of the following three main phases:

- Generation: Represents a cycle of the selected evolutionary algorithm.

- Local search phase: Improves the solutions from the population by applying one or more local search engines. Additionally, this phase occurs within the evolutionary cycle.

- Replacing the result of the local phase: Integrates the outcome of the local search into the solution, either by improving it or adding it to the population.

A novel work was proposed by Aguilar-Justo et al. [50]; they combined the essences of CC and Memetic Algorithms. Their proposal was called Differential Evolution Local Cooperative Search (DE-LoCoS), where a MA optimizes the candidate solution according to the subgroups obtained by the decomposition method.

Algorithm 3: DE-LoCoS Memetic Algorithm

```
1  Set MaxFEs
2  Cyclefrecuency = (4D)/(0.3NP)
3  MaxLocalFes = 0.3MaxFES
4  Pop = randomly initialized population
5  Evaluate initial population
6  while MaxFEs are not achieved do
7      for i = 1 to NP do
8          Select vectors r0 ≠ r1 ≠ r2 ≠ i
9          apply mutation, crossover operator of DE
10         Upgrade better element according to feasibility rules
11     if currentCycle mod Cyclefrecuency == 0 and MaxLocalFes are not
         achieved then
12         Δ = stdx(Pop)
13         Sn = Decompostion()
14         for k = 1 to K do
15             x_new = LocalOptimizer()
16     Upgrade better element according to feasibility rules
```

The exchange of information between the subgroups can follow the same strategies as in CC, e.g., a sequential exchange. In this strategy, after optimizing one subgroup, the data of the optimized subgroup is used to continue the optimization

of the other variable subcomponents or a parallel approach. In this way, all the subgroups are optimized simultaneously, and in the end, all the information is united to generate a single solution. Algorithm 3 shows the Memetic DE-LoCoS procedure proposed by Aguilar-Justo et al. [50].

DE-LoCoS is also one of the few works focused on problems with constraints, so it is an essential part of this study and is considered for this work.

2.3 Constrained Optimization

As mentioned, most of the work to solve LSO problems focuses on unconstrained problems. However, we have decided to collaborate on the specialized literature for constrained optimization problems. Various strategies exist to solve constrained problems. Some of these strategies modify the problems or functions to resemble unconstrained problems, and then the LSO algorithms can solve them efficiently. This section presents some popular methods for managing constraints.

According to Mezura-Montes [51], one of the most popular and effective constraint-handling techniques in current use belonged to this category and was initially proposed by Deb [52]. In this approach, a set of three feasibility criteria were proposed for a binary tournament selection in a GA as follows:

1. The one with the best objective function is chosen when comparing two feasible solutions.

2. The feasible one is chosen when comparing a feasible and an infeasible solution.

3. The one with the lowest sum of constraint violations is chosen when comparing two infeasible solutions.

The sum of constraint violations can be calculated as follows:

$$\Phi(\mathbf{x}) = \sum_{j=1}^{q} \max(0, g_j(\mathbf{x}))^2 + \sum_{k=1}^{r} |h_k(\mathbf{x})|, \tag{2.6}$$

where $g_j(\mathbf{x})$ represents the inequality constraints and $h_k(\mathbf{x})$ are the equality constraints, and all of their values are normalized.

Another strategy to solve constrained problems is penalty functions [53, 51]. Those functions are based on mathematical programming approaches, where a constrained optimization problem is transformed into an unconstrained problem. The general formula is given by Equation 2.7.

$$\Phi(\mathbf{x}) = f(\mathbf{x}) + p(\mathbf{x}) \tag{2.7}$$

where $\Phi(\mathbf{x})$ is the expanded objective function to be optimized, and $p(\mathbf{x})$ is the penalty value that can be calculated as follows:

$$\Phi(\mathbf{x}) = \sum_{j=1}^{q} t_j \max(0, g_j(\mathbf{x}))^2 + \sum_{k=1}^{r} c_k |h_k(\mathbf{x})|, \tag{2.8}$$

where t_j and c_k are positive constants called "penalty factors" and $g_j(\mathbf{x})$, $h_k(\mathbf{x})$ are the inequality and equality constraints respectively.

As noted, the aim is to decrease the fitness of infeasible solutions to favor the selection of feasible solutions. In Equation 2.7, the penalty value is added to the fitness of a solution because low values are preferred as expected in a minimization problem.

We integrated a constraint violation sum cvs as a style of penalty function to convert the constrained optimization problems into unconstrained problems. This sum is calculated by Equation 2.9

$$cvs(\mathbf{x}) = \sum_{j=1}^{q} \max(0, g_j(\mathbf{x})) + \sum_{k=1}^{r} \max(0, |h_k(\mathbf{x})| - \epsilon) \tag{2.9}$$

where a small tolerance $\epsilon = 1e^{-4}$ is adopted and $g_j(\mathbf{x})$, $h_k(\mathbf{x})$ are the inequality and equality constraints respectively.

2.4 Benchmark functions

The algorithms studied in this work were tested in a benchmark test proposed by Sayed et al. [54]. This test set has different degrees of separability complexity, which will be described later in this section. It can be tested over three dimensions (100, 500, and 1000), given that the functions are scalable in those dimensions.

These 18 functions were created by combining six objective functions with 1, 2, or 3 constraints. The objective functions are based on some problems in the literature that have been used in the Special Sessions and Competitions on Large Scale Global Optimization (CEC) 2008, 2010, and 2013 benchmark problems [6, 7, 55].

First, the equations of base functions are shown below according to [55]. Also, Figure 2.3 shows the 3D-projection of the six base functions presented.

- **Sphere Function** (Equation 2.10):

$$F_{sphere}(\mathbf{x}) = \sum_{i=1}^{D} x_i^2 \tag{2.10}$$

  where D is the dimension and $\mathbf{x} = (x_1, x_2, ..., x_D)$ is a D-dimensional row vector (i.e., a $1 \times D$ matrix). The Sphere function is simple. This function is the separable element in this test suite when using a naturally nonseparable function to form some partially nonseparable functions.

- **Elliptic Function** (Equation 2.11):

$$F_{elliptic}(\mathbf{x}) = \sum_{i=1}^{D} (10^6)^{\frac{i-1}{D-1}} x_i^2 \tag{2.11}$$

  where D is the dimension and $\mathbf{x} = (x_1, x_2, ..., x_D)$ is a D-dimensional row vector (i.e., a $1 \times D$ matrix). The number 10^6 is the condition number, transforming a Sphere function into an Elliptic function. An orthogonal matrix rotates the coordinates to make this function nonseparable.

- **Rastrigin Function** (Equation 2.12):

$$F_{rastrigin}(\mathbf{x}) = \sum_{i=1}^{D} [x_i^2 - 10\cos(2\pi x_i) + 10] \tag{2.12}$$

where D is the dimension and $\mathbf{x} = (x_1, x_2, ..., x_D)$ is a D-dimensional row vector (i.e., a $1 \times D$ matrix). Similarly, an orthogonal matrix is also used for coordinate rotation to make it nonseparable.

- **Ackley Function** (Equation 2.13):

$$F_{ackley}(\mathbf{x}) = -20\exp\left(-0.2\sqrt{\frac{1}{D}\sum_{i=1}^{D} x_i^2}\right) - \exp\left(\frac{1}{D}\sum_{i=1}^{D}\cos(2\pi x_i)\right) + 20 + e \tag{2.13}$$

where D is the dimension and $\mathbf{x} = (x_1, x_2, ..., x_D)$ is a D-dimensional row vector (i.e., a $1 \times D$ matrix). To make it nonseparable, an orthogonal matrix is again used for coordinate rotation.

- **Schwefel Function** (Equation 2.14):

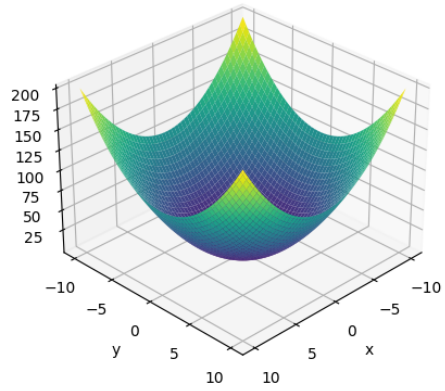
$$F_{schwefel}(\mathbf{x}) = \sum_{i=1}^{D}\left(\sum_{j=1}^{i} x_i\right)^2 \tag{2.14}$$

where D is the dimension and $\mathbf{x} = (x_1, x_2, ..., x_D)$ is a D-dimensional row vector (i.e., a $1 \times D$ matrix).

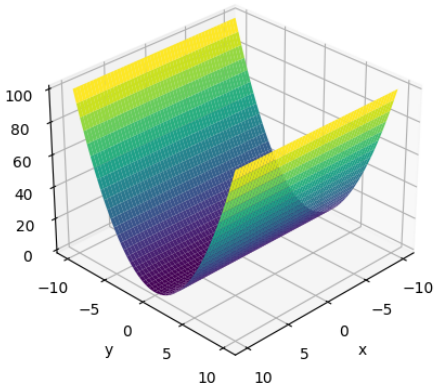
- **Rosenbrock Function** (Equation 2.15):

$$F_{rosenbrock}(\mathbf{x}) = \sum_{i=1}^{D-1} [100(x_i^2 - x_{i+1})^2 + (x_i - 1)^2] \tag{2.15}$$

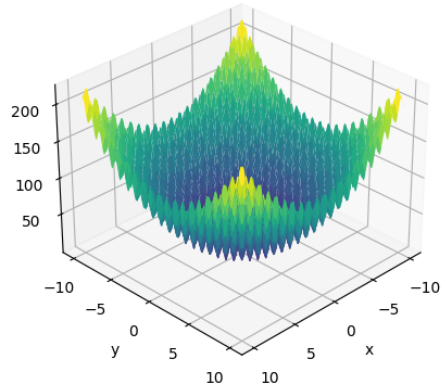
where $D \geq 2$ is the dimension and $\mathbf{x} = (x_1, x_2, ..., x_D)$ is a D-dimensional row vector (i.e., a $1 \times D$ matrix).



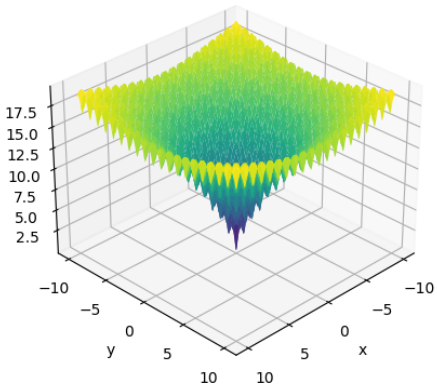
(a) Sphere Function



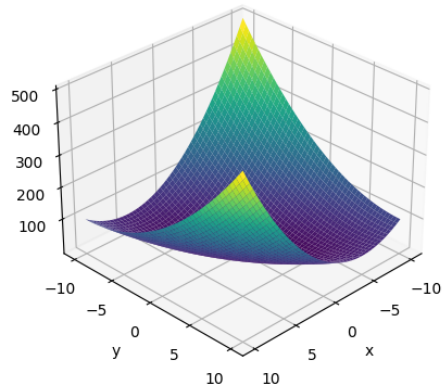
(b) Elliptic Function



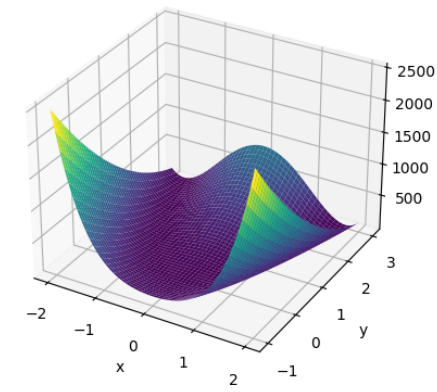
(c) Rastrigin's Function



(d) Ackley's Function



(e) Schwefel's Function



(f) Rosenbrock's Function

Figure 2.3: Base functions 3D-projection.

The details of the mathematical expression of each function are also given in this section; they were obtained from the work of Sayed et al.[56]. In addition, in Table 2.1, we show the components of these 18 test functions, that is, the objective function and the constraints that make up each function.

Table 2.1: Components of the 18 test functions. F_i represents the ith function, Obj_j the jth objective function, and g_1, g_2 and g_3, are the constraints.

Function	**Objective**	$\mathbf{g_1}$	$\mathbf{g_2}$	$\mathbf{g_3}$
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}		×	×	×

Objective functions:

- **Objective 1** (Equation 2.16): The first objective function in this test suite

has D separable variables at any dimension. When used in constrained problems, the problem has a completely separable objective function, and its complexity depends only on its constraints.

$$Obj_1 = \sum_{i=1}^{D} x_i^2 \tag{2.16}$$

- **Objective 2** (Equation 2.17): The second objective function has $D \times 0.9$ separable variables and $D \times 0.1$ nonseparable variables divided into groups of length 5.

$$Obj_2 = \sum_{count=1}^{COUNTS} \left(\sum_{i}^{n1} x_i^2 + \sum_{j}^{n2-1} [100(x_j^2 - x_{j+1})^2 + (x_j - 1)^2] + \sum_{l}^{n3} x_l^2 \right) \tag{2.17}$$

where $len = 5$, $\gamma = 0.10$, $COUNTS = (\gamma \times D)/len$, $i = (count - 1) \times len/\gamma + 1$, $n1 = (count - 1) \times len/\gamma + len$, $j = (count - 1) \times len/\gamma + len$, $n2 = (count - 1) \times len/\gamma + (1/\gamma)$, $l = (count - 1) \times len/\gamma + (1/\gamma) + 1$, and $n3 = count \times len/\gamma$.

- **Objective 3** (Equation 2.18): The third objective function has $D \times 0.9$ separable variables that use the Sphere function and $D \times 0.1$ nonseparable variables represented by separated pairs using Rosenbrock's function.

$$\begin{aligned} Obj_3 = \sum_{count=1}^{COUNTS} \Bigg(& \sum_{i}^{n1} [100(x_i^2 - x_{i+(len/\gamma)})^2 + (x_i - 1)^2] \\ & + \sum_{j}^{n2} (x_j^2 + x_{(j+len/\gamma)}^2) \Bigg) \end{aligned} \tag{2.18}$$

where $len = 5$, $\gamma = 0.10$, $COUNTS = (\gamma \times D)/(2 \times len)$, $i = [2 \times (count - 1) \times len/\gamma] + 1$, $n1 = [2 \times (count - 1) \times len/\gamma] + len$, $j = [2 \times (count - 1) \times len/\gamma] + len + 1$, $n2 = [2 \times (count - 1) \times len/\gamma] + (len/\gamma)$.

- **Objective 4** (Equation 2.19): The fourth objective function has more nonseparable variables, some overlapping. This overlapping makes the group of nonseparable variables that should be optimized in one sub-problem larger. It works like a bridge between two nonseparable groups. A parameter δ controls the spreading of the nonseparable groups over the dimension whenever it is scaled. The percentage of the nonseparable variables is $P = 20\%$ of the dimension D and is represented in four equal groups. There are $D \times 0.2$ variables that overlap sequentially between these nonseparable groups. It is important to note that when dimension increases, the number of nonseparable variables and the sequentially overlapping variables also increases.

$$
\begin{aligned}
Obj_4 = \sum_{count=1}^{COUNTS} \Bigg(& \sum_{i}^{n1-1} ([100(x_i^2 - x_{i+1})^2 + (x_i - 1)^2] \\
& + [100(x_{i+\delta}^2 - x_{i+\delta+1})^2 + (x_{i+\delta} - 1)^2]) \Big) \\
& + \sum_{i_{ov}}^{n1_{ov}-1} \big([100(x_{i_{ov}}^2 - x_{i_{ov}+1})^2 + (x_{i_{ov}} - 1)^2] \\
& + [100(x_{i_{ov}+\delta}^2 - x_{i_{ov}+\delta+1})^2 + (x_{i_{ov}+\delta} - 1)^2] \\
& + \sum_{j}^{n2} x_j^2 + \sum_{l}^{n3} x_l^2
\end{aligned}
\tag{2.19}
$$

where $len = 5$, $\gamma = 0.20$, $COUNTS = 2$, $i = ((count - 1) \times \gamma \times D/4) + 1$, $n2 = \gamma \times D/4$, $\delta = 0.6 \times D$, $i_{ov} = \gamma/4 \times D$, $n1_{ov} = (\gamma \times D/4) + (\gamma \times D/10)$, $j = (\gamma \times D/2) + 1$, $n2 = 3\gamma \times D$, $l = 0.7 \times D + 1$, and $n3 = D$. Here, δ is the position of the second nonseparable group, which is usually in the second half of the dimension and assures that the nonseparable variables are spread along the dimension.

- **Objective 5** (Equation 2.20):

$$
\begin{aligned}
Obj_5 = & \sum_{i}^{n1} \big([100(x_{2i-1}^2 - x_{2i+1})^2 + (x_{2i-1} - 1)^2] \\
& + [100(x_{2i}^2 - x_{2i+2})^2 + (x_{2i} - 1)^2] \\
& + [100(x_{2i+\delta-1}^2 - x_{2i+\delta+1})^2 + (x_{2i+\delta-1} - 1)^2] \\
& + [100(x_{2i+\delta}^2 - x_{2i+\delta+2})^2 + (x_{2i+\delta} - 1)^2] \\
& + \sum_{i_{ov}}^{n1_{ov}-1} \big([100(x_{i_{ov}}^2 - x_{i_{ov}+1})^2 + (x_{i_{ov}} - 1)^2] \\
& + [100(x_{i_{ov}+\delta}^2 - x_{i_{ov}+\delta+1})^2 + (x_{i_{ov}+\delta} - 1)^2] \big) \\
& + \sum_{j}^{n2} x_j^2 + \sum_{l}^{n3} x_l^2
\end{aligned} \tag{2.20}
$$

where $n1 = 0.1 \times D/2$, $i = 1$, $\delta = 0.6 \times D$, $i_{ov} = 1$, $n1_{ov} = 0.04 \times D - 1$, $j = (\gamma \times D/2) + 1$, $n2 = 0.6 \times D$, $l = 0.7 \times D + 1$, $n3 = D$ and δ is used to control the spreading of the nonseparable variables whenever the dimension changes.

- **Objective 6** (Equation 2.21):

$$
\begin{aligned}
Obj_6 = & \sum_{i}^{n1} \big([100(x_{2i-1}^2 - x_{2i+1})^2 + (x_{2i-1} - 1)^2] \\
& + [100(x_{2i}^2 - x_{2i+2})^2 + (x_{2i} - 1)^2] \\
& + \sum_{j}^{n2} \sum_{ov}^{OV} [100(x_{ov+\tau}^2 - x_{ov+\tau+1})^2 + (x_{ov+\tau} - 1)^2]
\end{aligned} \tag{2.21}
$$

where $i = 1$, $n1 = D/2$, $n2 = 0.1 \times D$, $j = 1$, $\tau = (j-1) \times (0.1 \times D)$ and ov is the number of overlapping variables, which is presented in this objective function as two variables. τ is the parameter that controls the occurrence of the overlapping variables by indicating their index.

The 18 problems of the test suite are formed from three categories. The first category has all six objective functions with one constraint, the second category has two constraints, and the last category has three constraints. The first constraint has groups of separable variables spread throughout the problem dimension. The second constraint has groups of three nonseparable variables that do not contradict the first constraint. The third constraint has groups of spliced nonseparable variables that are carefully designed not to contradict the other two constraints. They are designed as follows:

Constrained functions

- **Constraint 1** (Equation 2.22): The first constraint $g_1(\mathbf{x})$ is a completely separable constraint which contains $0.01 \times D$ groups of 5 separable variables. This constraint is designed so one group exists in the second quarter of any 100 variables. It does this when the variables start in position 26 of each of the 100 variables in the dimension.

$$g_1(\mathbf{x}) = \sum_{i}^{n1} \sum_{j}^{len} x_{j+\phi}^2 \leq 0 \tag{2.22}$$

  where $i = 1$, $n1 = 0.1 \times D$, $len = 5$, $j = 1$, $\phi = 25 + 100(n1 - 1)$, $n1$ is the number of nonseparable groups, and len is the number of nonseparable variables in each group. In addition, ψ is the parameter that controls the spreading of these groups in the dimension.

- **Constraint 2** (Equation 2.23): The second constraint, $g_2(\mathbf{x})$, is a completely nonseparable constraint that contains one, two, and three nonseparable groups of 3 variables each for the dimensions 100, 500, and 1000 respectively.

$$g_2(\mathbf{x}) = \sum_{i}^{l} \sum_{j}^{len-1} \left[100(x_{j+\phi}^2 - x_{j+\phi+1})^2 + (x_{j+\phi} - 1)^2\right] \leq 0 \tag{2.23}$$

where $i = 1$, $len = 3$, $j = 1$, $\phi = 50 + 400(i-1)$, $n1$, l is the number of nonseparable groups at each dimension, which are 1,2, and 3 groups for the 100, 500, and 1000 dimensions, ϕ is the parameter that controls the spreading of these groups in the dimension.

- **Constraint 3** (Equation 2.24): This constraint $g_3(\mathbf{x})$ is harder than the two previous constraints. It contains spliced nonseparable variables represented in the form of pairs, where the first variable in the pair is located in the first half of the dimension, and the second variable in the pair is located in the second half. The distance between the two variables of each pair increases when the dimension is increased. This separation between the two variables in each pair makes it more challenging to identify the interdependent pairs. This type of design is useful for showing the strength of the decomposition techniques in decreasing the interdependency among the subproblems. This is because good techniques will be able to identify the interdependent, nonseparable variables and group in common subproblems.

$$g_3(\mathbf{x}) = \sum_{i}^{k} \left[100(x_{\phi_1}^2 - x_{\phi_2})^2 + (x_{\phi_1} - 1)^2\right] \leq 0 \tag{2.24}$$

where $i = 100$, $\phi_1 = 10 + 6(i-1)$, $\phi_2 = 90 + 400(i-1)$, $k = 1, 2, 3$ for $D = 100, 500, 1000$ respectively, representing the number of nonseparable spliced variables, and ϕ_1, ϕ_2 are the positions of the first and second spliced variables in each nonseparable pair.

In addition, Table 2.2 contains a summarized list of the objectives and constraints characteristics in the column Description, the column Dimension contains the three values tested in this work, and the last column has the number of nonseparable variables for each function or constraint and for each dimension.

Objective	**Description**	**D**	**NsV**
Obj_1	Completely Separable	100 500 1000	 0
Obj_2	0.1×D nonseparable variables divided into groups of length 5	100 500 1000	10 50 100
Obj_3	0.1×D nonseparable variables in the form of separated (not sequential) pair.	100 500 1000	10 50 100
Obj_4	0.2×D sequential nonseparable variables of four groups and each group has $0.05 \times D$ sequential nonseparable variables + $0.02 \times D$ sequential overlapping variables between the nonseparable groups	100 500 1000	20+4 100+20 200+40
Obj_5	0.2×D spliced nonseparable variables of four groups and each group has 0.05×D spliced nonseparable variables + 2 + (0.04×D) sequential overlapping variables	100 500 1000	20+6 100+22 200+42
Obj_6	Spliced nonseparable variables in two groups of $n/2$ variables + 0.2×D overlapping nonseparable variables, every two variables of them consist of a variable from the first nonseparable group and a variable in the second nonseparable group. These overlapping variables are distributed evenly over the dimension N and occur in every *sp* number of variables (*sp*= 10).	100 500 1000	100+20 500+100 1000+200
Constraints	**Description**	**D**	**NsV**
g_1	Separable groups of 5 variables	100 500 1000	 0
g_2	Nonseparable groups of 3 variables.	100 500 1000	3 6 9
g_3	Spliced nonseparable pairs	100 500 1000	2 4 6

Table 2.2: Description of objective functions

Chapter 3

Variable Decomposition

In Chapter 2, we mentioned that most methods used in specialized literature are based on dividing the problem into smaller subproblems. Finding adequate decomposition is the main objective of this work, so in this chapter, we show the formal definition of the problem of decomposition of variables, how it has been addressed by the different methods for LSO, and are described by the type of main approach that motivates decomposition. At the end of this chapter, one of the proposals, a traditional Genetic Algorithm, is shown to optimize decomposition, which is the motivation of our proposal.

3.1 Variable Decomposition Problem

The divide-and-conquer strategy inspires decomposition-based algorithms; therefore, in the first stage of the process, the decision vector must be divided into lower-dimensional vectors. Although this framework sounds simple to implement, it brings several new challenges regarding the division into sub-problems, sub-problems optimization, communication between them, and the merging of their solutions [18].

Given the particular characteristics of the functions to be optimized, this division is not easy to carry out, mainly due to the problem's dimension, the separability or non-separability characteristics, and the type of variables in the decision vector.

We define the terms of separability under minimization below [55].

Definition 3.1 (Separable function). *Given a function $f(\mathbf{x}) : \Omega \rightarrow R$, f is a separable function if and only if:*

$$\arg\min_{x_1,x_2,...,x_D} f(x_1, x_2, ..., x_D) = (\arg\min_{x_1} f(x_1, ...), ..., \arg\min_{x_D} f(..., x_D)) \tag{3.1}$$

where Ω is an D-dimensional decision space. Otherwise, $f(\mathbf{x})$ is called a non-separable problem.

In other words, if it is possible to find the global optimum of a function by optimizing one dimension at a time regardless of the values taken by different sizes, then the function is said to be separable. It is non-separable otherwise.

Definition 3.2 (Fully nonseparable). *A function f(x) is fully nonseparable if every pair of its decision variables interact with each other.*

Definition 3.3 (Partially additively separable function). *A function $f(\mathbf{x}) : \Omega \rightarrow R$ is called partially additively separable if it has the following general form:*

$$f(\mathbf{x}) = \sum_{i=1}^{m} f_i(\mathbf{x_i}),\ m > 1 \tag{3.2}$$

where $f_i((x_i))$ is a non-separable sub-function, and m is the number of non-separable components of f.

That is, we can see the function in terms of sums with vectors of less dimension than the original one.

On the other hand, some optimization problems have overlapping or spliced variables. We call overlapping variables when they have to be in two or more sub-functions. Two interacting variables are known as spliced variables if they do not follow each other in the solution vector but are dependent on each other.

In Section 2.2, we presented some of the methods that have been proposed to solve LSO problems; some of them were classified as decomposition-based methods. These methods decompose the vector of variables of the objective function

into sub-vectors of lower dimensionality to work on them separately. We pose this decomposition as an optimization problem called the Variable Decomposition Problem.

3.2 Solution methods for the variable decomposition problem

According to Ma et al., [19], 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 following classes: static variable grouping, random variable grouping, variable grouping based on interaction learning, variable grouping based on domain knowledge, overlap, hierarchical variable grouping, and finally, some of their hybrids. Figure 3.1 shows this classification and adds an extra element that is the approach addressed in this work.

In the following paragraphs, we describe some of 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 sub-components and fix the variable grouping during optimization.

Among the works that perform static grouping of variables are the works of Potter and De Jong [18] and Liu et al. [4], which show good performance with fully separable problems. For non-separable problems, Van den Bergh and Engelbrecht [57] propose a static sequential decomposition. However, the decomposition process depends on an adequate number of sub-components that must be adequate from the beginning of the strategy, and the static decomposition process has poor performance in many problems.

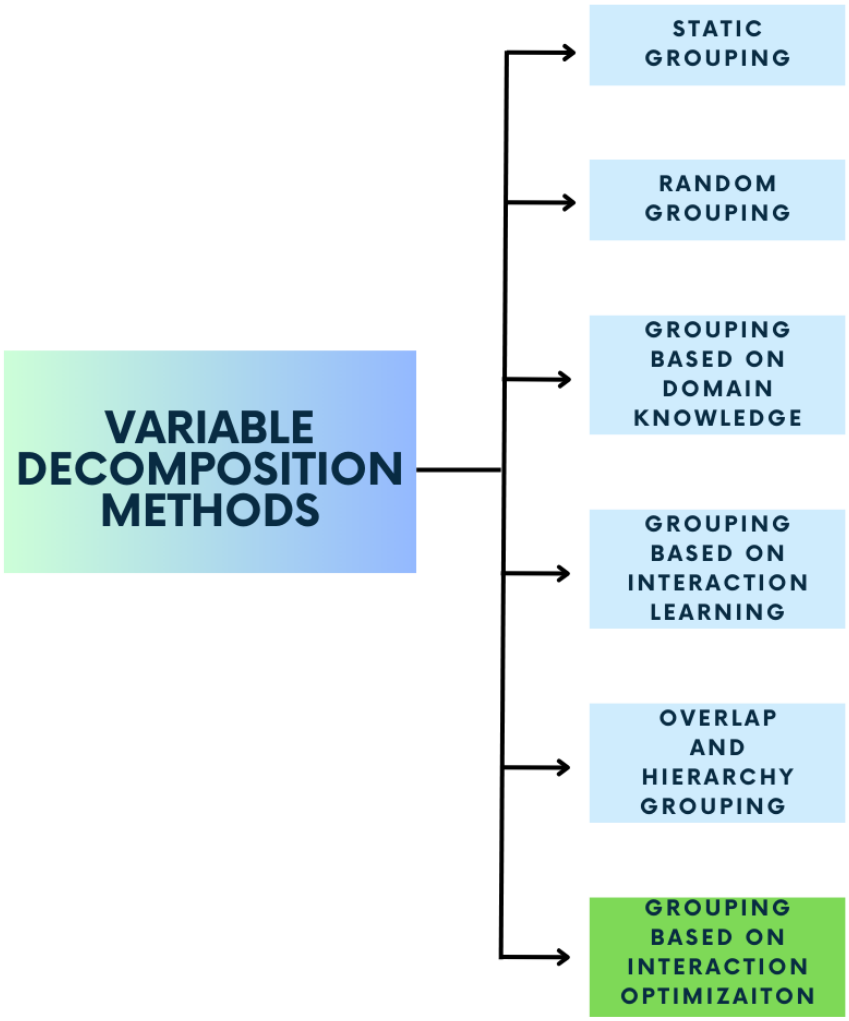


Figure 3.1: A classification of the variable decomposition techniques.

For that reason, several authors proposed *random variable grouping* strategies, such as the case of Yang et al. [58], who present random decomposition with a fixed number of sub-components. Moreover, the same authors propose in [20] a random decomposition with a dynamic number of sub-components. Omidvar et al. [23, 59] 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. [60]. Furthermore, Differential Grouping and Differential Grouping version 2 (DG2) from Odmivar et al. [28, 61], 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 [62, 63, 64, 65].

Moreover, Delta Grouping [59] is classified as a decomposition method based on statistical models, where all variables and the objective functions are considered 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 interactions. 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 [66], 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 [67, 68].

As an example of an approximate model, the fitness evaluation of a Large-Scale Continuous Optimization Problem is converted to evaluating a more straightforward, partially separable problem in [56]. Linkage adaption methods use specially designed evolution operators, representations, and mechanisms to divide variables into groups [69].

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. [70, 71] suggested assigning each variable to several sub-components, each of which contains more than one variable, and the sub-components compete with each other to represent shared variables.

Furthermore, Strasser et al. [72] introduce some overlapping grouping strategies, including random overlapping grouping, neighbor overlapping grouping, centered overlapping grouping, and more.

To address the issue of overlapping problems, Li et al. [73] propose a fuzzy decomposition algorithm that groups the variables according to their interaction degree. In this algorithm, the interaction structure matrix and the interactive degree for an LSO problem are calculated at first according to the interaction among all the decision variables. Then, the number of subgroups is determined based on the interactive degree. Based on the interaction structure matrix, a spectral clustering algorithm is proposed to decompose the decision variables.

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 functions. Guan et al. [74] used the conflicting probability of two flights to learn the variable interaction in solving the flight conflicting avoidance problem, 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. [56]; this new version is known as the Variable Interaction Identification Technique for Constrained Problems (VIIC) [54]. With inequality constraints, this method can find the interaction between variables in problems of 100, 500, and 1000 dimensions.

They 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 a function's separability and partial separability.

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 [75] improved the performance of the VIIC to achieve adequate decomposition for a fixed number of sub-groups. They transformed the constrained problem into an unconstrained problem, using a neighborhood heuristic to guide the search by their proposed decomposition, and then optimized it (called Neighbor Variable Interaction Identification Technique for Constrained Problems (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 called Dynamic Variable Interaction Identification Technique for Constrained Problems (DVIIC) without establishing a fixed number of subcomponents [76].

Later, Vakhnin and Sopov [77] 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 sub-components.

To try to enhance the existing methods for decomposing variables, we propose a Genetic Algorithm with a genetic encoding based on groups, better known as the Grouping Genetic Algorithm, to optimize the variable decomposition. Experimental results demonstrate the benefits of using a group-based encoding scheme for this problem and its advantages over the GA with an integer-based encoding scheme (DVIIC [76]). Also, we adapted DG2 to evaluate the decompositon in the LSO with constraints, in a similar way, the statistical results demonstrated our proposal obtained better values of decomposition and optimization. Both algorithms DG2 and DVIIC are described in the next sections.

3.3 Differential Grouping 2

As part of the experimentation of this work, we compared our proposal with one of the previously mentioned algorithms, Differential Grouping 2, and we describe it in

detail.

The Differential Grouping method in its original version was proposed by Omidvar et al. [23] as a method to identify the interaction between variables. It is a method based on the perturbation of variables that first measures the interaction of variables by employing the change of the objective function value when perturbing two variables with the same value, which was tested in LSO problems.

However, the DG algorithm did not perform very well when it was tested on the CEC'2013 test set of functions [55] since DG had some deficiencies such as (1) high computational cost in separable functions, (2) inability to detect the interaction of variables in problems with overlapping variables, (3) sensitivity to computational rounding errors and (4) the algorithm requires as input a threshold parameter ϵ.

Thus, the authors propose improvements to the algorithm in 2017, called DG2 [61]. In DG2, the number of evaluations required by the algorithm is reduced by half, and the parameter ϵ is calculated automatically, making DG2 a parameter-free algorithm to tune. This algorithm is assisted by Definition 3.4 to group variables.

Definition 3.4 (Interaction DG2)**.** *Let $f(\mathbf{x})$ be a additively separable function, $\forall a, b_1 \neq b_2, \delta \neq 0 \in \mathbb{R}$, x_p y x_q interact if:*

$$\Delta_{\delta,x_p}[f](\mathbf{x})|_{x_p=a,x_q=b_1} \neq \Delta_{\delta,x_p}[f](\mathbf{x})|_{x_p=a,x_q=b_2} \tag{3.3}$$

where $\Delta_{\delta,x_p}[f](\mathbf{x}) = f(...,x_p+\delta,...) - f(...,x_p,...)$ refers to the positive difference f in the interval δ.

The algorithm works roughly as follows: for each pair of variables the interaction between them is calculated, which is created and carried out in a matrix of size $n \times n$ denoted by Δ through the ISM function where each entry is the interaction value between the variables.

In addition there is a second matrix of the same size denoted Θ which is calculated through the DSM function, where the corresponding entry will be 1 if the variables present a value in the ISM matrix greater than a threshold ϵ, and will be

0 otherwise. Finding the 1s in the matrix Δ detects the groups of related variables. Algorithm 4 presents the general process of DG2.

Algorithm 4: $(g, \mathbf{x}_1, ..., \mathbf{x}_g, \mathbf{x}_{sep}, \Gamma) = \text{DG2}(f, b, \overline{\mathbf{x}}, \underline{\mathbf{x}})$

```
(Λ, F, f̌, f_base, Γ) = ISM(f, n, x̄, x);
Θ = DSM(Λ, F, f̌, f_base, n);
(k, y_1, ..., y_k) = ConnComp(Θ);
x_sep = {}, g = 0;
for i = 1 → k do
    if |y_i| = 1 then
        x_sep = x_sep ∪ y_i;
    else
        g = g + 1, x_g = y_i;
    end
end
```

As can be seen, the DG2 algorithm uses an ISM function that detects the interaction between each pair of variables.

Following definition 3.4 two variables x_1 and x_2 interact if satisface Equation 3.3 being x_p and x_q respectively.

Algorithm 5 shows the steps to calculate this ISM value. For brevity, the right-hand side of Equation 3.3 is denoted by $\Delta^{(1)}$ and the left-hand side by $\Delta^{(2)}$.

Additionally, Algorithm 6 shows the process of filling the matrix Θ, which will consist of zeros and ones depending on the independent threshold for each pair of variables according to [61].

Algorithm 5: $(\Lambda, \mathbf{F}, \check{f}, f_{base}, \Gamma) = \mathrm{ISM}(f, n, \overline{\mathbf{x}}, \underline{\mathbf{x}})$

$\Lambda = \mathbf{0}_{n\times n}$;
$\mathbf{F}_{n\times n} = \mathrm{NaN}_{n\times n}$;
$\check{f}_{n\times 1} = \mathrm{NaN}_{n\times 1}$;
$\mathbf{x}^{(1)} = \underline{\mathbf{x}}$, $f_{base} = f(\mathbf{x}^{(1)})$, $\Gamma = 1$;
$\mathbf{m} = \frac{1}{2}(\overline{\mathbf{x}} + \underline{\mathbf{x}})$;
for $i = 1 \rightarrow n-1$ **do**
  if $\neg is\ NaN(\check{f}_i)$ **then**
    $\mathbf{x}^{(2)} = \mathbf{x}^{(1)}, \mathbf{x}_i^{(2)} = \mathbf{m}_i$;
    $\check{f}_i = f(\mathbf{x}^{(2)}), \Gamma = \Gamma + 1$;
    for $j = i+1 \rightarrow n$ **do**
      if $\neg is\ nan(\check{f}_i)$ **then**
        $\mathbf{x}^{(3)} = \mathbf{x}^{(1)}, \mathbf{x}_i^{(3)} = \mathbf{m}_i$;
        $\check{f}_i = f(\mathbf{x}^{(3)}), \Gamma = \Gamma + 1$;
      end
      $\mathbf{x}^{(4)} = \mathbf{x}^{(1)}, \mathbf{x}_i^{(4)} = \mathbf{m}_i, \mathbf{x}_j^{(4)} = \mathbf{m}_j$;
      $\mathbf{F}_{ij} = f(\mathbf{x}^{(4)}), \Gamma = \Gamma + 1$;
      $\Delta^{(1)} = \check{f}_i - f(\mathbf{x}^{(1)})$;
      $\Delta^{(2)} = \mathbf{F}_{ij} - \check{f}_j$;
      $\Lambda_{ij} = |\Delta^{(1)} - \Delta^{(2)}|$
    end
  end
end

Algorithm 6: $\Theta = \mathrm{DSM}(\Lambda, \mathbf{F}, \check{f}, f_{base}, n)$

$\Theta = \mathrm{NaN}_{n\times n}$;
$\eta_1 = \eta_2 = 0$;
for $i = 1 \to n-1$ **do**
  for $j = i+1 \to n$ **do**
    $f_{max} = \max\{f_{base}, \mathbf{F}_{ij}, \check{f}_i, \check{f}_j\}$;
    $e_{inf} = \gamma_2 \cdot \max\{f_{base} + \mathbf{F}_{ij}, \check{f}_i + \check{f}_j\}$;
    $e_{sup} = \gamma\sqrt{n} \cdot f_{max}$;
    if $\Lambda_{ij} < e_{inf}$ **then**
      $\Theta_{ij} = 0; \eta_0 = \eta_0 + 1$;
    end
    else if $\Lambda_{ij} > e_{sup}$ **then**
      $\Theta_{ij} = 1; \eta_1 = \eta_1 + 1$;
    end
  end
end
for $i = 1 \to n-1$ **do**
  for $j = i+1 \to n$ **do**
    $f_{max} = \max\{f_{base}, \mathbf{F}_{ij}, \check{f}_i, \check{f}_j\}$;
    $e_{inf} = \gamma_2 \cdot \max\{f_{base} + \mathbf{F}_{ij}, \check{f}_i + \check{f}_j\}$;
    $e_{sup} = \gamma\sqrt{n} \cdot f_{max}$;
    if $\Theta_{ij} \neq NaN$ **then**
      $\epsilon = \frac{\eta_0}{\eta_0+\eta_1} \cdot e_{inf} + \frac{\eta_1}{\eta_0+\eta_1} \cdot e_{sup}$;
      if $\Lambda_{ij} > \epsilon$ **then**
        $\Theta_{ij} = 1$;
      else
        $\Theta_{ij} = 0$;
      end
    end
  end
end

3.4 Dynamic Variable Interaction Identification Technique for Constrained Problems

DVIIC was developed for Aguilar-Justo et al. [76] to search for a good variable arrangement and suitable sub-groups. The basic steps of a traditional GA are: 1) evaluate the population, 2) select the parents for reproduction, 4) crossover those selected parents in step 2, 5) mutate the offspring, and 6) replace the current population with the new generation. These steps are repeated until reaching a stop criterion.

The details of the proposed algorithm, step by step, are described below.

3.4.0.1 Decomposition Evaluation

To evaluate an individual, they use the difference proposed by [54], where the $grps_{diff}$ value of Equation 4.5 is minimized. However, to maximize the number of sub-problems, they updated the $grps_{diff}$ as follows: 1) If the number of sub-groups is one, then the $grps_{diff}$ takes an extremely greater value. 2) If the $grps_{diff}$ is zero, the decomposition is perfect, and it is rewarded by subtracting the number of sub-groups of the individual. 3) In another case, the $grps_{diff}$ value does not change. (see Equation 3.4).

$$grps_{diff} = \begin{cases} infinite & \text{if } m = 1 \\ grps_{diff} - m & \text{if } grps_{diff} = 0 \\ grps_{diff} & \text{otherwise} \end{cases} \tag{3.4}$$

3.4.0.2 Genetic Encoding

The position represents the variable number, while the value in the chromosome represents the label of the group to which the variable belongs (genotypic representation of Figure 3.2a). However, to evaluate the individual, it is necessary to transform the chromosome into a phenotypic representation (Figure 3.2b). This

representation comprises the variables arrangement S_n and the variable number by group V. The length of V is the total number of sub-groups.

$$ind = [1, 1, 1, 2, 1, 2, 2, 2, 3, 3]$$

(a) Genotypic integer representation

$$S_n = [1, 2, 3, 5, 4, 6, 7, 8, 9, 10] \; V = [4, 4, 2]$$

(b) Phenotypic integer representation

Figure 3.2: Genotypic and Phenotypic representation for integer representation

3.4.0.3 Selection and crossover

Every individual is subject to crossover. All individuals could be recombined to breed a new generation. Given a crossover probability, the parents will be recombined by a two-point crossover. Figure 3.3 shows an example of such a variation operator.

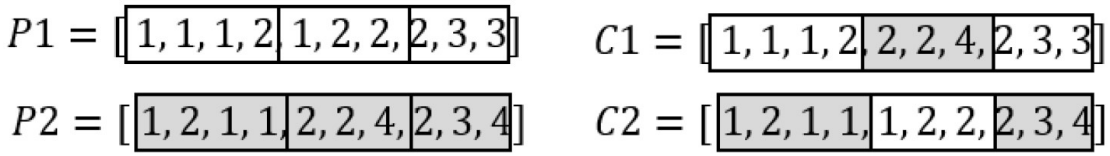


Figure 3.3: Example of two-point crossover operator for integer representation.

3.4.0.4 Mutation

In their work, Justo et al. used a uniform mutation. Each chromosome is likely to mutate, changing its label to a random label that lies between one of the dimensions of the problem.

$$ind = [1, 1, \mathbf{1}, 2, 1, \mathbf{2}, \mathbf{2}, 2, 3, 3]$$
$$mut = [1, 1, \mathbf{5}, 2, 1, \mathbf{3}, \mathbf{5}, 2, 3, 3]$$

Figure 3.4: Example of uniform mutation operator for integer representation.

3.4.0.5 Solution repair

After crossover or mutation operators, the individual could have many different labels. However, to maintain the representation clear and straightforward, it is necessary to repair the solution.

The repair of this proposal consists of re-labeling the solution to keep the labels ascending. In Figure 3.4, the mutated individual has the labels $1, 2, 3, 5$, which means that this solution represents four sub-groups, but the maximum label is five. Therefore, the repaired solution is $mut = [1, 1, 4, 2, 1, 3, 4, 2, 3, 3]$.

Another repair method consists of joining the groups of separable variables. That is, let a solution of 5 variables with the following decomposition $ind = [1, 2, 3, 4, 4]$, where there are four groups, but three of them have a single variable. Therefore, those sub-groups are merged, and $ind_{repair} = [1, 1, 1, 2, 2]$.

3.4.0.6 Replacement

The replacement is generational with elitism, i.e., always the best individual passes to the next generation, replacing a random individual in the new population.
The Genetic algorithm described above showed very efficient results when used to decompose variables in large-scale optimization problems, so the genetic idea is the motivation of this work, proposing an alternative based on groups to optimize the generation of sub-problems.

Chapter 4

Grouping Genetic Algorithm for Variable Decomposition

In this chapter, we present the main proposal of this work, addressing the decomposition of variables from the point of view of discrete optimization. It is intended to find the decomposition of the decision vector in minor dimensionality vectors, which contain the variables that interact with each other and where there is no interaction between the different sub-vectors.

First, we show the analogy of the problem of decomposition of variables with other discrete optimization problems known as grouping problems. In addition, some of the most popular methods to solve these problems are listed, highlighting the use of Grouping Genetic Algorithms, which are Genetic Algorithms focused on groups; therefore, their operators work in different ways than usual. Then, given the above, we propose the design of each component of our Genetic Algorithm based on groups and show its operation through examples.

4.1 Grouping problems in combinatorial optimization

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 $X = x_1, x_2, ..., x_D$ 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 [78], 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. [79] organized the grouping problems into three categories, using as a criterion 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 non-order-dependent class. Thus, we can say that the decomposition problem is a variable, identical, and not order-dependent grouping problem.

Grouping problems are challenging to solve. Most are NP-hard, implying no algorithm can find an optimal solution for every instance in polynomial time [80]. There are several NP-hard grouping problems, such as the Bin Packing Problem, Job Shop Scheduling Problem, etc. Ramos-Figueroa et al. [78] studied the strategies that work with most problems like the ones mentioned before.

4.2 Solution methods for grouping problems

In the work of Ramos-Figueroa et al. [78], a literature review examines the different metaheuristics used in the state-of-the-art to solve grouping problems.

Among the approaches they studied are neighborhood search algorithms, also called trajectory searches. This strategy was designed to generate and explore the neighbors of a current solution. Neighbor solutions are created using different techniques, such as modifying the value of one variable or exchanging two or more elements. Hence, the neighborhood size depends on the strategy used to generate the neighbors.

Among the algorithms that stand out for solving this type of problem are Hill Climbing (HC) [81], Variable Neighborhood Search (VNS) [82], Simulated Annealing (SA) [83], and Tabu Search (TS) [84].

Hill Climbing is an iterative improvement search strategy in which, at each iteration, the neighborhood of the current solution is used to improve the fitness of the solutions. Similarly, Variable Neighborhood Search is also an iterative improvement algorithm, with the difference that this method explores increasingly distant neighborhoods one at a time.

On the other hand, Simulated Annealing is inspired by the chemical process of metal annealing. In this algorithm, if a new solution in the neighborhood is better than the current solution, it will always replace it. If the neighbors of the

current solution are worse than the current solution, they are evaluated to decide probabilistically if a transition to a new solution is made.

And finally, Tabu Search is an algorithm that uses the concept of memory and implements it through simple structures. TS works as an ordinary descent method that only permits moving to neighbors that improve the quality of the current solution. However, a short-term memory, known as the tabu list, stores recently visited solutions (or their attributes) to expand the local search and escape from local optimums. Thus, the neighborhood of the current solution is restricted, considering the solutions that do not belong to the tabu list.

Another strategy mentioned in this review was the use of swarm intelligence algorithms. These algorithms emulate the collective self-organized behavior of natural systems in solving problems. A population of simple agents works together in swarm intelligence to optimize problems.

Different algorithms have been proposed with this inspiration. Some of them have been used to solve several grouping problems and are very popular in the scientific community, such as Ant Colony Optimization (ACO) [85], PSO [86], and Artificial Bee Colony (ABC) [87].

Ant Colony Optimization was designed to solve discrete problems. ACO encodes a given combinatorial optimization problem instance as a fully connected graph whose nodes are components of solutions and edges are connections between components.

A pheromone variable is associated with each element, which can be read and modified by each ant. At each iteration of the algorithm, several artificial ants are considered. These ants build a solution to the problem, step by step, adding a feasible solution component according to a stochastic mechanism biased by the pheromone and heuristic information about the problem. Next, the pheromone values are updated to make solution components of good solutions more desirable for ants in future iterations.

Second, PSO is a metaheuristic initially planned to deal with continuous domains but has also been adapted to tackle discrete problems. This metaheuristic

was designed using as inspiration the swarming and flocking behaviors in animals. PSO begins generating a population of random particles; each is a candidate solution to the problem. It is defined by three vectors in the D-dimensional search space: a velocity vector, a position vector, and a memory vector, which helps it remember its fittest known position discovered so far.

At each iteration, directed transformations move each particle in the search space in response to discoveries obtained from the environment, allowing solutions to be improved as the algorithm iterates.

The last one, Artificial Bee Colony, was inspired by the intelligent foraging behavior of honey bees. In ABC, a colony of artificial forager bees (agents) searches for rich artificial food sources (good solutions for a given problem). To apply ABC, the considered optimization problem is first converted to the problem of finding the best parameter vector, which minimizes an objective function. Then, the artificial bees randomly discover a population of initial solution vectors to iteratively improve them by moving towards better solutions using a neighbor search mechanism while abandoning poor solutions.

Finally, according to Ramos-Figueroa et al., Evolutionary Algorithms mentioned in Section 2.2.1 are among the most popular algorithms for solving grouping problems.

GAs have been developed and applied to solve a wide range of grouping problems, including GAs with different schemes for representing solutions, selection, crossover, replacement, mutation, etc. This metaheuristic has shown promising results in solving grouping problems because it can easily incorporate new general or specific ideas. For the above reasons, we propose a Grouping Genetic Algorithm (GGA) to solve the variable decomposition problem.

The GGA was designed in 1992 by Falkenauer [88] 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. [89] remark that the encoding of a grouping problem solution into a chromosome is a vital 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 various types of encoding and their advantages in solving grouping problems.

The state-of-the-art [78] 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 [90].

In this work, we propose the first GGA for the variable decomposition problem in Large-Scale Constrained Optimization Problems to the author's knowledge. A comparative study is conducted to evaluate the performance of our Grouping Genetic Algorithm versus the Genetic Algorithm DVIIC [76] on the decomposition of variables for Large-Scale Constrained Optimization Problems.

We implement similar operators and equivalent parameter settings to promote a fair comparison. The experiments were carried out using 18 test functions, each with 100, 500, and 1000 variables. The results validate the advantages of group-based encoding over integer-based encoding.

4.3 Grouping Genetic Algorithm for Variable Decomposition

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 Grouping Genetic Algorithm that we called GGA-VD [2] with operators and parameters with similar features to the Genetic Algorithm DVIIC (proposed by Aguilar-Justo et al. [76]) 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. [76] includes an integer-based representation, where a chromosome has a fixed length equal to the number of variables. Each gene represents a variable and indicates the group where the variable is set.

On the other hand, our GGA-VD includes a group-based representation, where a chromosome can have a variable length equal to the number of sub-components, and each gene represents a sub-component and indicates the variables that belong to this subset.

In Algorithm 7, we show the general steps of the GGA-VD proposed in this work. The precise details are shown in the following subsections.

Algorithm 7: Grouping Genetic Algorithm for Variable Decomposition algorithm (GGA-VD).

```
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
```

The process begins by generating an initial population P of pop_size individuals created by the population initialization strategy (Line 1). After that, each individual 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 re-evaluated to update the population and the best global solution found so far (Lines 8–9).

The following subsections detail the components and operators of our GGA-VD.

4.3.1 Genetic Encoding

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

The groups of variables represent each individual in the population. Figure 4.1 shows an example of an individual that represents a problem with ten variables numbered from 0 to 9 randomly assigned to four sub-components 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 sub-component is variable and can be between 1 and D; the number of sub-components m is between 1 and D.

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

Figure 4.1: Example of a group-based chromosome.

4.3.2 Decomposition Evaluation

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

Sayed et al. [54] proposed a decomposition evaluation inspired by the definitions of problem separability [91] and partial separability [92].

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 sub-problems 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 sub-problems, $f(x_i)$, 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 sub-problems, where the summation of all sub-problems 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 sub-problems and V is the number of variables in the k-th sub-problem.

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 8 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 4.1 and 4.2, 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 $fit_{all_{c_1 c_2}}$, and then we initialize $fit_{grps_{c_1 c_2}}$ as 0 (Lines 2–3). Afterward, 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. In contrast, the remaining variables take the value c_2 to evaluate this arrangement and obtain $fit_{grps_{k,c_1}}$.

Algorithm 8: 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 4.1 and 4.2
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 4.3
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 4.4
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 $grps_{diff} = |fit_{all_{c_1c_2}} - fit_{grps_{c_1c_2}}|$

$$fit_{all_{c_1}} = Obj(\mathbf{x}) + cvs(\mathbf{x}),\ x_i = c_1,\ \forall i \in [1, D] \tag{4.1}$$

$$fit_{all_{c_2}} = Obj(\mathbf{x}) + cvs(\mathbf{x}),\ x_i = c_2,\ \forall i \in [1, D] \tag{4.2}$$

$$\mathbf{x}_{k,c_1} = \begin{cases} c_1 & \forall x_i \in grps_k \\ c_2 & \text{otherwise} \end{cases} \tag{4.3}$$

$$\mathbf{x}_{k,c_2} = \begin{cases} c_2 & \forall x_i \in grps_k \\ c_1 & \text{otherwise} \end{cases} \tag{4.4}$$

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 4.3 and 4.4.

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.

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 4.1 and 4.2,

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

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

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 an arrangement $\mathbf{x}_{k,c_1}$ according to Equation 4.3 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 problem decomposition aims to create the best decomposition; that is, to create independent sub-components and minimize the difference $grps_{diff}$ shown in Equation 3.4.

Therefore, we have adopted a similar evaluation to the one proposed by Aguilar-Justo et al. [76], which is defined next: to maximize the number of sub-problems, 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 using the previous evaluation function benefits a decomposition with a high number of sub-components, in some cases, a complete decomposition (as many

groups as numbers of variables) would be presented as optimal when it is not the case. For this reason, a modification in the evaluation function is necessary (avoiding the benefit to many groups).

Therefore, our algorithm has modified the evaluation function. This change is summarized in Equation 4.5.

$$grps_{diff} = \begin{cases} infinite & \text{if } m = 1 \\ grps_{diff} & \text{otherwise} \end{cases} \tag{4.5}$$

4.3.3 Population Initialization

The initial population in most GGAs is generally generated by obtaining random partitions of the elements to group. In our GGA-VD, 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 sub-components 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 [76], 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).

4.3.4 Grouping Crossover Operator

After choosing the individuals 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 analogous to the one used in the Genetic Algorithm DVIIC [76].

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). These elements 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.

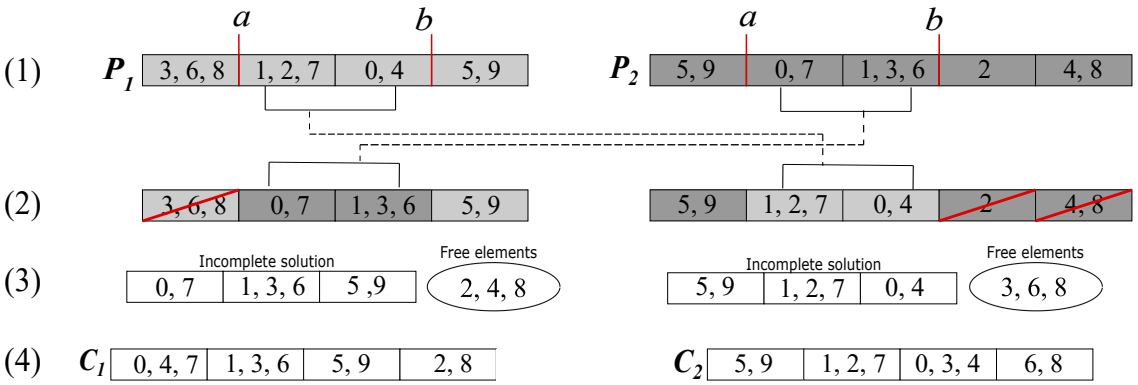


Figure 4.2: Example of two-point crossover operator for group-based representation.

In Figure 4.2, we can see an example of crossover for two individuals with ten 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 eight 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.

4.3.5 Grouping Mutation Operator

The mutation operator used in the Genetic Algorithm DVIIC [76] 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, we have implemented the group-oriented elimination mutation operator so that GGAs can take a similar operator.

This operator works by eliminating a random group of individuals. 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 4.3 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 abovementioned strategy.

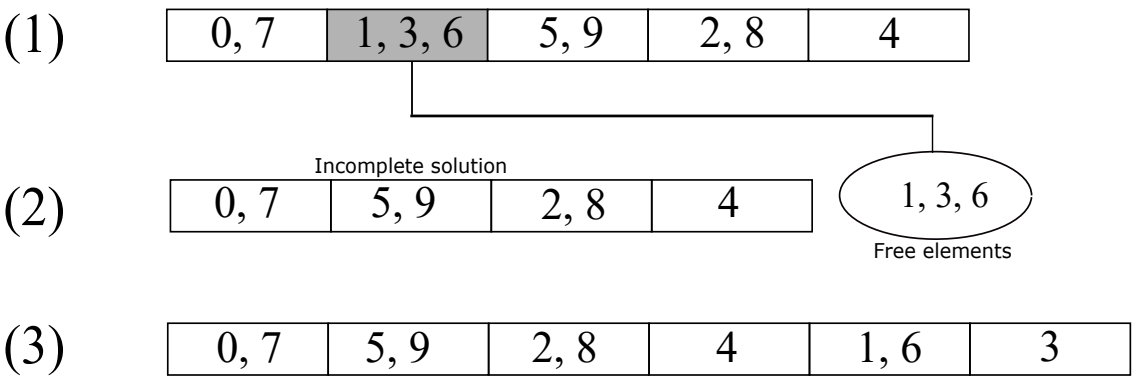


Figure 4.3: Example of elimination mutation operator for group-based representation.

4.3.6 Selection and Replacement Strategies

In a Genetic Algorithm, we must select the population members that will be candidates for crossover and mutation. A selection scheme decides which individuals can pass their genes to the next generation 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 [93, 94].

Several strategies have been proposed for the parent selection (individuals for crossover). In our GGA-VD, we use a selection scheme similar to that included in the Genetic Algorithm DVIIC [76], 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. 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 [76]. Given a mutation probability p_m, for each individual in the population, a random number between 0 and 1 is generated. The individual will be mutated when this number is less or equal than p_m.

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) [95].

Similar to the strategy of the Genetic Algorithm DVIIC [76], in our GGA-VD, the offspring replace the parents after crossover. After the mutation, the mutated individuals replace the original ones. Elitism is adopted to maintain the best solution for the population, replacing the worst individual in the new population.

4.4 Improvement of the GGA-VD

As part of the GGA-VD performance study for the variable decomposition, some of its components have been modified to analyze if there are no improvements in the decomposition results.

4.4.1 Selection and replacement strategies

In the first place, we tested three strategies to select the parents for the crossover process and two methods to reinsert the offspring into the population, which were proposed to solve other grouping problems. Furthermore, along with these selection strategies, we tested two different replacement methods for the population with the offspring.

On the other hand, the crossover and mutation operators used in the original version of the GGA-VD were improved by incorporating a heuristic strategy to reinsert the free elements in each offspring individual to have correct and complete solutions. These methods are described below.

4.4.1.1 Roulette selection

Holland proposed this technique [96] and has been the most commonly used method since the origins of Genetic Algorithms.

The Roulette algorithm (according to DeJong [97]) is as follows: First, calculate the sum of expected values $T = \sum_i Ve_i$ where Ve_i is the fitness of the i-th individual divided by the average fitness of the population ($Ve = \frac{f_i}{\bar{f}}$). Then, repeat n times (n is the number of individuals to cross):

1. Generate a random number r between 0.0 and T.
2. Cycle through the individuals in the population, adding the expected values until the sum is greater than or equal to r.
3. The individual who makes this sum exceeds the limit is selected.

4.4.1.2 Tournament selection

This technique was proposed by Wetzel [98] in 1983. The basic idea of the method is to select parents based on direct comparisons of individuals. The algorithm of the deterministic version is as follows:

1. Shuffle the individuals in the population.
2. Choose p individuals (typically $p = 2$).
3. Compare them based on their fitness.
4. The winner of the tournament is the fittest individual.
5. The population must be shuffled a total of p times to select n parents.

4.4.1.3 Controlled selection

Controlled selection was proposed for solving the Bin Packing Problem by Quiroz-Castellanos et al. [99].

In this technique, n_c parents are selected to generate n_c children. The sets of parents G and R are generated, with good and random individuals, respectively; these elements will be pairwise combined, crossing G_i with R_i ($0 \leq i \leq c/2$). Set G includes $n_c/2$ individuals taken at random with a uniform probability from the best n_c individuals of P according to their fitness, and set R includes $n_c/2$ individuals chosen at random from the population (without taking into account the best individual) with a uniform probability.

4.4.1.4 Controlled replacement

Like controlled selection, this technique was proposed by Quiroz-Castellanos et al. [99] and works as follows.

After applying controlled selection, we have a set of high-quality solutions G as the first parent and a set of random solutions R as the second. The crossover

operator generates a set C of n_c children. The first $n_c/2$ children are introduced to the population P and replace the individuals in the set of random parents R. The other $n_c/2$ children are introduced in $P \setminus R$ with the following rule: if children have not been inserted into P, then introduce them, replacing the worst solutions.

4.4.2 Improvement of crossover and mutation operators

One of the details observed in the first test of GGA-VD was that with the original crossover and mutation operator, the final solutions obtained by the algorithm had several groups of size one.

This characteristic could be attributed to reinserting elements in a group of the solution or individual at the end of the process of the operators; since they were added to a new empty group, this did not allow the exploitation of the search space by creating individuals utterly different from the original.

This section shows the modifications made to the crossover and mutation operators. In both operators, free elements must be reinserted in the individual. This reinsertion considers the previously created group; the free elements can be added to one of these groups or a new group.

The operator parameters in the new experiments were the same as in the original experiment, i.e., 90 percent crossover and 10 percent mutation rates.

4.4.2.1 Crossover

Figure 4.4 shows how the crossover operator works in a short example; the free elements are randomly inserted in the existing groups, and even a new group is created with one of these elements.

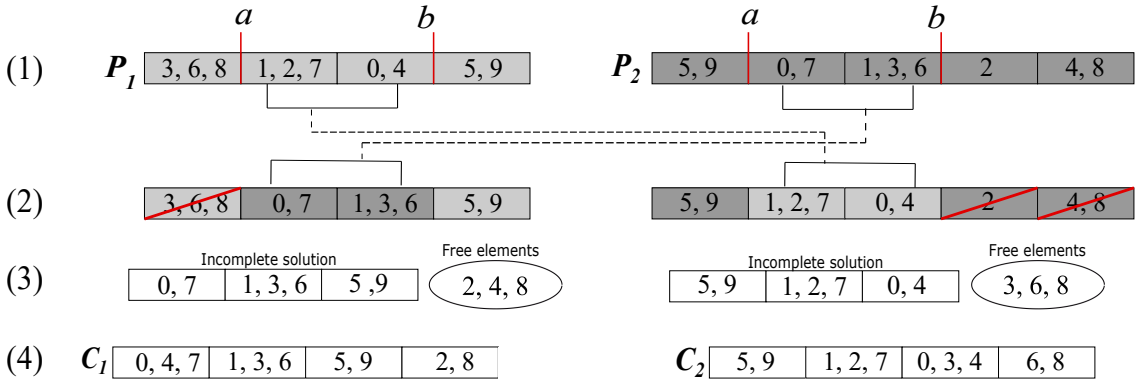


Figure 4.4: Example of the improved crossover operator

4.4.2.2 Mutation

Similarly, in Figure 4.5, we can see how the mutation operator works and how the free elements are inserted in the individual through the same reinsertion strategy mentioned in the crossover.

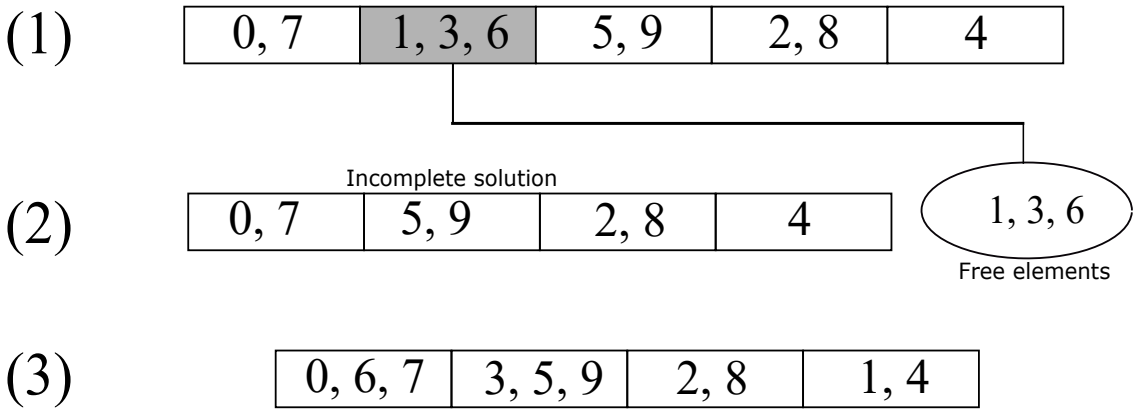


Figure 4.5: Example of the improved mutation operator

Chapter 5

Experiments and results

This chapter contains experiments to compare and improve the initial Grouping Genetic Algorithm.

First, we calculate the statistical results of the GGA-VD against the popular Differential Grouping 2, as well as against the Genetic Algorithm proposed for decomposition that is based on traditional operators of the evolutionary computation; this is to measure the efficiency of our proposal and to show statistically and mathematically the advantages of the proposed algorithm.

Then, once it is shown that the algorithm is efficient compared to the other proposals, its disadvantages and areas of opportunity are studied. Then, the complete optimization process results are shown based on the decomposition carried out by the GGA-VD to optimize the LSO problems.

Finally, a first characterization analysis is carried out to find those functions in which the algorithm presented more significant difficulties and the possible relationships between the $grps_{diff}$ obtained by Equation 4.5 in the decomposition algorithm and the values of the final optimization of the problems.

5.1 Stage 1: Grouping Genetic Algorithm for Variable Decomposition

To measure the performance of the GGA-VD proposal, we decided to compare it first against Differential Groupin version 2 and second against the DVIIC algorithm, which was designed with the same approach as the GGA-VD but that we consider that its performance could be improved by changing the representation of a Genetic Algorithm.

5.1.1 Experiment 1. Empirical Evaluation between GGA-VD and DG2

As we said in Section 3.2, DG2 creates a matrix with the interdependence measures between the variables; once the interactions are obtained, the subproblems that DG2 detects can be obtained. Our evaluation function $Gprs_{diff}$ measured this decomposition.

In this section, the results of this function are shown for the 18 benchmark functions in the three dimensions 100, 500, and 1000, with the statistics of the best evaluation; the median and the standard deviation were 25 independent runs.

Furthermore, each table shows each function's Wilcoxon Rank Sum test at a 95% confidence level (column W); a checkmark (✓) means significant differences in favor of the GGA; an equality symbol (=) represents no significant differences between both algorithms.

The DG2 algorithm is parameter-free, and ϵ is self-adaptive according to [61]. The GGA-VD parameters are described below:

- 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.

These experiments were conducted on an Intel(R) Core(TM) i5 CPU with 2.50 GHz, Python 3.4, and Microsoft Windows 10.

In Table 5.1, we show the results of the GGA-VD against DG2 with the functions evaluated in 100D. In all cases, the evaluation of the GGA-VD is better with a relatively high difference, even in the first three functions that contain the completely separable part.

Table 5.1: GGA-VD and DG2 statistical results in dimension 100. Best results are shown in boldface

Algorithm	**GGA-VD**			**DG2**			
Function/Statistic	**Best**	**Median**	**Std**	**Best**	**Median**	**Std**	**W**
F1	**0.00E+00**	**0.00E+00**	**0.00E+00**	1.32E+02	2.55E+02	8.73E+01	✓
F2	**0.00E+00**	**0.00E+00**	**0.00E+00**	3.45E+02	3.79E+02	3.52E+01	✓
F3	**0.00E+00**	**0.00E+00**	**0.00E+00**	3.56E+02	1.02E+03	2.48E+02	✓
F4	**0.00E+00**	**0.00E+00**	**0.00E+00**	8.82E+02	1.32E+03	2.81E+02	✓
F5	**0.00E+00**	**0.00E+00**	**0.00E+00**	1.59E+02	7.21E+02	3.61E+02	✓
F6	**0.00E+00**	**0.00E+00**	**0.00E+00**	1.30E+03	2.34E+03	7.01E+02	✓
F7	**0.00E+00**	**0.00E+00**	**0.00E+00**	5.57E+02	2.63E+03	5.62E+02	✓
F8	**0.00E+00**	**0.00E+00**	**0.00E+00**	3.59E+03	4.09E+03	6.91E+02	✓
F9	**0.00E+00**	**0.00E+00**	**0.00E+00**	3.75E+03	5.74E+03	1.02E+03	✓
F10	**0.00E+00**	**0.00E+00**	**0.00E+00**	7.38E+03	9.35E+03	2.31E+03	✓
F11	**0.00E+00**	**0.00E+00**	**0.00E+00**	3.27E+04	5.44E+04	3.83E+03	✓
F12	**0.00E+00**	**0.00E+00**	**0.00E+00**	2.84E+04	4.64E+05	1.03E+04	✓
F13	**0.00E+00**	**0.00E+00**	**2.31E+04**	2.44E+05	3.27E+05	9.79E+04	✓
F14	**0.00E+00**	**0.00E+00**	**5.35E+02**	4.86E+05	9.38E+05	2.36E+05	✓
F15	**4.66E-10**	**4.66E-10**	**2.60E+02**	3.65E+05	1.28E+06	2.84E+05	✓
F16	**1.79E+04**	**5.38E+04**	**1.72E+04**	2.80E+03	4.22E+02	2.11E+02	✓
F17	**1.07E+05**	**4.28E+05**	**9.52E+04**	2.73E+07	1.43E+08	2.79E+06	✓
F18	**1.63E+05**	**4.88E+05**	**1.40E+05**	5.43E+05	5.31E+05	3.82E+05	✓

On the other hand, in Table 5.2, we have the statistics already mentioned for the functions evaluated with 500 variables; again, the GGA-VD surpasses the per-

formance of DG2. DG2 has a relatively small evaluation value only in the first function, but the evaluations practically increased as the complexity of the function increased.

Table 5.2: GGA-VD and DG2 statistical results in dimension 500. Best results are shown in boldface

Algorithm	**GGA-VD**			**DG2**			
Function/Statistic	**BEST**	**MEDIAN**	**STD**	**BEST**	**MEDIAN**	**STD**	**W**
F1	**0.00E+00**	**0.00E+00**	**0.00E+00**	2.30E-01	1.91E+02	8.44E+00	✓
F2	**0.00E+00**	**0.00E+00**	**0.00E+00**	3.45E+03	4.63E+03	3.56E+02	✓
F3	**0.00E+00**	**0.00E+00**	**0.00E+00**	2.84E+03	5.46E+03	1.02E+03	✓
F4	**0.00E+00**	**0.00E+00**	**0.00E+00**	9.36E+02	2.84E+03	1.29E+03	✓
F5	**0.00E+00**	**0.00E+00**	**0.00E+00**	3.24E+03	6.44E+03	2.13E+03	✓
F6	**0.00E+00**	**0.00E+00**	**0.00E+00**	1.22E+04	2.99E+04	1.22E+04	✓
F7	**0.00E+00**	**0.00E+00**	**0.00E+00**	1.26E+04	2.37E+04	6.67E+03	✓
F8	**0.00E+00**	**0.00E+00**	**0.00E+00**	3.82E+04	5.33E+05	5.53E+04	✓
F9	**0.00E+00**	**0.00E+00**	**0.00E+00**	3.25E+04	8.63E+04	2.56E+04	✓
F10	**0.00E+00**	**0.00E+00**	**0.00E+00**	2.39E+05	9.94E+05	1.24E+04	✓
F11	**0.00E+00**	**0.00E+00**	**0.00E+00**	4.21E+04	5.56E+04	2.31E+04	✓
F12	**0.00E+00**	**0.00E+00**	**0.00E+00**	2.41E+05	8.01E+05	3.22E+05	✓
F13	**0.00E+00**	**0.00E+00**	**2.56E+04**	1.10E+05	1.32E+05	1.74E+04	✓
F14	**0.00E+00**	**0.00E+00**	**6.86E+04**	8.77E+05	7.77E+06	2.32E+05	✓
F15	**0.00E+00**	**5.96E-08**	**1.27E+05**	1.45E+06	3.31E+06	1.27E+06	✓
F16	**8.13E+02**	**3.25E+03**	**7.04E+02**	9.44E+06	5.36E+08	1.92E+07	✓
F17	**7.96E+04**	**1.59E+05**	**1.75E+04**	5.76E+07	3.11E+08	2.74E+07	✓
F18	**3.83E+05**	**7.66E+05**	**1.46E+05**	3.53E+08	4.94E+08	3.43E+06	✓

Finally, we have Table 5.3 with the results of the functions evaluated in a thousand variables. These functions obtained a very high evaluation according to their $Gprs_{diff}$ value and in comparison with the GGA. Again, our proposal obtains better values in all cases.

It is essential to mention the difference in values since those obtained with DG2 are much higher.

Table 5.3: GGA-VD and DG2 statistical results in dimension 1000. Best results are shown in boldface

Algorithm	**GGA-VD**			**DG2**			
Function/Statistic	**BEST**	**MEDIAN**	**STD**	**BEST**	**MEDIAN**	**STD**	**W**
F1	**0.00E+00**	**0.00E+00**	**0.00E+00**	4.72E+02	7.66E+02	1.56E+02	✓
F2	**0.00E+00**	**0.00E+00**	**0.00E+00**	2.25E+03	2.36E+04	1.26E+03	✓
F3	**0.00E+00**	**0.00E+00**	**0.00E+00**	7.14E+03	9.23E+03	2.12E+03	✓
F4	**0.00E+00**	**0.00E+00**	**0.00E+00**	7.62E+02	1.80E+03	8.99E+03	✓
F5	**0.00E+00**	**0.00E+00**	**0.00E+00**	7.21E+03	8.49E+03	2.13E+03	✓
F6	**0.00E+00**	**0.00E+00**	**0.00E+00**	1.22E+04	2.99E+04	8.13E+03	✓
F7	**0.00E+00**	**0.00E+00**	**0.00E+00**	5.46E+03	5.57E+04	1.97E+03	✓
F8	**0.00E+00**	**0.00E+00**	**0.00E+00**	5.64E+04	7.62E+05	1.58E+04	✓
F9	**0.00E+00**	**0.00E+00**	**0.00E+00**	8.23E+04	6.34E+05	1.26E+05	✓
F10	**0.00E+00**	**0.00E+00**	**0.00E+00**	3.32E+06	1.82E+07	1.64E+05	✓
F11	**0.00E+00**	**0.00E+00**	**0.00E+00**	9.33E+06	9.95E+08	2.32E+07	✓
F12	**0.00E+00**	**0.00E+00**	**0.00E+00**	2.23E+08	8.22E+08	3.21E+07	✓
F13	**0.00E+00**	**0.00E+00**	**3.51E+04**	2.66E+08	6.23E+08	6.62E+06	✓
F14	**0.00E+00**	**0.00E+00**	**5.49E+03**	8.74E+08	3.77E+09	1.22E+08	✓
F15	**0.00E+00**	**0.00E+00**	**2.00E+05**	6.41E+08	4.33E+09	3.31E+08	✓
F16	**4.07E+05**	**8.15E+05**	**1.27E+05**	2.39E+08	5.36E+08	2.92E+07	✓
F17	**4.32E+04**	**8.63E+04**	**2.52E+04**	3.46E+09	2.21E+10	7.63E+09	✓
F18	**4.20E+05**	**8.41E+05**	**1.20E+05**	6.85E+09	9.14E+09	4.33E+08	✓

The tables above show us how the DG2 algorithm performs poorly regarding the $Gprs_{diff}$ values. This may be because this algorithm was initially proposed for functions without constraints; in addition, the benchmark functions in this study are more complex than those presented by Omidvar et al., where they propose this algorithm.

5.1.2 Experiment 2. Empirical Evaluation between GGA-VD and DVIIC

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. [76].

Therefore, we chose the same set of test functions the authors used. It is the first set for Large-Scale Constrained Optimization Problems, which Sayed et al. proposed in 2015 [54] and we described in Section 2.4.

We have compared the results of our GGA-VD against the DVIIC, in which Aguilar et al. [76] 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, 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.

The following tables show the results of executing our proposed GGA-VD and the Genetic Algorithm DVIIC. Furthermore, each table shows each function's Wilcoxon Rank Sum test at a 95% confidence level (column W). A checkmark (✓) means significant differences in favor of the GGA; an equality symbol (=) represents no significant differences between both algorithms.

First, Table 5.4 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 standard deviation of the values of the evaluation function (Equation 4.5) are shown.

Table 5.4: GGA-VD and DVIIC statistical results in dimension 100. Best results are shown in boldface.

Algorithm	**GGA-VD**			**DVIIC**			
Function/Statistic	**Best**	**Median**	**Std**	**Best**	**Median**	**Std**	**W**
F1	**0.00E+00**	**0.00E+00**	**0.00E+00**	**0.00E+00**	2.18E-11	2.96E-11	✓
F2	**0.00E+00**	**0.00E+00**	**0.00E+00**	**0.00E+00**	2.35E+04	1.20E+04	✓
F3	**0.00E+00**	**0.00E+00**	**0.00E+00**	1.38E+05	1.38E+05	6.77E+04	✓
F4	**0.00E+00**	**0.00E+00**	**0.00E+00**	4.29E+04	8.57E+04	1.69E+04	✓
F5	**0.00E+00**	**0.00E+00**	**0.00E+00**	1.27E+04	1.78E+04	2.48E+03	✓
F6	**0.00E+00**	**0.00E+00**	**0.00E+00**	8.97E+05	1.44E+06	2.36E+05	✓
F7	**0.00E+00**	**0.00E+00**	**0.00E+00**	2.51E+05	1.01E+06	2.62E+05	✓
F8	**0.00E+00**	**0.00E+00**	**0.00E+00**	6.70E+05	8.38E+05	9.84E+04	✓
F9	**0.00E+00**	**0.00E+00**	**0.00E+00**	2.13E+03	3.20E+03	3.58E+02	✓
F10	**0.00E+00**	**0.00E+00**	**0.00E+00**	5.36E+05	1.07E+06	2.05E+05	✓
F11	**0.00E+00**	**0.00E+00**	**0.00E+00**	4.12E+02	5.84E+02	9.04E+01	✓
F12	**0.00E+00**	**0.00E+00**	**0.00E+00**	9.28E+03	1.76E+04	3.20E+03	✓
F13	**0.00E+00**	**0.00E+00**	**2.31E+04**	6.31E+05	1.09E+06	1.08E+05	✓
F14	**0.00E+00**	**0.00E+00**	**5.35E+02**	1.55E+07	1.93E+07	1.72E+06	✓
F15	**4.66E-10**	**4.66E-10**	**2.60E+02**	5.42E+06	8.72E+06	9.33E+05	✓
F16	**1.79E+04**	**5.38E+04**	**1.72E+04**	4.93E+05	6.81E+05	7.26E+04	✓
F17	**1.07E+05**	**4.28E+05**	9.52E+04	4.83E+05	6.99E+05	**8.07E+04**	✓
F18	**1.63E+05**	**4.88E+05**	**1.40E+05**	1.80E+06	2.62E+06	2.94E+05	✓

We can observe in the same table that the GGA-VD improves the decomposition evaluation function value in all cases compared to DVIIC. As we can see, unlike DVIIC, the GGA-VD reaches the value of 0 in the best result in 14 of the 18 cases.

Furthermore, the median and standard deviation values obtained by the GGA-VD 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 significant differences in favor of the GGA-VD in all cases.

Table 5.5: GGA-VD and DVIIC statistical results in dimension 500. The best results are shown in boldface.

Algorithm	**GGA-VD**			**DVIIC**			
Function/Statistic	**Best**	**Median**	**Std**	**Best**	**Median**	**Std**	**W**
F1	**0.00E+00**	**0.00E+00**	**0.00E+00**	**0.00E+00**	**0.00E+00**	**0.00E+00**	=
F2	**0.00E+00**	**0.00E+00**	**0.00E+00**	4.66E-10	4.43E+04	1.16E+04	✓
F3	**0.00E+00**	**0.00E+00**	**0.00E+00**	3.59E+04	7.19E+04	1.23E+04	✓
F4	**0.00E+00**	**0.00E+00**	**0.00E+00**	1.17E+06	1.24E+06	2.98E+04	✓
F5	**0.00E+00**	**0.00E+00**	**0.00E+00**	7.20E+06	8.73E+06	5.68E+05	✓
F6	**0.00E+00**	**0.00E+00**	**0.00E+00**	3.34E+06	4.12E+06	2.09E+05	✓
F7	**0.00E+00**	**0.00E+00**	**0.00E+00**	5.19E+04	1.13E+05	1.94E+04	✓
F8	**0.00E+00**	**0.00E+00**	**0.00E+00**	8.79E+05	9.75E+05	4.19E+04	✓
F9	**0.00E+00**	**0.00E+00**	**0.00E+00**	5.58E+04	1.67E+05	3.46E+04	✓
F10	**0.00E+00**	**0.00E+00**	**0.00E+00**	9.69E+06	1.19E+07	5.03E+05	✓
F11	**0.00E+00**	**0.00E+00**	**0.00E+00**	1.95E+06	2.58E+06	1.80E+05	✓
F12	**0.00E+00**	**0.00E+00**	**0.00E+00**	3.50E+06	3.84E+06	1.10E+05	✓
F13	**0.00E+00**	**0.00E+00**	**2.56E+04**	9.61E+05	1.26E+06	7.07E+04	✓
F14	**0.00E+00**	**0.00E+00**	**6.86E+04**	9.10E+05	1.35E+06	1.08E+05	✓
F15	**0.00E+00**	**5.96E-08**	**1.27E+05**	7.97E+06	9.88E+06	4.48E+05	✓
F16	**8.13E+02**	**3.25E+03**	**7.04E+02**	1.38E+07	1.69E+07	7.77E+05	✓
F17	**7.96E+04**	**1.59E+05**	**1.75E+04**	2.26E+07	2.43E+07	1.48E+07	✓
F18	**3.83E+05**	**7.66E+05**	**1.46E+05**	2.78E+07	3.63E+07	2.05E+06	✓

Second, Table 5.5 shows the results of our algorithm to solve the same 18 functions, now with 500 variables. The GGA-VD 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 5.4, 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 no significant differences between the two algorithms in function 1. It shows significant differences in favor of the GGA-VD 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-VD approach, and F_1 is trivial to solve by the two algorithms.

Finally, Table 5.6 contains the results for the 18 functions with both algorithms implemented on 1000 variables.

Table 5.6: GGA-VD and DVIIC statistical results in dimension 1000. The best results are shown in boldface.

Algorithm	**Algorithm**			**DVIIC**			
Function/Statistic	**Best**	**Median**	**Std**	**Best**	**Median**	**Std**	**W**
F1	**0.00E+00**	**0.00E+00**	**0.00E+00**	**0.00E+00**	**0.00E+00**	**0.00E+00**	=
F2	**0.00E+00**	**0.00E+00**	**0.00E+00**	4.18E+03	1.25E+04	5.39E+03	✓
F3	**0.00E+00**	**0.00E+00**	**0.00E+00**	2.42E+02	2.90E+02	1.98E+01	✓
F4	**0.00E+00**	**0.00E+00**	**0.00E+00**	3.93E+06	4.86E+06	2.33E+05	✓
F5	**0.00E+00**	**0.00E+00**	**0.00E+00**	1.33E+06	1.73E+06	1.19E+05	✓
F6	**0.00E+00**	**0.00E+00**	**0.00E+00**	1.05E+06	1.36E+06	7.85E+04	✓
F7	**0.00E+00**	**0.00E+00**	**0.00E+00**	1.10E+06	1.61E+06	1.28E+05	✓
F8	**0.00E+00**	**0.00E+00**	**0.00E+00**	4.16E+06	4.91E+06	1.82E+05	✓
F9	**0.00E+00**	**0.00E+00**	**0.00E+00**	4.56E+06	4.92E+06	1.09E+05	✓
F10	**0.00E+00**	**0.00E+00**	**0.00E+00**	2.16E+06	2.47E+06	8.18E+04	✓
F11	**0.00E+00**	**0.00E+00**	**0.00E+00**	1.99E+05	2.13E+05	4.33E+03	✓
F12	**0.00E+00**	**0.00E+00**	**0.00E+00**	2.93E+05	3.18E+05	8.43E+03	✓
F13	**0.00E+00**	**0.00E+00**	**3.51E+04**	3.36E+07	3.98E+07	1.49E+06	✓
F14	**0.00E+00**	**0.00E+00**	**5.49E+03**	4.60E+07	1.76E+08	6.74E+07	✓
F15	**0.00E+00**	**0.00E+00**	**2.00E+05**	1.02E+05	3.49E+07	1.55E+07	✓
F16	**4.07E+05**	**8.15E+05**	**1.27E+05**	7.71E+07	8.72E+07	2.37E+06	✓
F17	**4.32E+04**	**8.63E+04**	**2.52E+04**	7.71E+06	1.15E+07	1.05E+06	✓
F18	**4.20E+05**	**8.41E+05**	**1.20E+05**	3.03E+07	4.39E+07	3.07E+06	✓

In this experiment, we observed that similar to the experiment with 500 variables, our GGA-VD 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 no significant differences

between the two algorithms in function F_1 and indicate significant differences favoring the GGA-VD in the remaining 17 functions. Similarly to 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-VD approach for the other 17 remaining functions.

Given the previous tables, we confirm our algorithm performs better 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 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.

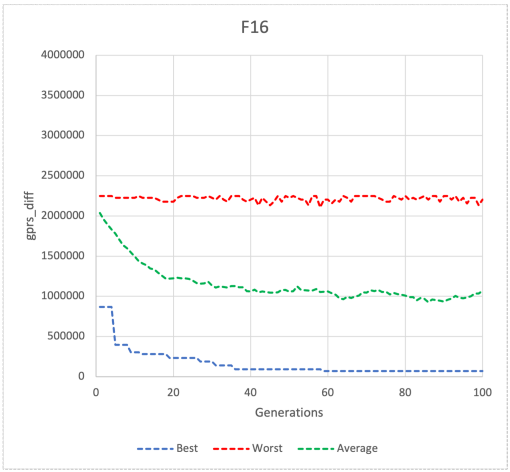
5.1.3 Experiment 3. Convergence Analysis of GGA-VD

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 briefly study our algorithm's convergence.

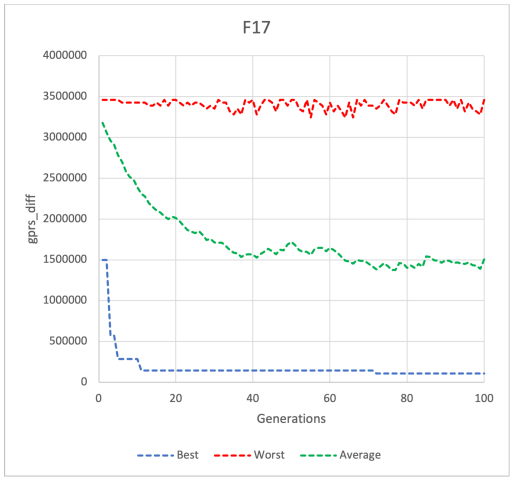
To understand the online behavior of our algorithm, we carried out some plots of the GGA-VD convergence for the most difficult functions of the benchmark. We shows the convergence in functions F_{16}, F_{17}, and F_{18} through 100 generations for three dimension values.

First, we show the convergence of the worst individual in the population- 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

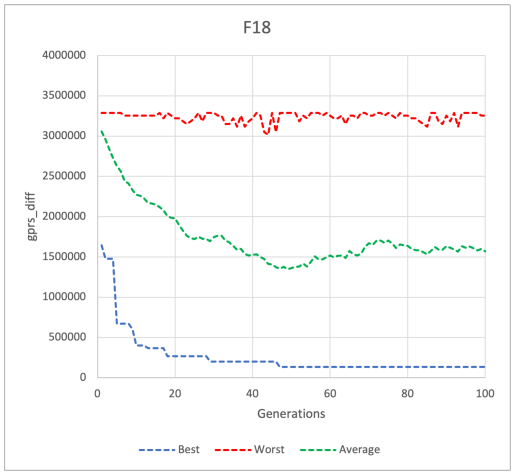
convergence across the 100 generations of the best individual in the population in terms of their decomposition evaluation value (blue color).



(a) 100D-F_{16}



(b) 100D-F_{17}



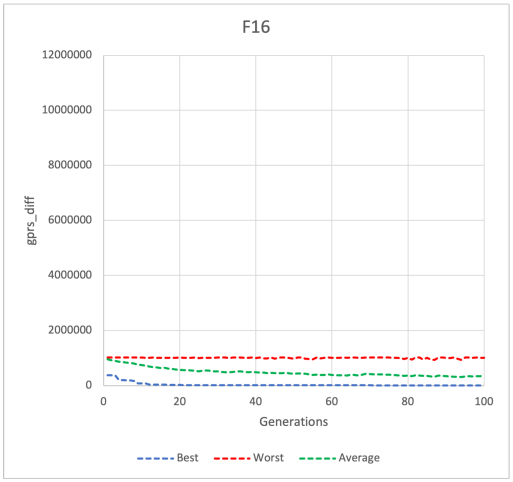
(c) 100D-F_{18}

Figure 5.1: Convergence plots of 100 generations for functions F_{16},F_{17}, and F_{18} with 100 variables.

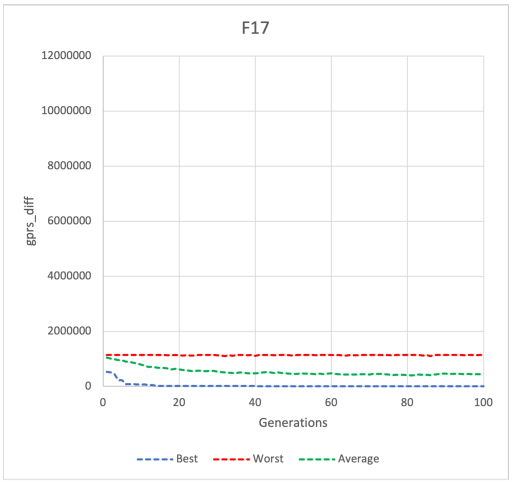
Figure 5.1 shows the convergence in the experiment with 100 variables from one of the 25 GGA-VD runs. We can observe similar behavior in the three functions, with decomposition evaluation values below 4.0×10^7 in all three cases (best, worst,

and average).

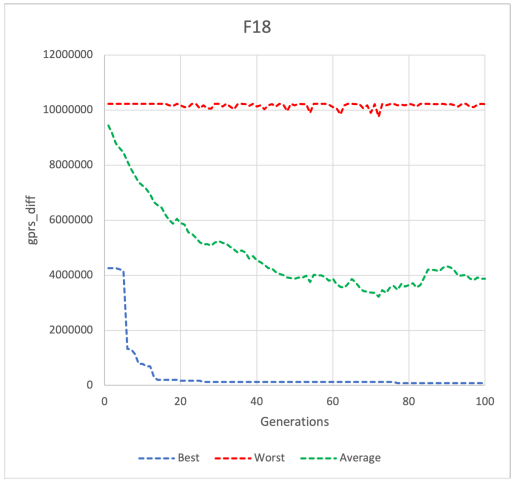
It is important to note that the behavior of the best individual presents an early convergence in the three functions and that the decomposition evaluation of the worst individuals has remained stable over the generations.



(a) 500D-F_{16}



(b) 500D-F_{17}

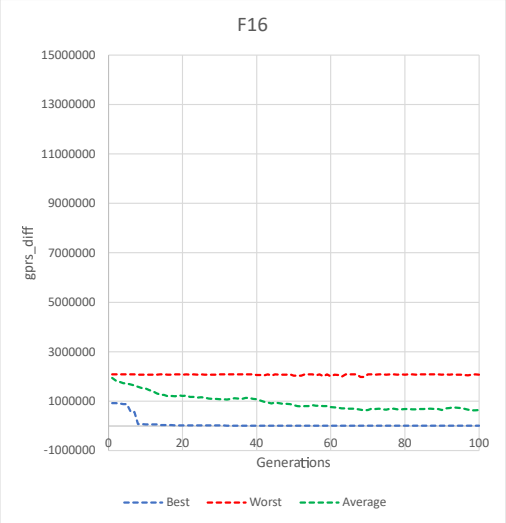


(c) 500D-F_{18}

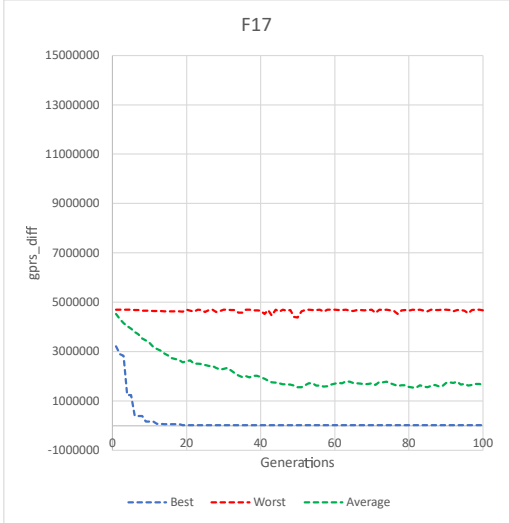
Figure 5.2: Convergence plots of 100 generations for functions F_{16}, F_{17}, and F_{18} with 500 variables.

Regarding the convergence of the functions with 500 variables (Figure 5.2),

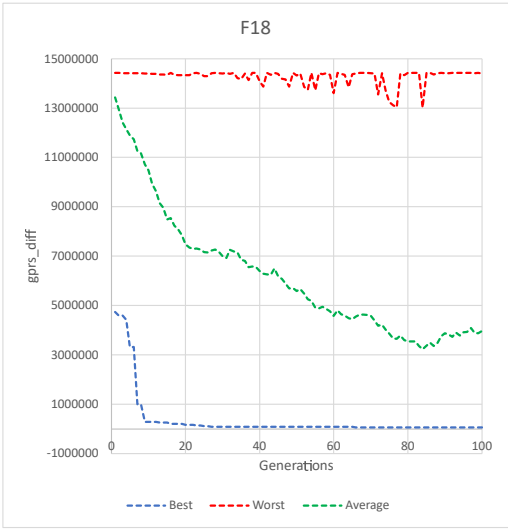
we can observe that function F_{18} shows the highest values of the decomposition evaluation for the three values (best, worst, and average), unlike the other functions.



(a) 1000D-F_{16}



(b) 1000D-F_{17}



(c) 1000D-F_{18}

Figure 5.3: Convergence plots of 100 generations for functions F_{16}, F_{17}, and F_{18} with 1000 variables.

Similar to those functions with 100 variables, this value does not converge to

zero in any of the cases, and the best individual has a quick convergence. According to the graphs, we can also infer 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 5.3), 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-VD do not appear to lead to better solutions. We can see from the plots that not all of the 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-VD produces reasonable 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.

5.2 Stage 2: Improvement of the GGA-VD

We describe the experiments to analyze the impact of using different selection and replacement strategies. In this section, we describe the experiments. Eight versions of the GGA-VD were created, which combine the three types of selection mentioned in Section 4.4.1 with the two replacement methods (six different versions). Also, we change the crossover and mutation operator for the improved ones also described in Section 4.4.1 (two different versions).

For these experiments, the three most difficult functions of the benchmark were selected to show the new proposal's performance compared to the results obtained by the original GGA. These functions are F_{16}, F_{17} and F_{18} with 100, 500 y 1000 variables.

The parameters were configured identically to those in Stage 1 for a fair comparison, using the same computing equipment.

The last column in each table shows whether or not significant differences are using the Wilcoxon test at a 95% confidence level (W column). If the significant differences favor the original $GGA-VD$ with GGA-VD, a checkmark (✓) is shown; if they are against the original proposal, a mark (×) is shown; and if there are no significant differences between the two algorithms, the equals sign (=) is displayed.

5.2.1 Experiment 1

This first experiment combines roulette selection and replacement of the worst individuals with the offspring created. The statistical results of this combination are shown in Tables 5.7, 5.8 and 5.9, we can observe evident differences only in function 17 evaluated in 1000 dimensions. In $100D$, we see the same best and median values for all the cases with the changed algorithm, similarly in $500D$. It seems to be not a fundamental change with the new selection and replacement methods.

Table 5.7: Statistical results of GGA-VD with roulette selection and replacement of the worst - 100D

Algorithm	**GGA-VD**			**GGA-RS-WR**			
Function/Statistic	**BEST**	**MEDIAN**	**STD**	**BEST**	**MEDIAN**	**STD**	**W**
F16	**1.79E+04**	5.38E+04	1.72E+04	9.05E+04	2.72E+05	8.54E+04	✓
F17	**1.07E+05**	4.28E+05	9.52E+04	9.05E+04	2.72E+05	7.39E+04	✓
F18	**1.63E+05**	4.88E+05	1.40E+05	9.05E+04	2.72E+05	8.31E+04	✓

Table 5.8: Statistical results of GGA-VD with roulette selection and replacement of the worst - 500D

Algorithm	**GGA-VD**			**GGA-RS-WR**			
Function/Statistic	**BEST**	**MEDIAN**	**STD**	**BEST**	**MEDIAN**	**STD**	**W**
F16	**8.13E+02**	3.25E+03	7.04E+02	7.17E+04	2.87E+05	7.78E+04	✓
F17	7.96E+04	1.59E+05	1.75E+04	**7.17E+04**	2.15E+05	7.26E+04	×
F18	3.83E+05	7.66E+05	1.46E+05	**7.17E+04**	2.15E+05	4.94E+04	×

Table 5.9: Statistical results of GGA-VD with roulette selection and replacement of the worst - 1000D

Algorithm	**GGA-VD**			**GGA-RS-WR**			
Function/Statistic	**BEST**	**MEDIAN**	**STD**	**BEST**	**MEDIAN**	**STD**	**W**
F16	4.07E+05	8.15E+05	1.27E+05	**6.81E+04**	1.02E+05	1.01E+05	×
F17	4.32E+04	8.63E+04	2.52E+04	**8.54E+02**	3.42E+03	5.91E+04	×
F18	4.20E+05	8.41E+05	1.20E+05	**4.11E+05**	4.11E+05	6.84E+04	×

5.2.2 Experiment 2

The second experiment is a combination of roulette selection and controlled selection. In tables 5.10, 5.11, and 5.12, we have an evident improvement in most of the functions. There are best values for the results over 100 variables, but they are the same for the best and median values. In $500D$, the results were worse than the ones of the original version of GGA.

Table 5.10: Statistical results of GGA-VD with roulette selection and controlled replacement - 100D

Algorithm	**GGA-VD**			**GGA-RS-CR**			
Function/Statistic	**BEST**	**MEDIAN**	**STD**	**BEST**	**MEDIAN**	**STD**	**W**
F16	1.79E+04	5.38E+04	1.72E+04	**2.35E+03**	4.70E+03	2.02E+03	×
F17	1.07E+05	4.28E+05	9.52E+04	**2.35E+03**	4.70E+03	4.50E+03	×
F18	1.63E+05	4.88E+05	1.40E+05	**2.35E+03**	4.70E+03	2.88E+03	×

Table 5.11: Statistical results of GGA-VD with roulette selection and controlled replacement - 500D

Algorithm	**GGA-VD**			**GGA-RS-CR**			
Function/Statistic	**BEST**	**MEDIAN**	**STD**	**BEST**	**MEDIAN**	**STD**	**W**
F16	**8.13E+02**	3.25E+03	7.04E+02	4.45E+05	5.56E+05	2.21E+05	✓
F17	**7.96E+04**	1.59E+05	1.75E+04	4.45E+05	4.45E+05	9.44E+04	✓
F18	**3.83E+05**	7.66E+05	1.46E+05	4.45E+05	4.45E+05	2.79E+05	✓

Table 5.12: Statistical results of GGA-VD with roulette selection and controlled replacement - 1000D

Algorithm	**GGA-VD**			**GGA-RS-CR**			
Function/Statistic	**BEST**	**MEDIAN**	**STD**	**BEST**	**MEDIAN**	**STD**	**W**
F16	4.07E+05	8.15E+05	1.27E+05	**1.96E+04**	1.96E+04	8.25E+03	×
F17	4.32E+04	8.63E+04	2.52E+04	**9.78E+03**	1.96E+04	5.55E+03	×
F18	4.20E+05	8.41E+05	1.20E+05	**1.47E+04**	2.20E+04	7.29E+03	×

5.2.3 Experiment 3

In the third experiment, which combines tournament selection and replacement of the worst individuals, Tables 5.13, 5.14, and 5.15 show that smaller fitness values are obtained in most functions. The behavior of the best results in functions 17 and 18 with a thousand variables is interesting; this can be a good indicator for using these selection and replacement strategies.

Table 5.13: Statistical results of GGA-VD with tournament selection and replacement of the worst - 100D

Algorithm	**GGA-VD**			**GGA-TS-WR**			
Function/Statistic	**BEST**	**MEDIAN**	**STD**	**BEST**	**MEDIAN**	**STD**	**W**
F16	1.79E+04	5.38E+04	1.72E+04	**1.06E+04**	4.24E+04	9.44E+03	×
F17	1.07E+05	4.28E+05	9.52E+04	**2.12E+04**	3.18E+04	1.00E+04	×
F18	1.63E+05	4.88E+05	1.40E+05	**2.12E+04**	3.18E+04	9.34E+03	×

Table 5.14: Statistical results of GGA-VD with tournament selection and replacement of the worst - 500D

Algorithm	**GGA-VD**			**GGA-TS-WR**			
Function/Statistic	**BEST**	**MEDIAN**	**STD**	**BEST**	**MEDIAN**	**STD**	**W**
F16	**8.13E+02**	3.25E+03	7.04E+02	7.37E+03	9.83E+03	1.81E+03	✓
F17	7.96E+04	1.59E+05	1.75E+04	**4.91E+03**	9.83E+03	1.88E+03	×
F18	3.83E+05	7.66E+05	1.46E+05	**7.37E+03**	9.83E+03	6.80E+02	×

Table 5.15: Statistical results of GGA-VD with tournament selection and replacement of the worst - 1000D

Algorithm	**GGA-VD**			**GGA-TS-WR**			
Function/Statistic	**BEST**	**MEDIAN**	**STD**	**BEST**	**MEDIAN**	**STD**	**W**
F16	**4.07E+05**	8.15E+05	1.27E+05	7.62E+05	7.62E+05	8.38E+04	✓
F17	4.32E+04	8.63E+04	2.52E+04	**3.91E+02**	5.22E+02	3.22E+05	×
F18	4.20E+05	8.41E+05	1.20E+05	**6.06E+02**	8.08E+02	1.68E+02	×

5.2.4 Experiment 4

The fourth experiment combines tournament selection and controlled replacement; according to the Tables 5.16, 5.17 and 5.18 in this experiment, most of the functions had a higher evaluation than the evaluation with the original proposal of the GGA-VD, that is, the combination of strategies of this experiment gives bad results.

Table 5.16: Statistical results of GGA-VD with tournament selection and controlled replacement - 100D

Algorithm	GGA-VD			GGA-TS-CR			
Function/Statistic	BEST	MEDIAN	STD	BEST	MEDIAN	STD	W
F16	**1.79E+04**	5.38E+04	1.72E+04	4.03E+04	5.37E+04	2.70E+04	✓
F17	1.07E+05	4.28E+05	9.52E+04	**4.03E+04**	5.37E+04	1.65E+04	×
F18	1.63E+05	4.88E+05	1.40E+05	**2.69E+04**	5.37E+04	4.74E+04	×

Table 5.17: Statistical results of GGA-VD with tournament selection and controlled replacement - 500D

Algorithm	GGA-VD			GGA-TS-CR			
Function/Statistic	BEST	MEDIAN	STD	BEST	MEDIAN	STD	W
F16	**8.13E+02**	3.25E+03	7.04E+02	4.03E+04	4.03E+04	2.86E+04	✓
F17	7.96E+04	1.59E+05	1.75E+04	**4.03E+04**	4.03E+04	9.73E+03	×
F18	3.83E+05	7.66E+05	1.46E+05	**4.03E+04**	4.03E+04	6.88E+04	×

Table 5.18: Statistical results of GGA-VD with tournament selection and controlled replacement - 1000D

Algorithm	GGA-VD			GGA-TS-CR			
Function/Statistic	BEST	MEDIAN	STD	BEST	MEDIAN	STD	W
F16	**4.07E+05**	8.15E+05	1.27E+05	7.22E+05	8.13E+05	2.48E+05	✓
F17	**4.32E+04**	8.63E+04	2.52E+04	5.42E+05	7.22E+05	2.42E+05	✓
F18	**4.20E+05**	8.41E+05	1.20E+05	7.22E+05	7.22E+05	3.45E+05	✓

5.2.5 Experiment 5

The fifth experiment combined controlled selection and replacement of the worst individuals. These results are shown in the Tables 5.19, 5.20 and 5.21, and we can observe that function 17 evaluated with 100 variables reached a value of best equal to cero, which does not happen in the other experiments, and in the functions

evaluated in 1000 variables the results obtained by the GGA-VD are improved. original.

Table 5.19: Statistical results of GGA-VD with controlled selection and replacement of the worst - 100D

Algorithm	**GGA-VD**			**GGA-CS-WR**			
Function/Statistic	**BEST**	**MEDIAN**	**STD**	**BEST**	**MEDIAN**	**STD**	**W**
F16	**1.79E+04**	5.38E+04	1.72E+04	2.93E+04	8.78E+04	2.81E+04	✓
F17	1.07E+05	4.28E+05	9.52E+04	**0.00E+00**	8.78E+04	3.10E+04	×
F18	1.63E+05	4.88E+05	1.40E+05	**2.93E+04**	8.78E+04	2.58E+04	×

Table 5.20: Statistical results of GGA-VD with controlled selection and replacement of the worst - 500D

Algorithm	**GGA-VD**			**GGA-CS-WR**			
Function/Statistic	**BEST**	**MEDIAN**	**STD**	**BEST**	**MEDIAN**	**STD**	**W**
F16	**8.13E+02**	3.25E+03	7.04E+02	2.77E+05	5.55E+05	1.41E+05	✓
F17	**7.96E+04**	1.59E+05	1.75E+04	1.39E+05	5.55E+05	1.07E+05	✓
F18	3.83E+05	7.66E+05	1.46E+05	**2.77E+05**	5.55E+05	1.42E+05	×

Table 5.21: Statistical results of GGA-VD with controlled selection and replacement of the worst - 1000D

Algorithm	**GGA-VD**			**GGA-CS-WR**			
Function/Statistic	**BEST**	**MEDIAN**	**STD**	**BEST**	**MEDIAN**	**STD**	**W**
F16	4.07E+05	8.15E+05	1.27E+05	**2.24E+03**	4.49E+03	7.30E+02	×
F17	4.32E+04	8.63E+04	2.52E+04	**3.37E+03**	4.49E+03	5.14E+02	×
F18	4.20E+05	8.41E+05	1.20E+05	**2.24E+03**	4.49E+03	9.09E+02	×

5.2.6 Experiment 6

In the sixth experiment, we combine controlled selection and controlled replacement. This combination does not show much improvement in performance according to Tables 5.22, 5.23, and 5.24 as it performs poorly compared to the original GGA.

Table 5.22: Statistical results of GGA-VD with controlled selection and controlled replacement - 100D

Algorithm	**GGA-VD**			**GGA-CS-CR**			
Function/Statistic	**BEST**	**MEDIAN**	**STD**	**BEST**	**MEDIAN**	**STD**	**W**
F16	1.79E+04	5.38E+04	1.72E+04	**1.01E+04**	3.11E+05	1.59E+05	×
F17	1.07E+05	4.28E+05	9.52E+04	**7.77E+04**	2.33E+05	8.30E+04	×
F18	1.63E+05	4.88E+05	1.40E+05	**1.55E+05**	2.33E+05	9.88E+04	×

Table 5.23: Statistical results of GGA-VD with controlled selection and controlled replacement - 500D

Algorithm	**GGA-VD**			**GGA-CS-CR**			
Function/Statistic	**BEST**	**MEDIAN**	**STD**	**BEST**	**MEDIAN**	**STD**	**W**
F16	**8.13E+02**	3.25E+03	7.04E+02	1.35E+05	1.80E+05	3.54E+04	✓
F17	**7.96E+04**	1.59E+05	1.75E+04	1.35E+05	1.80E+05	1.42E+04	✓
F18	3.83E+05	7.66E+05	1.46E+05	**1.35E+05**	1.80E+05	3.31E+04	×

Table 5.24: Statistical results of GGA-VD with controlled selection and controlled replacement - 1000D

Algorithm	**GGA-VD**			**GGA-CS-CR**			
Function/Statistic	**BEST**	**MEDIAN**	**STD**	**BEST**	**MEDIAN**	**STD**	**W**
F16	4.07E+05	8.15E+05	1.27E+05	**4.87E+04**	6.49E+04	1.54E+04	×
F17	**4.32E+04**	8.63E+04	2.52E+04	6.49E+04	6.49E+04	1.03E+04	✓
F18	4.20E+05	8.41E+05	1.20E+05	**6.49E+04**	6.49E+04	1.23E-07	×

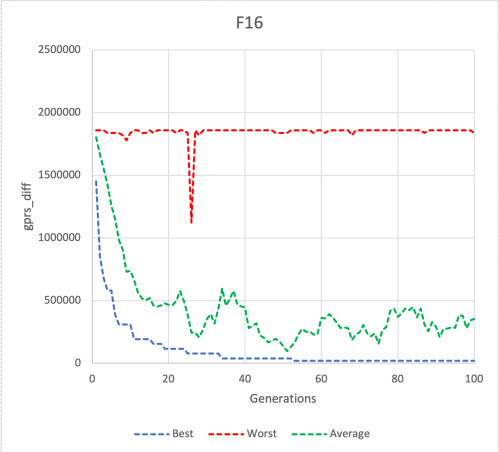
As part of the comparison of the results obtained through the six experiments, we linked convergence graphs that show for each of the three functions F_{16}, F_{17}, and F_{18}, the value of the difference $grps_{diff}$ of Equation 4.5 over the 100 generations for the best individual, the worst individual and the average fitness of the population. In all cases, the red line represents the change in the worst individual, the green line represents the average, and the blue line represents the best fitness per generation.

First, we describe the graphs with the functions in $100D$. Figure 5.4 shows the graphs of the six experiments evaluated in the F_{16} function with 100 variables. We can first observe that the line representing the worst individual is particularly different in the CS-CR combination that represents experiment 6; it is interesting to see how the worst individual is sometimes deficient and then returns to a very high value. Another characteristic is that the green and blue lines do not seem to touch or be very close in the first three experiments, unlike the last three, where this is more evident.

Then, Figure 5.5 shows the fitness behavior of function F_{17} in 100 dimensions; now, we can observe a particular behavior in the graphs of experiment 2 and experiment 6 in the line of the worst individual. Also, as in the graph of experiment 3, the average looks higher than the other experiments.

Finally, regarding the functions with 100 variables, we continue with the function F_{18}. Figure 5.6 with F_{18} shows something similar to F_{17}; however, the peaks in the red lines are less evident in this function. In addition, the green line representing the average seems to rise more often than in the other functions.

Another detail that we observe in these graphs is that in the three functions in the three dimensions, there are similar average values, just as it seems that at the beginning, all the lines start in the same range of fitness values.



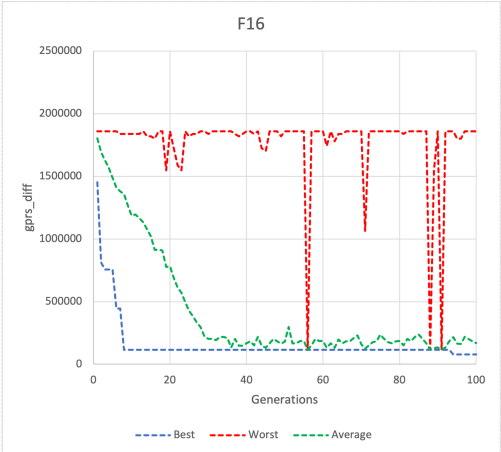
(a) 100D-F_{16} RS-WR

(b) 100D-F_{16} RS-CR

(c) 100D-F_{16} TS-WR

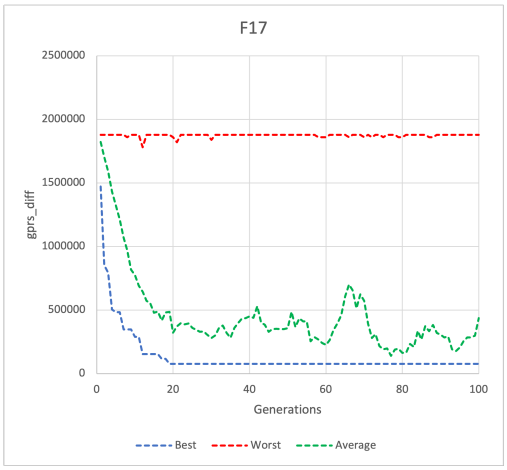
(d) 100D-F_{16} TS-CR

(e) 100D-F_{16} CS-WR

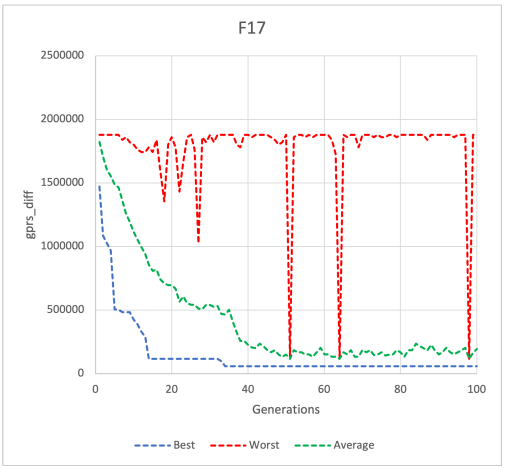


(f) 100D-F_{16} CS-CR

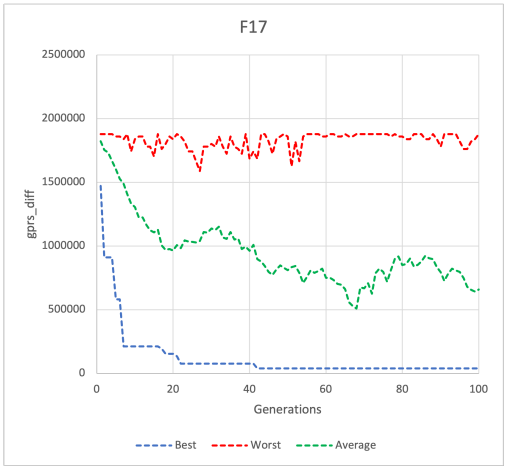
Figure 5.4: Convergence plots of function F_{16}, $D = 100$ comparing the combinations of selection and replacement.



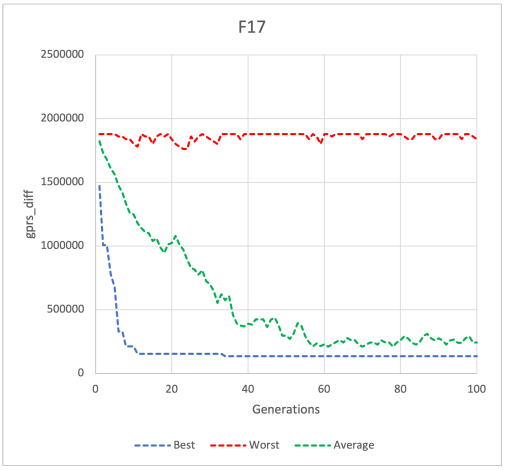
(a) 100D-F_{17} RS-WR



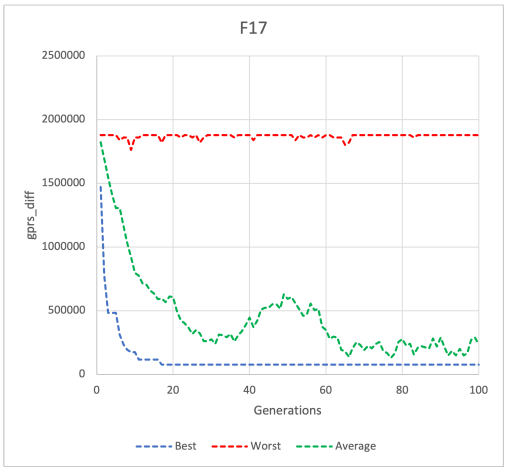
(b) 100D-F_{17} RS-CR



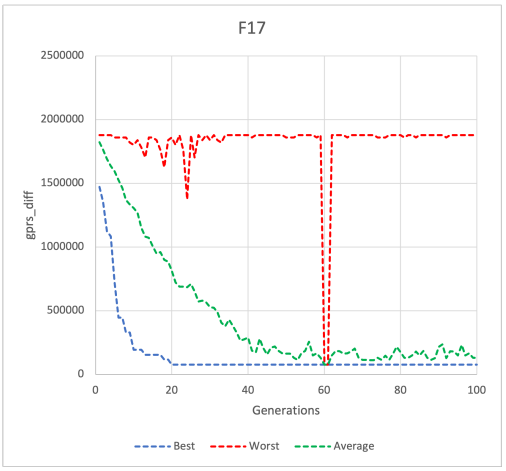
(c) 100D-F_{17} TS-WR



(d) 100D-F_{17} TS-CR

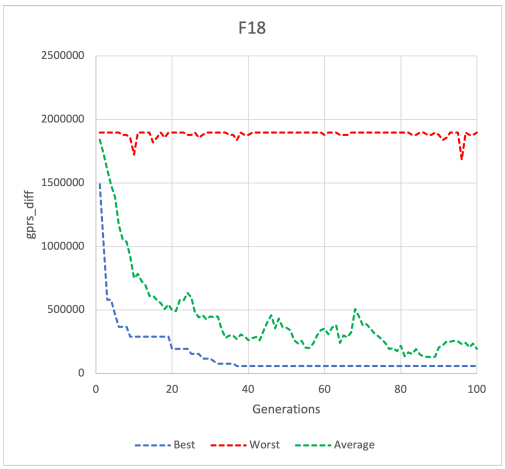


(e) 100D-F_{17} CS-WR

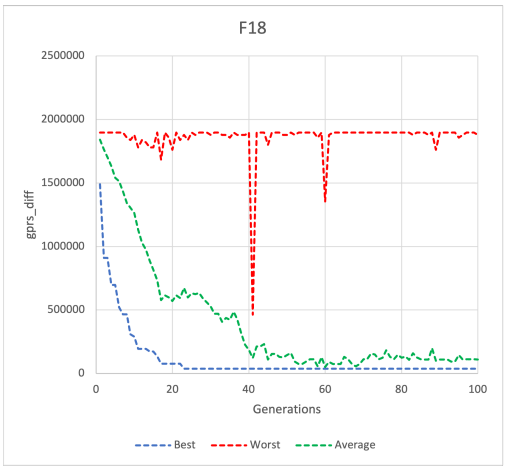


(f) 100D-F_{17} CS-CR

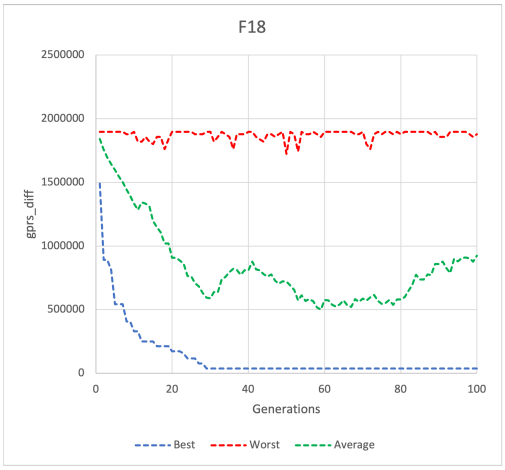
Figure 5.5: Convergence plots of function F_{17}, $D = 100$ comparing the combinations of selection and replacement.



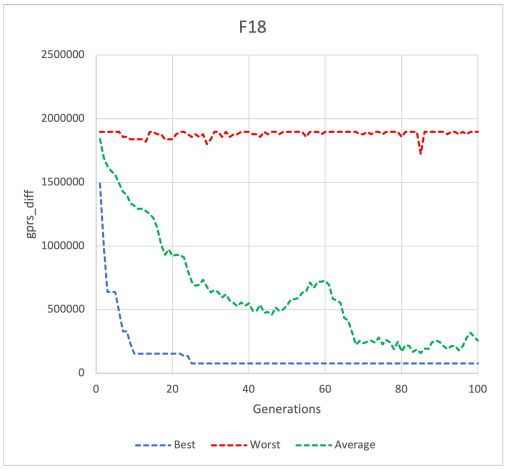
(a) 100D-F_{18} RS-WR



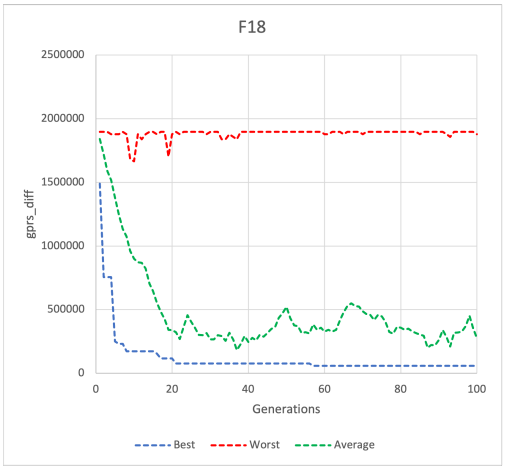
(b) 100D-F_{18} RS-CR



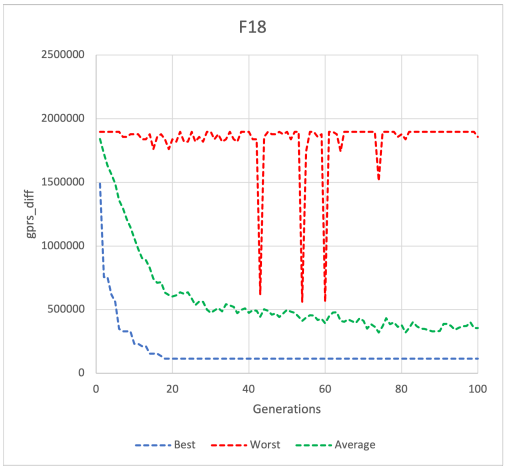
(c) 100D-F_{18} TS-WR



(d) 100D-F_{18} TS-CR



(e) 100D-F_{18} CS-WR



(f) 100D-F_{18} CS-CR

Figure 5.6: Convergence plots of function F_{18}, $D = 100$ comparing the combinations of selection and replacement.

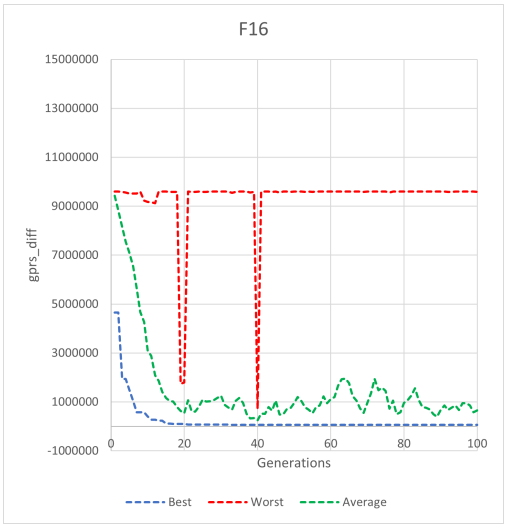
Now, we analyze Figures 5.7, 5.8, and 5.9, these are the graphs for the three functions in $500D$.

In Figure 5.7, we have the graphs of the six experiments with the F_16 function at $500D$. It is interesting to see the behavior of the worst individuals throughout the generations; for example, for experiment 1, two peaks are noted that drop considerably compared to the other graphs. In experiments 2 and 6, for example, it is noted how the curve descends and ascends constantly. For the green curve that represents the average fitness, we see how in experiment 1 and experiment 5, it drops very quickly and then rises and falls on several occasions. And finally, the blue curve descends quickly in most cases, except in experiment 2.

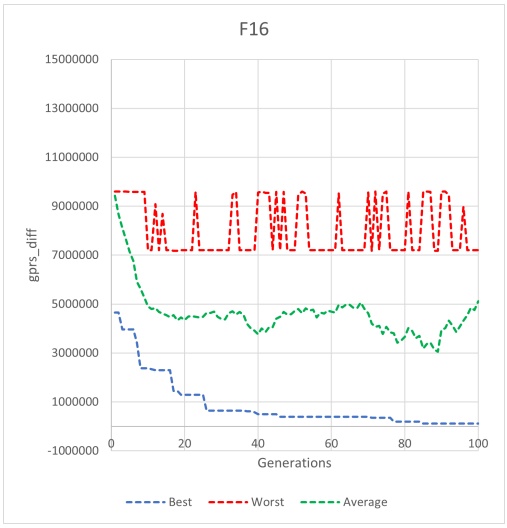
In the following function, Figure 5.8, F_{17}, we now observe shallow peaks in four of the six experiments in the curve of the worst individual. In contrast, in experiments 3 and 4, the curve does not decrease much over the hundred generations. On the other hand, the behavior of the best individual is similar in the six experiments; it seems that the curve decreases early and remains constant or has a very slight change over the rest of the generations.

Finally, regarding the last function, i.e., F_{18}, whose graphs we observe in Figure 5.9, we see similar behavior to the function F_{17}, which speaks about the curve of the best individual throughout the generations. Conversely, the green line generally seems to be descending in most cases with slight peaks. Regarding the red curve, we again observe very different behaviors; for example, in the first and fourth experiments, we have a single shallow peak, while in experiments 2, 3, and 6, there are peaks that are not as deep but are maintained throughout the generations.

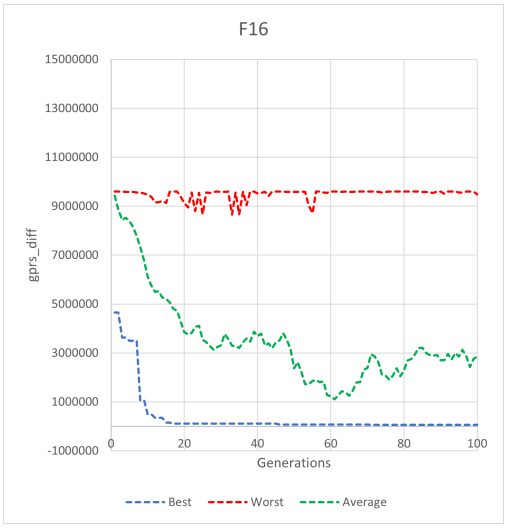
This behavior could indicate the changes the selected strategies manage to make to the population's individuals and how they vary quite a lot. So, we continue analyzing the images.



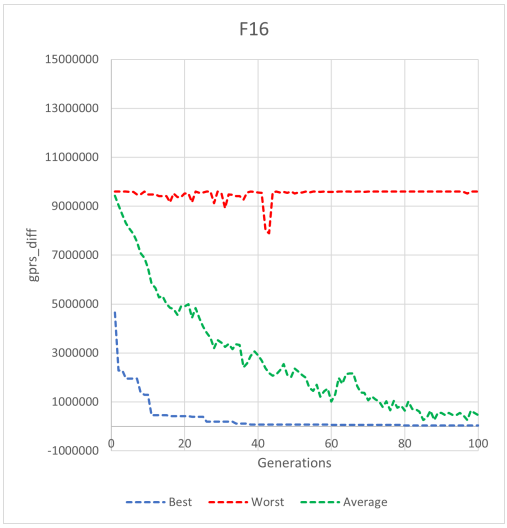
(a) 500D-F_{16} RS-WR



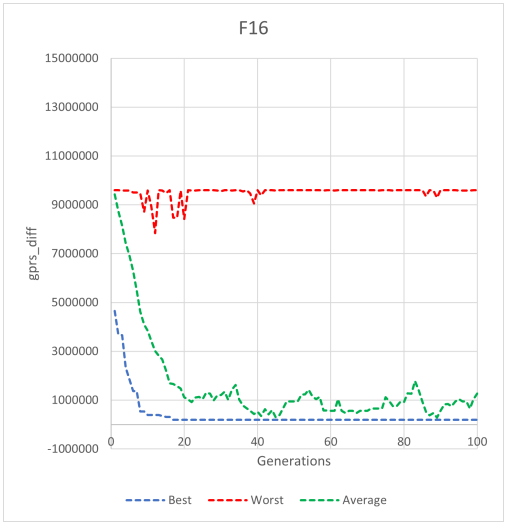
(b) 500D-F_{16} RS-CR



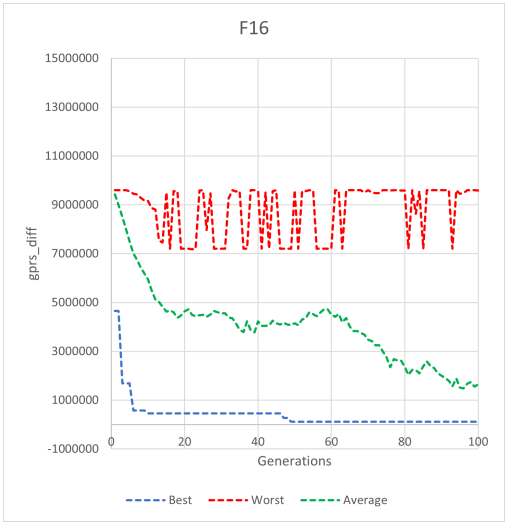
(c) 500D-F_{16} TS-WR



(d) 500D-F_{16} TS-CR

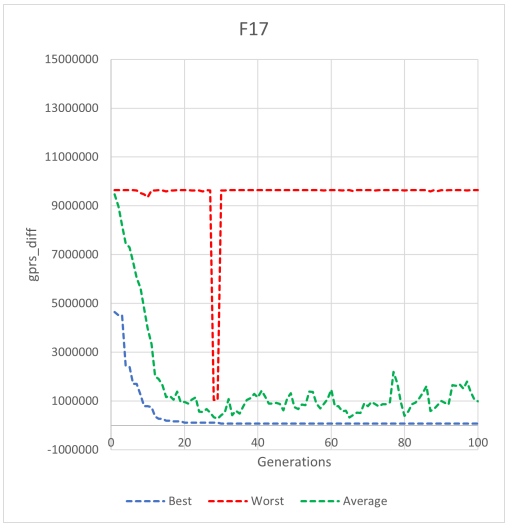


(e) 500D-F_{16} CS-WR

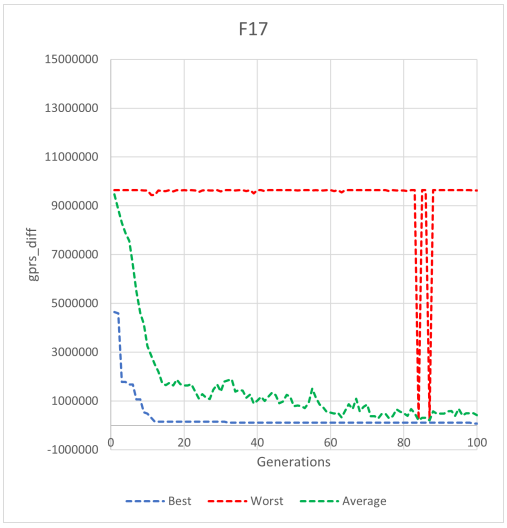


(f) 500D-F_{16} CS-CR

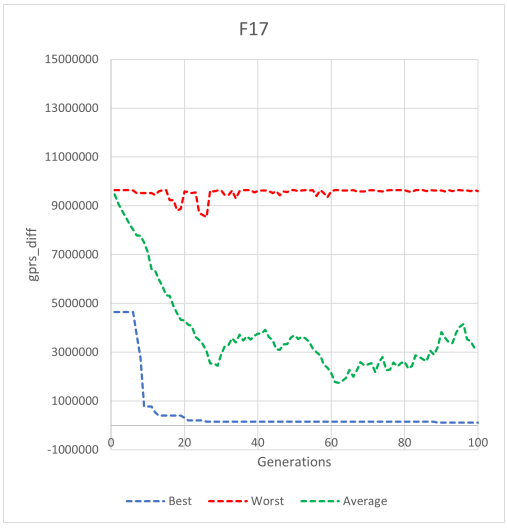
Figure 5.7: Convergence plots of function F_{16}, $D = 500$ comparing the combinations of selection and replacement.



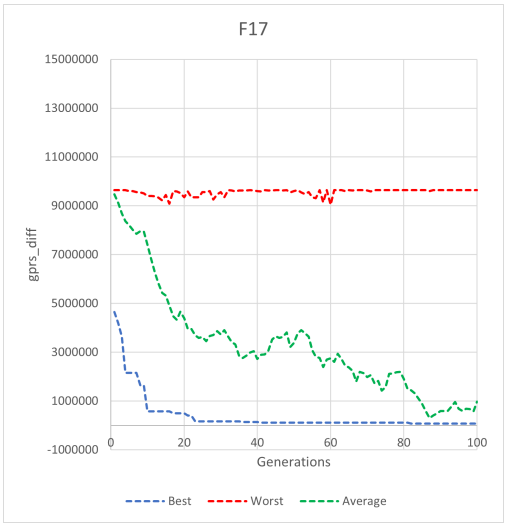
(a) 500D-F_{17} RS-WR



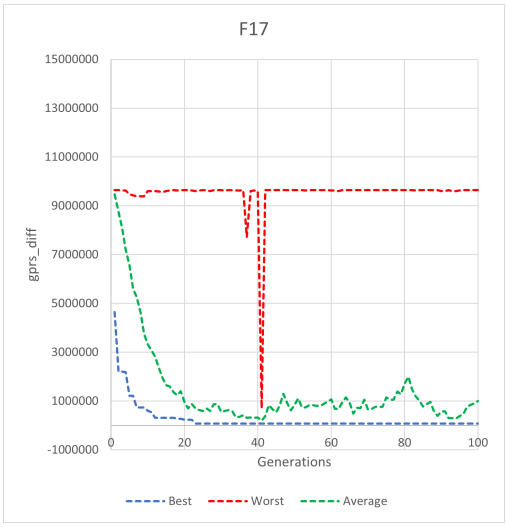
(b) 500D-F_{17} RS-CR



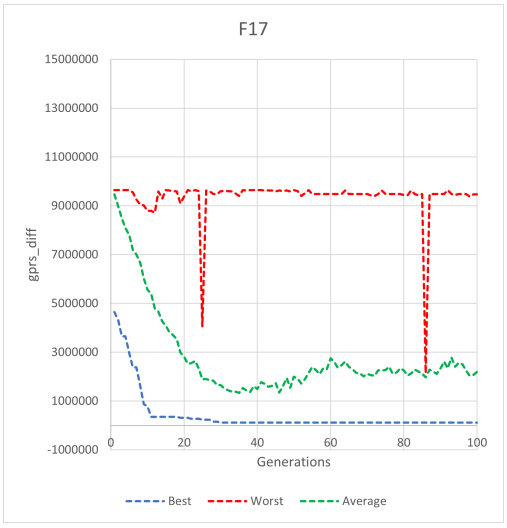
(c) 500D-F_{17} TS-WR



(d) 500D-F_{17} TS-CR

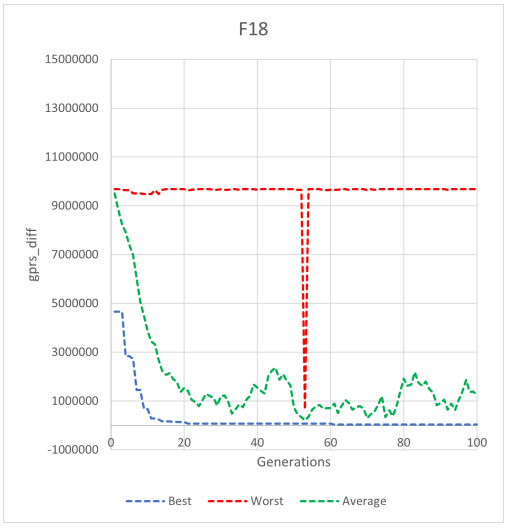


(e) 500D-F_{17} CS-WR

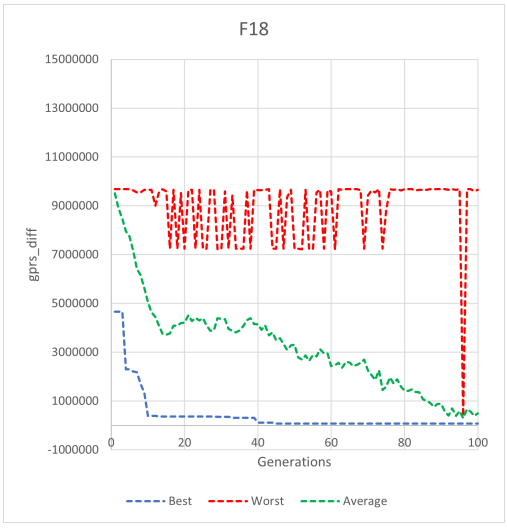


(f) 500D-F_{17} CS-CR

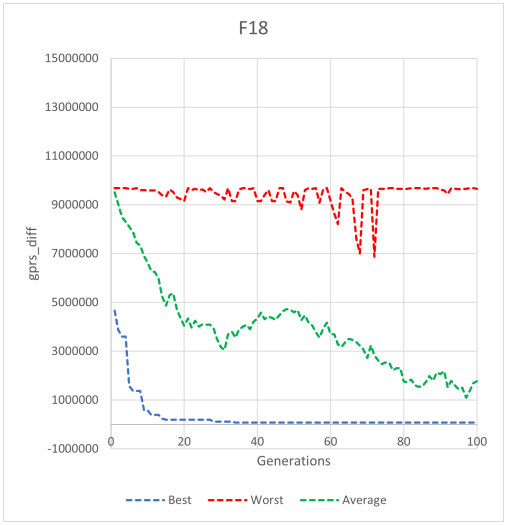
Figure 5.8: Convergence plots of function F_{17}, $D = 500$ comparing the combinations of selection and replacement.



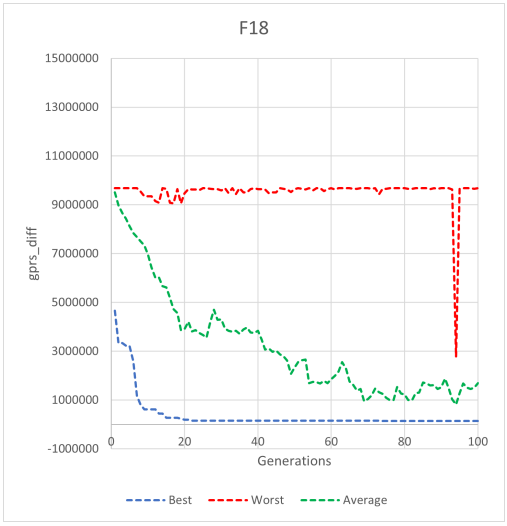
(a) 500D-F_{18} RS-WR



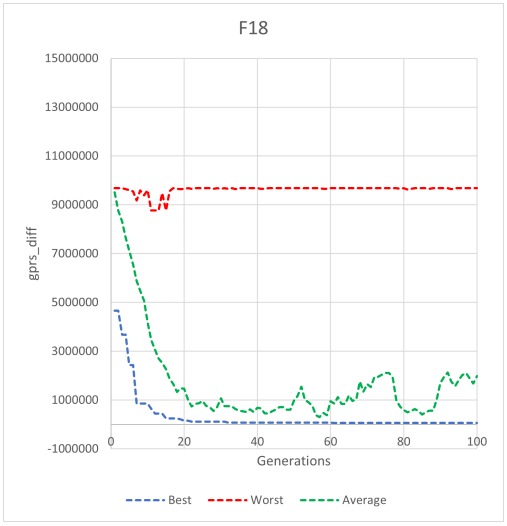
(b) 500D-F_{18} RS-CR



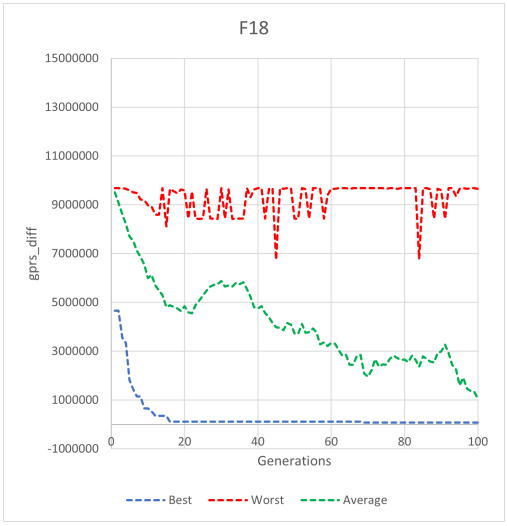
(c) 500D-F_{18} TS-WR



(d) 500D-F_{18} TS-CR



(e) 500D-F_{18} CS-WR



(f) 500D-F_{18} CS-CR

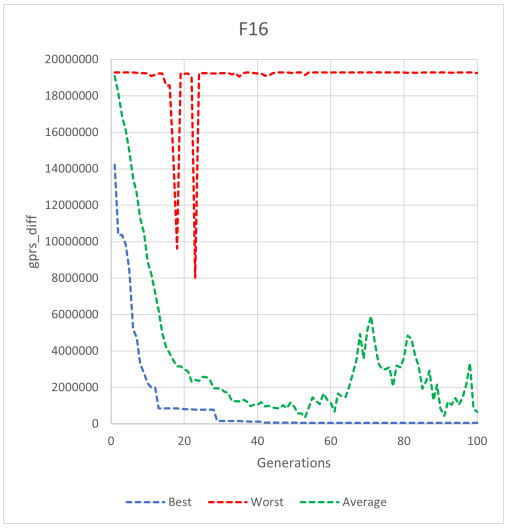
Figure 5.9: Convergence plots of function F_{18}, $D = 500$ comparing the combinations of selection and replacement.

The last graphs with the results of these six experiments belong to the functions evaluated in a thousand dimensions. The functions are the same three: F_{16}, F_{17}, and F_{18}, and the graphs of each experiment for each function are shown in Figures 5.10, 5.11, and 5.12 respectively.

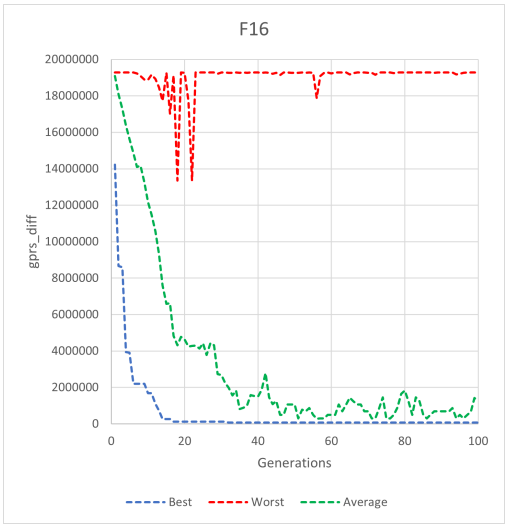
In the case of F_{16}, the line representing the best individual drops drastically before twenty generations in almost all cases. Regarding the green line, which is the average of the population evaluations, we see that it is generally descending; however, in some cases, there are many peaks; for example, in the case of experiments 3 and 5, it even seems that the curve begins to rise again. Regarding the worst individual, represented by the red line, we see constant behaviors except for some declines, which in the case of experiments 4 and 6 seem to be very close to the best.

Secondly, the function F_{17} shares how the blue curve behaves with the previous function. It is interesting how, again, in experiments 3 and 5, the behavior of the averages rises quite a bit. In addition, in half of the experiments, we have a behavior close to constant in the curve of the population's worst individual; in some others, we can again observe the peaks that fall quite a bit.

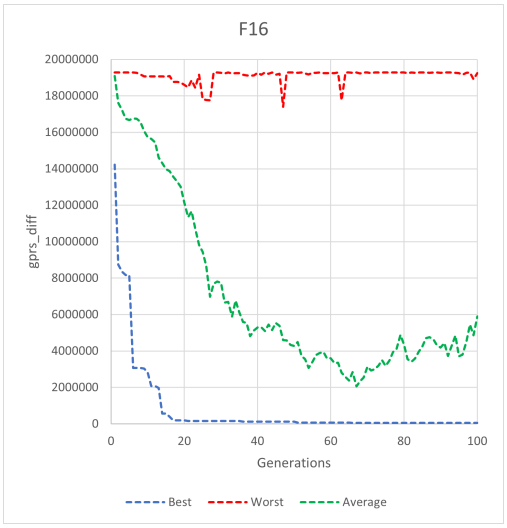
Regarding the last function F_{18}, we have a different behavior in the previous experiment; we see how the curve of the average of the evaluations rises around generation 60 and does not descend again; from that point onwards, the red curve is also more constant. Again, the curve of the best individual falls from the initial generations. Experiments 3 and 4 do not contain such evident peaks in the red curve, unlike the other experiments; however, in reality, none coincide or come close to the best individual in the population.



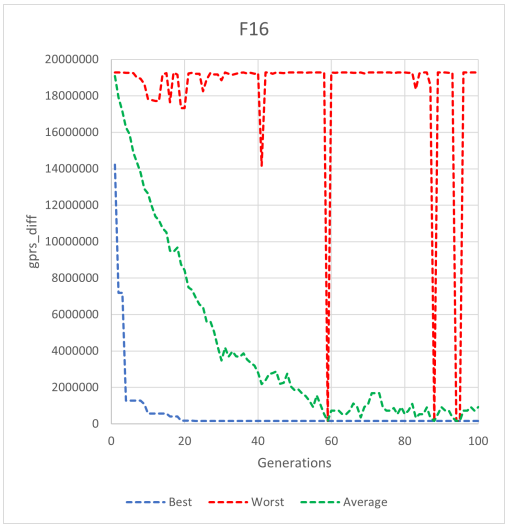
(a) 1000D-F_{16} RS-WR



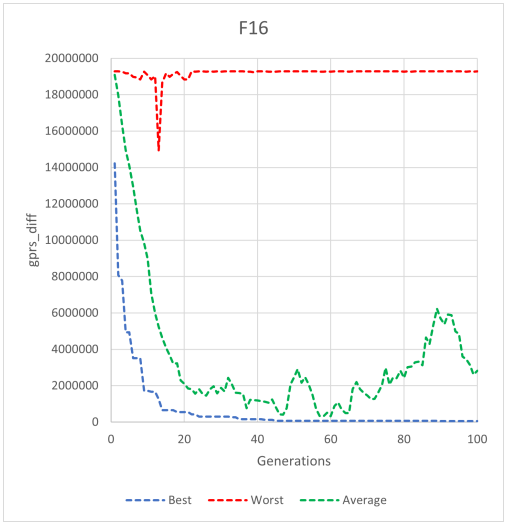
(b) 1000D-F_{16} RS-CR



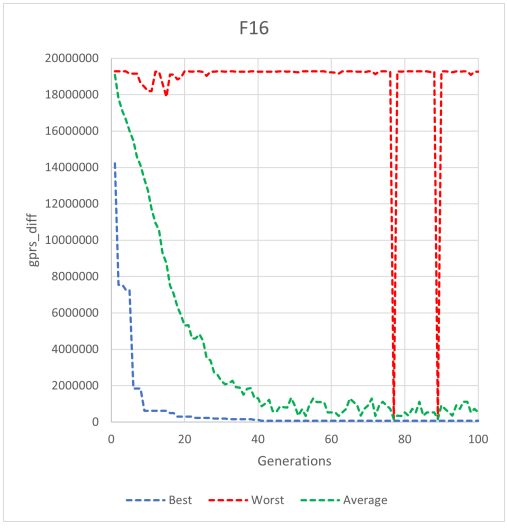
(c) 1000D-F_{16} TS-WR



(d) 1000D-F_{16} TS-CR

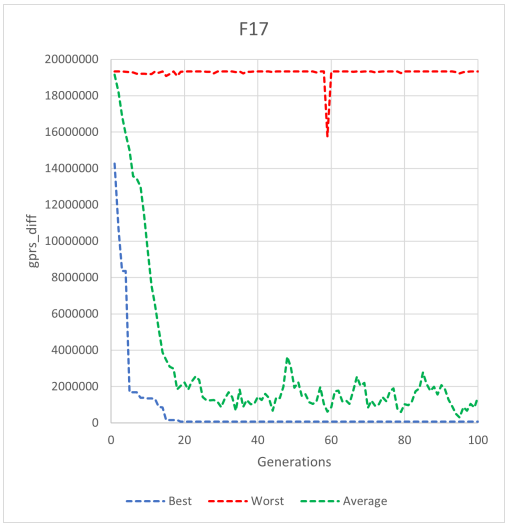


(e) 1000D-F_{16} CS-WR

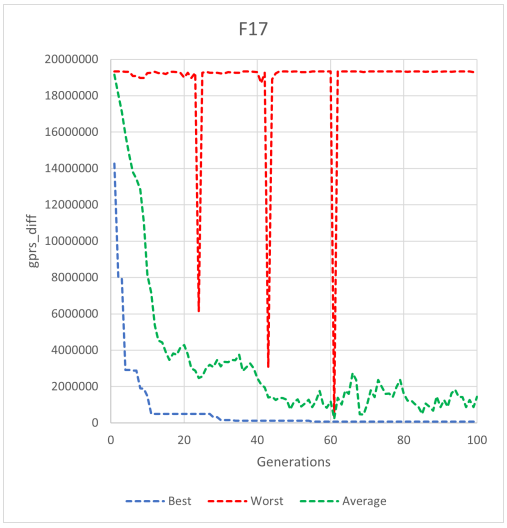


(f) 1000D-F_{16} CS-CR

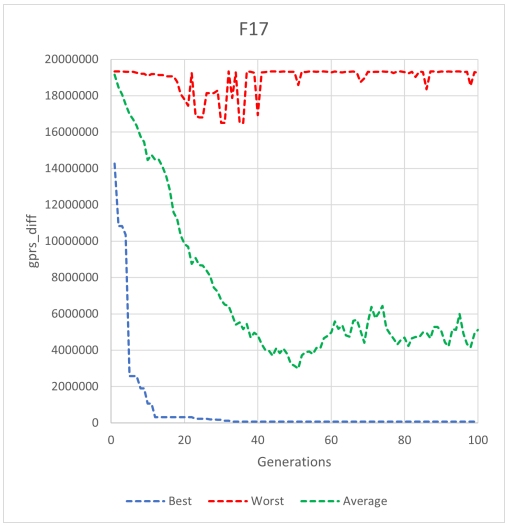
Figure 5.10: Convergence plots of function F_{16}, $D = 1000$ comparing the combinations of selection and replacement.



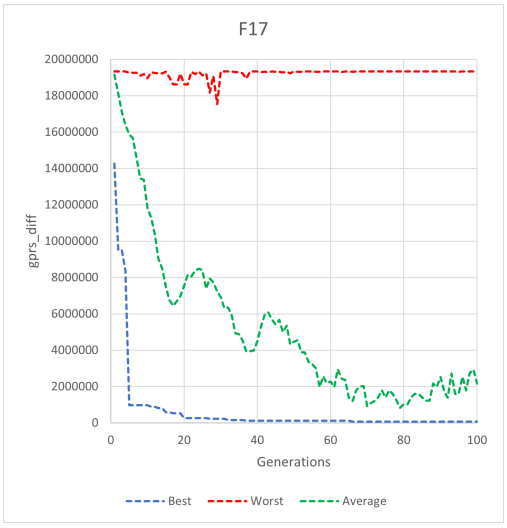
(a) 1000D-F_{17} RS-WR



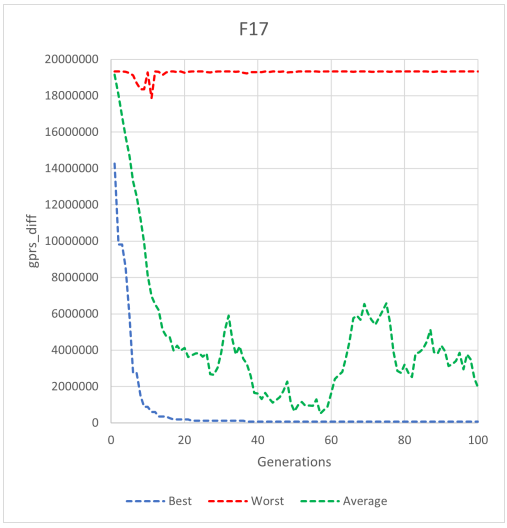
(b) 1000D-F_{17} RS-CR



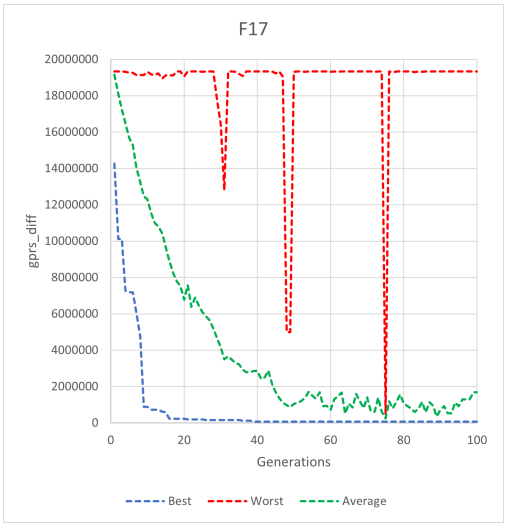
(c) 1000D-F_{17} TS-WR



(d) 1000D-F_{17} TS-CR

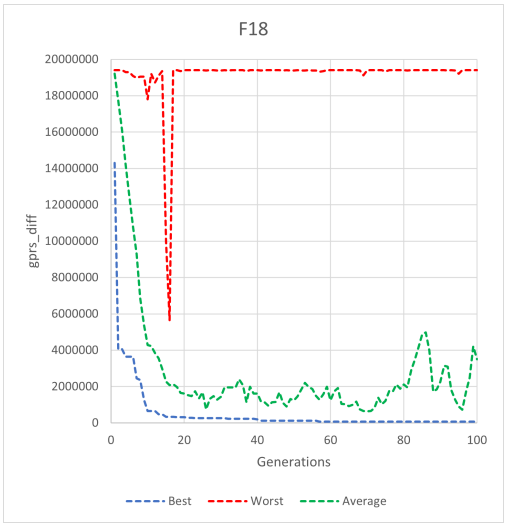


(e) 1000D-F_{17} CS-WR

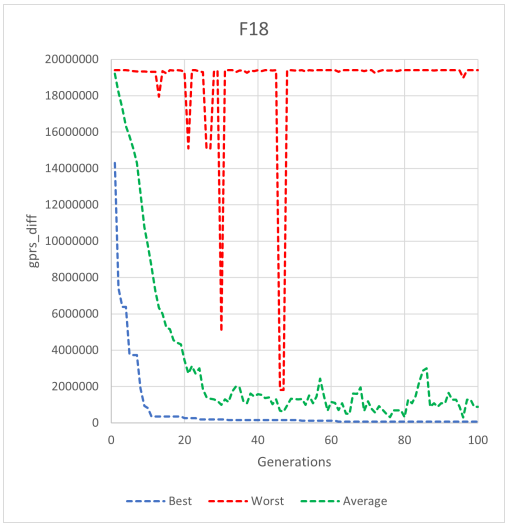


(f) 1000D-F_{17} CS-CR

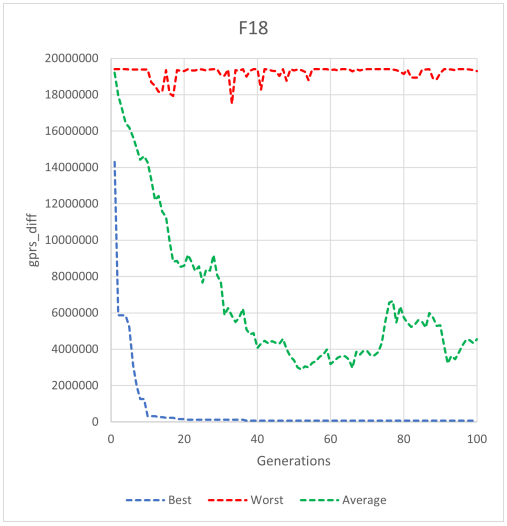
Figure 5.11: Convergence plots of function F_{17}, $D = 1000$ comparing the combinations of selection and replacement.



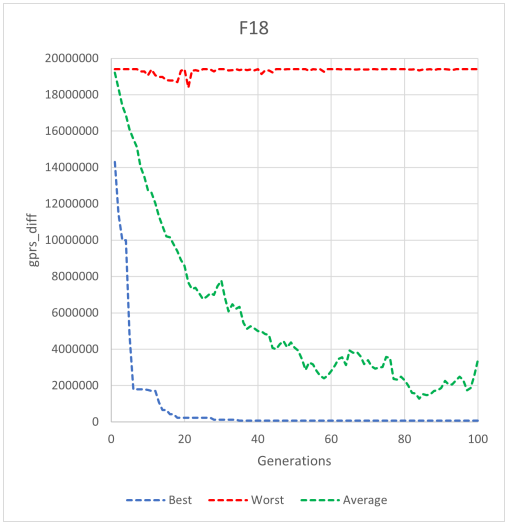
(a) 1000D-F_{18} RS-WR



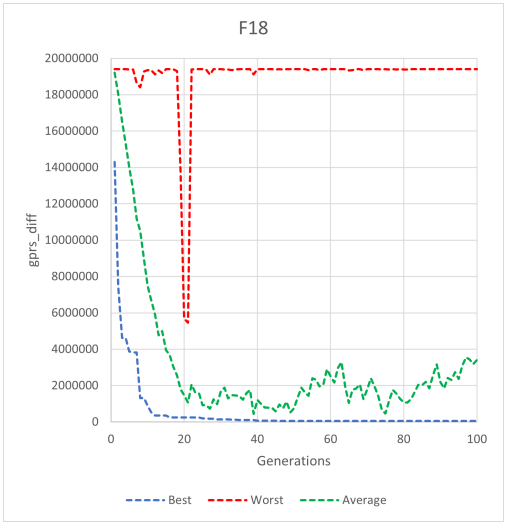
(b) 1000D-F_{18} RS-CR



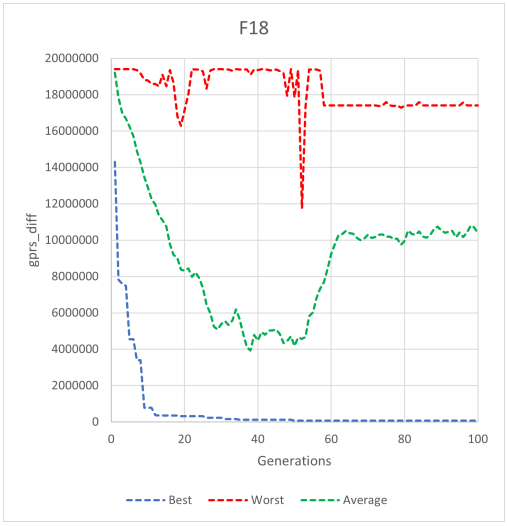
(c) 1000D-F_{18} TS-WR



(d) 1000D-F_{18} TS-CR



(e) 1000D-F_{18} CS-WR



(f) 1000D-F_{18} CS-CR

Figure 5.12: Convergence plots of function F_{18}, $D = 1000$ comparing the combinations of selection and replacement.

On the other hand, we repeat the experiments using the modified crossover operator first; this is experiment 7, and experiment 8 consists of testing the modified mutation operator.

5.2.7 Experiment 7

In this experiment, the performance of the modified crossover method is evaluated. Tables 5.25, 5.26, and 5.27 show the statistical results showing the best, median, and standard deviation of the 25 independent runs. The first three values are those reported by the original algorithm and the second by this new experiment. We see how, in the functions with one hundred variables, the results with the proposed crossover strategy seem to provide smaller fitness values. In the other cases, the results are lower.

Table 5.25: Statistical results of the modification of the GGA-VD crossover operator - 100D

Algorithm	**GGA-VD**			**GGA-cruza_rep_m**			
Function/Statistic	**BEST**	**MEDIAN**	**STD**	**BEST**	**MEDIAN**	**STD**	**W**
F16	1.79E+04	5.38E+04	1.72E+04	**2.98E-08**	5.13E+03	2.09E+03	×
F17	1.07E+05	4.28E+05	9.52E+04	**0.00E+00**	2.29E+05	9.93E+04	×
F18	1.63E+05	4.88E+05	1.40E+05	**0.00E+00**	6.59E+03	5.92E+03	×

Table 5.26: Statistical results of the modification of the GGA-VD crossover operator- 500D

Algorithm	**GGA-VD**			**GGA-cruza_rep_m**			
Function/Statistic	**BEST**	**MEDIAN**	**STD**	**BEST**	**MEDIAN**	**STD**	**W**
F16	**8.13E+02**	3.25E+03	7.04E+02	1.72E+03	3.73E+04	2.41E+04	✓
F17	**7.96E+04**	1.59E+05	1.75E+04	4.35E+05	5.56E+06	2.78E+06	✓
F18	**3.83E+05**	7.66E+05	1.46E+05	6.39E+05	7.94E+06	4.19E+06	✓

Table 5.27: Statistical results of the modification of the GGA-VD crossover operator- 1000D

Algorithm	**GGA-VD**			**GGA-cruza_rep_m**			
Function/Statistic	**BEST**	**MEDIAN**	**STD**	**BEST**	**MEDIAN**	**STD**	**W**
F16	4.07E+05	8.15E+05	1.27E+05	**1.98E+05**	4.88E+05	1.72E+05	×
F17	**4.32E+04**	8.63E+04	2.52E+04	3.94E+06	8.05E+06	2.49E+06	✓
F18	**4.20E+05**	8.41E+05	1.20E+05	1.01E+06	2.58E+06	7.83E+05	✓

According to the description of the change in the crossover operator, it is possible to create new groups when reinserting free elements and inserting them in the groups of the incomplete individual.

The Tables show that this does not significantly improve the decomposition results. This can be attributed to the fact that this change in the operator promotes decreasing the number of groups. Because a large percentage of the population is crossed, there is a risk of losing essential groups and not generating new ones since reinserting an element free in an existing group is higher than the probability of creating new groups.

5.2.8 Experiment 8

Here, the performance of the modified mutation method is evaluated. In Tables 5.28, 5.29, and 5.30, we have the statistical results showing the best, median, and standard deviation of the 25 independent runs. The first three values are those reported by the original algorithm, and the second ones are those reported by this new experiment.

Table 5.28: Statistical results of the modification of the GGA-VD mutation operator- 100D

Algorithm	**GGA-VD**			**GGA-muta_rep_m**			
Function/Statistic	**BEST**	**MEDIAN**	**STD**	**BEST**	**MEDIAN**	**STD**	**W**
F16	1.79E+04	5.38E+04	1.72E+04	**0.00E+00**	9.64E+04	1.00E+05	×
F17	1.07E+05	4.28E+05	9.52E+04	**0.00E+00**	1.57E+05	9.94E+04	×
F18	1.63E+05	4.88E+05	1.40E+05	**0.00E+00**	2.89E+05	1.39E+05	×

Table 5.29: Statistical results of the modification of the GGA-VD mutation operator- 500D

Algorithm	**GGA-VD**			**GGA-muta_rep_m**			
Function/Statistic	**BEST**	**MEDIAN**	**STD**	**BEST**	**MEDIAN**	**STD**	**W**
F16	8.13E+02	3.25E+03	7.04E+02	**0.00E+00**	2.18E+03	1.72E+03	×
F17	7.96E+04	1.59E+05	1.75E+04	**0.00E+00**	5.36E+01	2.97E+01	×
F18	3.83E+05	7.66E+05	1.46E+05	**7.84E+02**	1.57E+03	1.38E+03	×

Table 5.30: Statistical results of the modification of the GGA-VD mutation operator- 1000D

Algorithm	**GGA-VD**			**GGA-muta_rep_m**			
Function/Statistic	**BEST**	**MEDIAN**	**STD**	**BEST**	**MEDIAN**	**STD**	**W**
F16	4.07E+05	8.15E+05	1.27E+05	**0.00E+00**	1.64E+05	1.04E+05	×
F17	4.32E+04	8.63E+04	2.52E+04	**0.00E+00**	9.73E+01	6.12E+01	×
F18	4.20E+05	8.41E+05	1.20E+05	**2.70E+03**	5.40E+03	2.97E+03	×

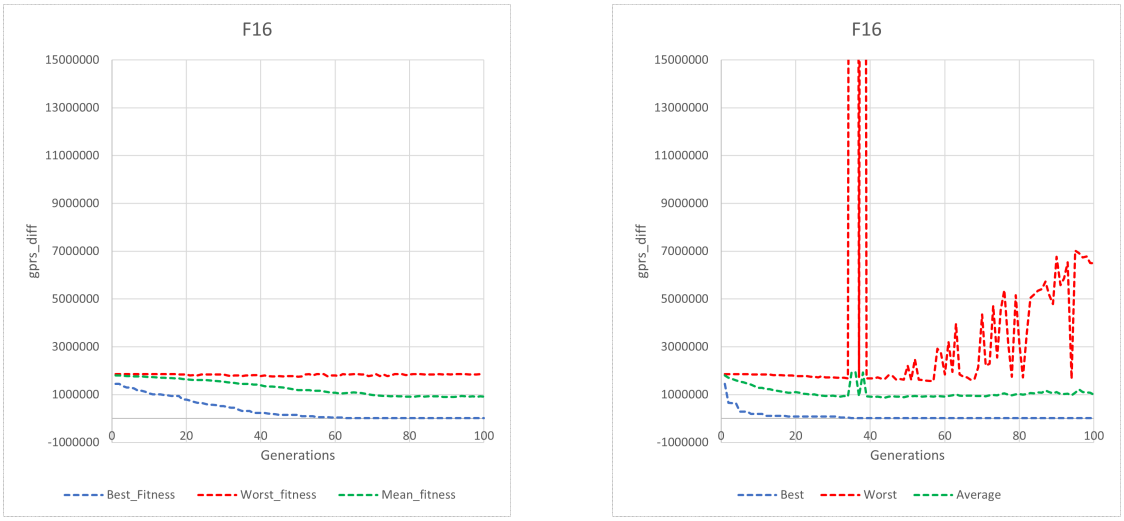
As with the crossover operator, the mutation operator allows free elements to be inserted into existing groups, allowing the number of groups to remain the same in the worst case. In the results tables, we can observe that this change in the operator generates evident improvements in the decomposition evaluation.

As part of the study, we made some convergence plots to analyze the algorithm's generational change. We describe the experiments over the three most complex functions of the set with $D = 100, 500$, and 1000.

We study the convergence through the hundred generations as subgraphs in a single graph. The first subplot shows the convergence of the algorithm that uses the proposed crossover method, which is the result of experiment 7. The second one is the experiment with the new mutation algorithm, i.e., experiment 8.

Figure 5.13 shows the graphs above for the F_{16} function with one hundred variables. We see how the modified crossover graph shows the blue graph of the best individual converges slowly.

On the other hand, in the graph of modified, we see how although the best individual seems to have a low fitness from the beginning, the red curve containing the flow of the worst individual shows extreme behavior with the peaks obtained in some generations. It seems that from that point on, the general behavior of the line is ascending with some fairly evident peaks.



(a) 100D-F_{16} modified crossover

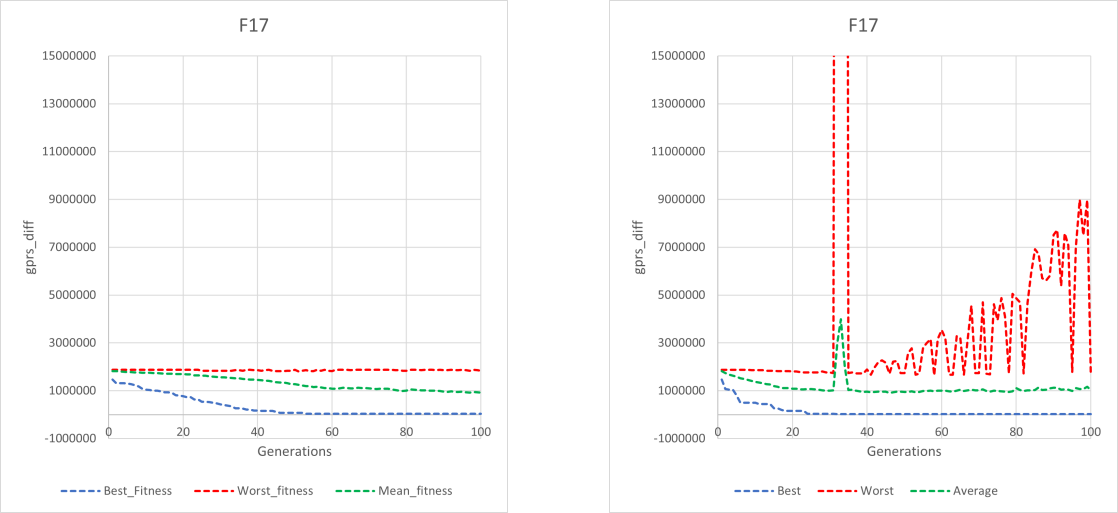
(b) 100D-F_{16} modified mutation

Figure 5.13: Convergence plots of function F_{16}, $D = 100$ comparing improved crossover and mutation operator.

Then, in Figure 5.14, we see a very similar performance in the function F_{17} compared to the function F_{16}, in this case, it seems that the best individual takes longer to improve if we look at the graph of modified crossover, where we see that

the blue curve takes longer to descend.

We also observe that in the graph of modified mutation, the red curve has an extreme peak, and there begin to be evident peaks after approximately 60 generations, these values are around $5E + 6$.



(a) 100D-F_{17} modified crossover

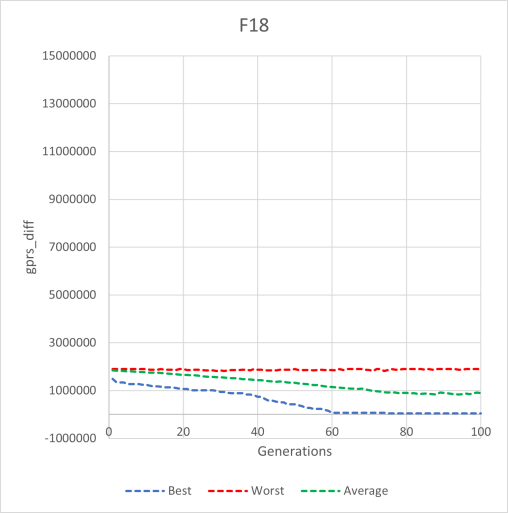
(b) 100D-F_{16} modified mutation

Figure 5.14: Convergence plots of function F_{17}, $D = 100$ comparing improved crossover and mutation operator.

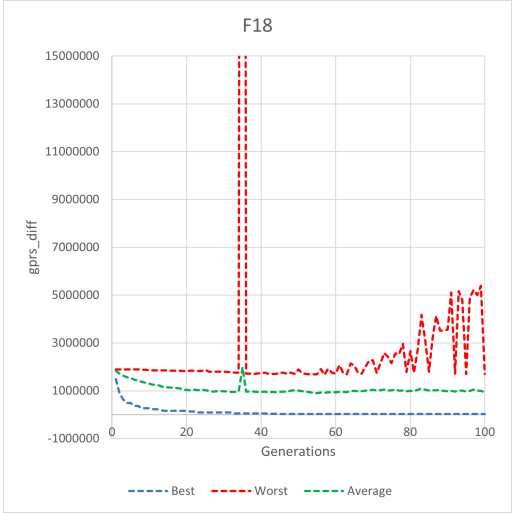
For function F_{18} in the graph in Figure 5.15, we have again behaviors like those of the two previous functions. This function's most complex convergence is evidently the slower convergence with the new crossover operator, which does not happen with the mutate operator.

We notice how the graphs of the two experiments, 7 and 8, perform similarly in the three chosen test functions.

Another important fact is the behavior of the average value in the three functions, which seems to vary very little throughout the iterations or generations of the algorithms, regardless of the operator, changing greatly only at the extreme peaks of the experiment using the new mutation operator.



(a) 100D-F_{18} modified crossover



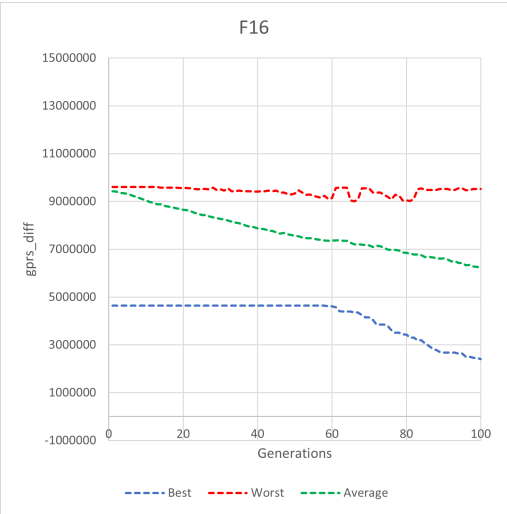
(b) 100D-F_{18} modified mutation

Figure 5.15: Convergence plots of function F_{18}, $D = 100$ comparing improved crossover and mutation operator.

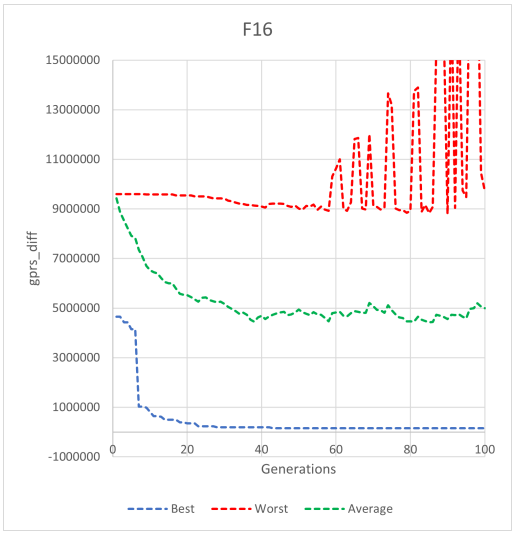
Now, we describe the graphs for the same three functions evaluated in 500 dimensions.

In the first function F_{16} (Figure 5.16), we see how the graph in blue is practically constant at the beginning of experiment 7 and then begins to descend, which is different in experiment 8 since the curve descends very fast there.

We can also see many peaks in the green and red curves corresponding to the average of the evaluations and the worst individual, respectively; in experiment 8, said values seem to be increasing. However, the best value is almost constant after 20 generations.



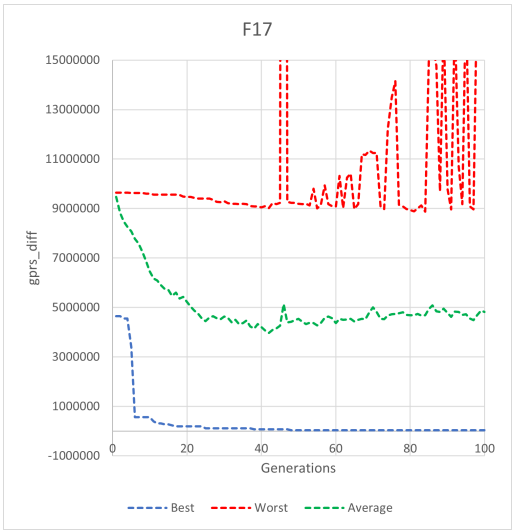
(a) 500D-F_{16} modified crossover



(b) 500D-F_{16} modified mutation

Figure 5.16: Convergence plots of function F_{16}, $D = 500$ comparing improved crossover and mutation operator.

(a) 500D-F_{17} modified crossover

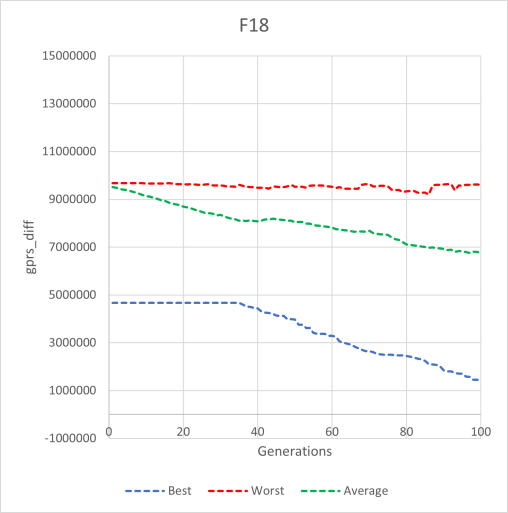


(b) 500D-F_{17} modified mutation

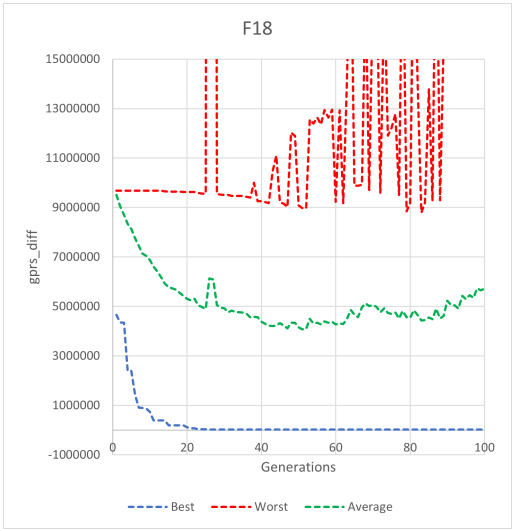
Figure 5.17: Convergence plots of function F_{17}, $D = 500$ comparing improved crossover and mutation operator.

In Figure 5.17, we observe the convergence of the F_{17} function in both experiments conducted at 500 dimensions (500D). The convergence behaves similarly to that of the F_{16} function during the initial generations and for about half of the total generations. In the modified crossover experiment, the convergence remains constant, while in the other experiment, it exhibits rapid convergence.

Similarly, we show the graphs for F_{18} evaluated at 500D in Figure 5.21,. The new crossover experiment does not show much variation with respect to the other functions. However, in experiment 8, which consists of the evaluation of the modified mutation operator, we see peaks from approximately 20 generations that are reflected in the average of the evaluations with changes in the form of small peaks.



(a) 500D-F_{18} modified crossover



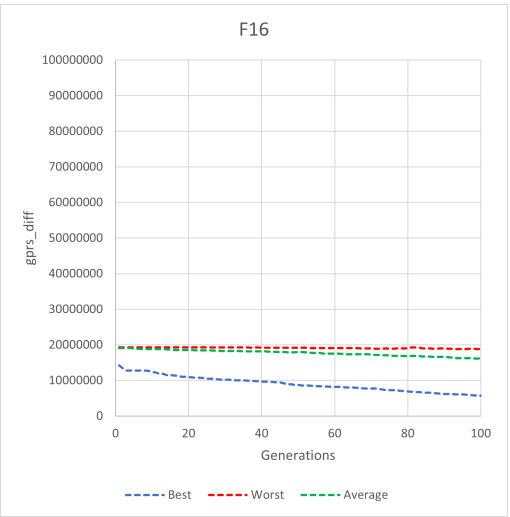
(b) 500D-F_{18} modified mutation

Figure 5.18: Convergence plots of function F_{18}, $D = 500$ comparing improved crossover and mutation operator.

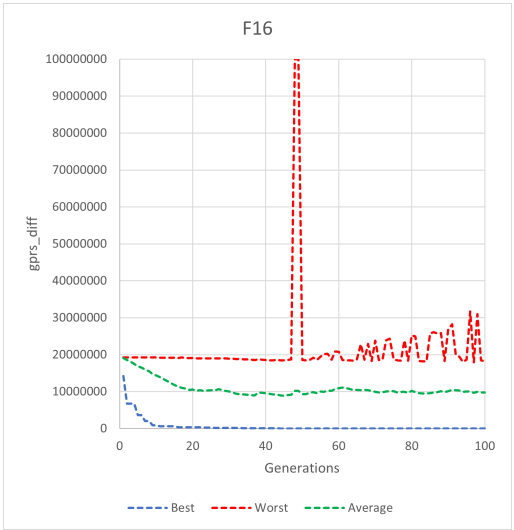
In general, in the three functions, it is noted that the new crossover operator does not seem to have a great influence on the convergence of the algorithm. However, the modified mutation operator causes obvious and important changes in the evaluations. However, it also harms the average evaluation since it seems not to allow its convergence.

Finally, we analyze the convergence of the three functions F_{16}, F_{17}, and F_{18}, which are now evaluated using 1000 variables over the course of a hundred generations. To facilitate a proper comparison, the scale on the y-axis is set equal for all three functions in both experiments, as extreme values were reached in some cases.

Figure 5.19 illustrates the convergence of the F_{16} function in both experiments. We notice that after 100 generations, the evaluation does not reach zero in the first experiment. In contrast, the second experiment shows distinct peaks and extreme values. Furthermore, the fitness values in both experiments remain quite high.



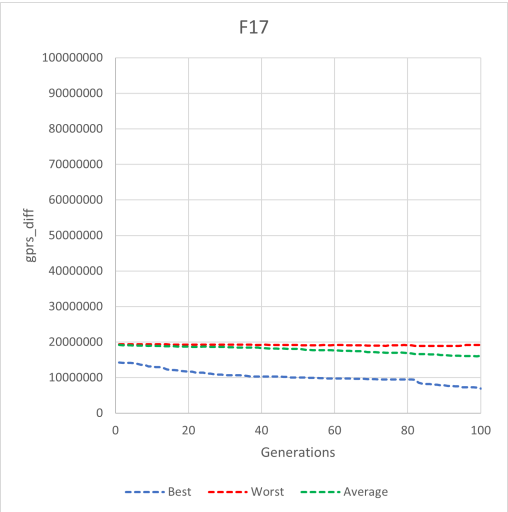
(a) 1000D-F_{16} modified crossover



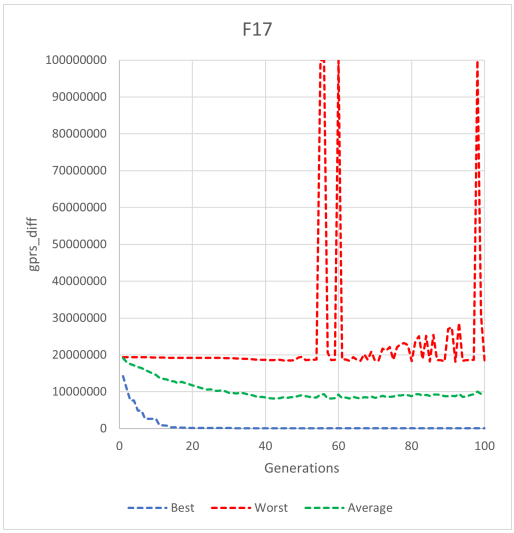
(b) 1000D-F_{16} modified mutation

Figure 5.19: Convergence plots of function F_{16}, $D = 1000$ comparing improved crossover and mutation operator.

The convergence of the function F_{17} is illustrated in Figure 5.20. In the case of the graph from experiment 7, we observe a different behavior compared to the other experiments. Notably, there is a decrease around generation 80, but this does not appear to affect the overall performance of the algorithm. Additionally, in the graph from experiment 8, which utilized a modified mutator operator, we again see peaks; however, the graphs for both the best and average performance do not show significant changes throughout the generations.

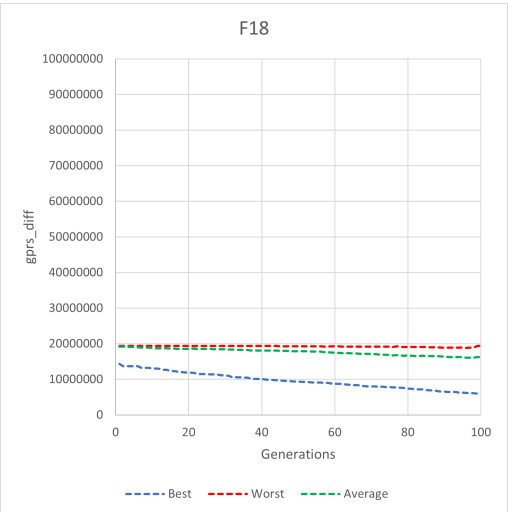


(a) 1000D-F_{17} modified crossover

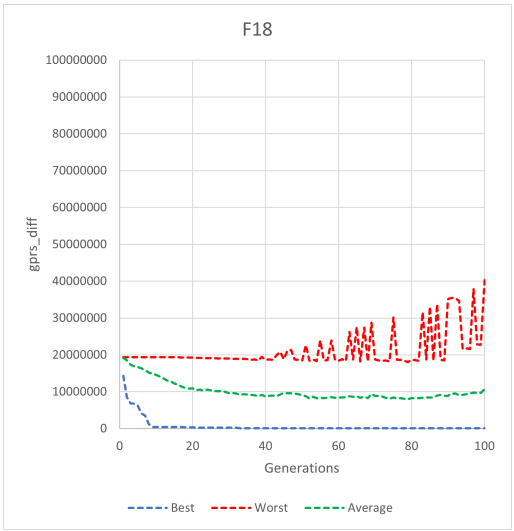


(b) 1000D-F_{17} modified mutation

Figure 5.20: Convergence plots of function F_{17}, $D = 1000$ comparing improved crossover and mutation operator.



(a) 1000D-F_{18} modified crossover



(b) 1000D-F_{18} modified mutation

Figure 5.21: Convergence plots of function F_{18}, $D = 1000$ comparing improved crossover and mutation operator.

Finally, for F_{18}, we have a similar performance of both versions of the algorithm. An interesting pattern is the rise of the peaks in the convergence graph of experiment 8 for the worst individual, which may be an indicator that the solutions could be decreasing in quality as the generations progress.

Based on the previous experiments and the observations made of each one, we have considered the results obtained and the convergence graphs. This tells us that experiment 2, roulette selection and controlled replacement, and experiment 3, tournament selection and replacement of the worst individuals, give the best combinations of selection and replacement. These experiments will now be tested with all the benchmark features to compare with the original version of the GGA.

5.3 Stage 3: Large-scale Optimization Experiments

After decomposing into sub-components, each must be optimized and cooperate to solve the problem completely. Various strategies have been used for the optimization process, including cooperative co-evolution and memetic algorithms, as said in Section 2.2.

In this section, we show optimization results for large-scale constrained problems, which were proposed in the work of Sayed et al. [54]. We seek to compare the performance of our proposal with the DVIIC strategy proposed in the work of Aguilar-Justo et al. [76], which we described in Section 3.2.

The experiments are divided as follows: First, the DVIIC decomposition is compared against the decomposition obtained by the GGA-VD by optimizing the test functions with a CC algorithm. Second, the same two decomposition strategies can be compared by optimizing using a memetic algorithm.

According to the results reported in the previous sections, we chose the TS-WR-IM version of the GGA-VD algorithm, i.e., GGA-VD with tournament selection, worst replacement, and improved mutation operator, a version of the GGA-VD that presents some of the best results. Therefore, this version is used for the decomposition applied to each optimization algorithm.

The parameters were set similarly for a fair comparison and based on the literature:

- CC algorithm: $maxFEs = 7,5e+6$,
- Memetic algorithm: $maxFEs = 7,5e+6$
- DE: $F = 0,5$, $CR = 0,9$ $NP = 50$.

Tables 5.31 - 5.36 show the results obtained from the abovementioned experiments. There are two tables for each dimension value. The first table compares the two decomposition strategies using the same memetic algorithm for optimization, and the second compares them using CC.

Each table shows the best value obtained, the median, and the standard deviation of the results obtained from 25 independent runs. The best results are highlighted in bold. The last column in each table shows whether or not significant differences are using the Wilcoxon test at a 95% confidence level (W column). If the significant differences favor the algorithm with GGA-VD, a checkmark is shown; if they are against our proposal, a mark is shown; and if there are no significant differences between the two algorithms, the equals sign is displayed.

In Table 5.31, we can see the results obtained by the memetic algorithm for the problems with 100 variables. We can observe that the algorithm using the GGA-VD decomposition achieves the best results in most cases. The other version using the DVIIC algorithm only obtains smaller fitness values in two functions, F_7 and F_{17}.

On the other hand, the results obtained by the CC algorithm using both decomposition methods for the problems with 100 variables are reported in Table 5.32. We can observe a similar behavior to the memetic algorithm, with smaller fitness values in 15 of the 18 functions when the GGA-VD decomposition is used.

Table 5.31: Memetic algorithm - 100D

	DELoCos-GGA			DELoCos-DVIIC			
	Best	**Median**	**Std**	**Best**	**Median**	**Std**	**W**
F1	**0.00E+00**	0.00E+00	0.00E+00	0.00E+00	0.00E+00	0.00E+00	=
F2	**1.24E+00**	3.45E+00	9.80E-01	2.41E+00	6.78E+00	2.43E+00	✓
F3	**1.30E+00**	1.56E+00	3.00E-01	2.87E+00	2.98E+00	5.40E-01	✓
F4	**8.00E-01**	1.34E+00	3.27E-01	1.45E+00	5.77E+00	2.91E+00	✓
F5	**4.30E-02**	4.53E+00	1.23E+00	1.27E+00	8.56E+00	3.66E+00	✓
F6	**2.34E+00**	5.46E+00	1.56E+00	5.44E+00	7.23E+00	9.90E-01	✓
F7	5.60E+00	7.60E+00	1.78E+00	**4.56E+00**	5.78E+00	1.82E+00	×
F8	**5.34E-02**	1.94E+00	9.80E-01	1.07E+00	2.19E+00	1.55E+00	✓
F9	**7.90E-01**	2.30E-01	7.60E-01	5.67E+00	1.02E+01	3.32E+00	✓
F10	**2.40E-01**	1.20E+00	8.80E-01	9.80E-01	3.45E+00	1.53E+00	✓
F11	**3.40E-02**	1.56E+00	1.38E+00	1.23E+00	6.54E+00	3.77E+00	✓
F12	**3.20E-02**	2.20E-01	5.20E-01	2.34E+00	3.27E+00	1.21E+00	✓
F13	**5.60E-01**	1.23E+00	4.50E-01	1.30E-02	1.45E+00	6.60E-01	✓
F14	**3.40E-04**	1.99E+00	1.13E+00	2.01E+00	4.98E+00	2.33E+00	✓
F15	**2.56E-01**	2.45E+00	9.88E-01	1.20E+00	5.49E+00	1.04E+00	✓
F16	**1.76E+00**	3.78E+00	2.43E+00	3.66E+00	2.05E+01	8.54E+00	✓
F17	2.35E+01	7.63E+01	4.65E+00	**1.82E+01**	5.41E+01	1.93E+01	×
F18	**1.01E+02**	5.67E+02	2.49E+02	2.34E+02	8.92E+02	3.87E+02	✓

Table 5.32: CC algorithm - 100D

	CC-GGA			CC-DVIIC			
	Best	**Median**	**Std**	**Best**	**Median**	**Std**	**W**
F1	**0.00E+00**	0.00E+00	0.00E+00	0.00E+00	0.00E+00	0.00E+00	=
F2	**9.80E-01**	1.45E+00	7.60E-01	1.78E+00	3.21E+00	1.40E+00	✓
F3	**3.40E-01**	1.05E+00	7.80E-01	1.22E+00	4.65E+00	1.32E+00	✓
F4	4.50E-01	9.40E-01	5.60E-01	**3.20E-02**	7.60E-01	2.46E-01	×
F5	**6.70E-01**	7.81E+00	3.45E+00	1.48E+00	2.56E+00	8.84E-01	✓
F6	**3.54E+00**	5.44E+00	1.20E+00	8.95E+00	1.62E+01	5.64E+00	✓
F7	**2.80E-01**	2.36E+00	6.50E-01	1.26E+00	6.34E+00	2.87E+00	✓
F8	**5.60E-04**	1.36E+00	4.66E-01	2.64E+00	6.46E+00	3.84E+00	✓
F9	**2.50E-01**	2.45E+00	1.66E+00	2.21E+00	6.58E+00	1.58E+00	✓
F10	**6.40E-03**	2.36E+00	1.55E+00	2.10E-01	1.98E+00	8.40E-01	✓
F11	4.50E-01	4.81E+00	2.65E+00	**5.00E-03**	4.96E+00	3.45E+00	×
F12	**6.00E-03**	2.06E+00	1.65E+00	1.56E+00	5.64E+00	2.88E+00	✓
F13	**8.70E-03**	2.60E-01	1.68E-02	8.40E-01	1.68E+00	6.80E-01	✓
F14	**5.79E-01**	6.54E+00	2.69E+00	6.58E+00	1.07E+01	6.54E+00	✓
F15	**5.78E-03**	2.14E-01	1.67E-01	5.48E+00	6.58E+00	9.80E-01	✓
F16	**6.87E-01**	4.99E+00	3.87E+00	7.89E+00	1.76E+01	6.49E+00	✓
F17	**8.46E+01**	1.03E+02	1.07E+01	1.30E+02	2.38E+02	5.50E+01	✓
F18	3.98E+02	8.96E+02	6.45E+02	**2.56E+02**	5.68E+02	3.22E+02	×

Now, we show the results for the test problems with 500 variables. First, Table 5.33 contains the results for the 18 functions with both algorithms implemented with the two decomposition strategies. In this experiment, we observed that in a similar way to the experiment with 100 variables, our De-LoCoS-GGA obtained the smallest best, median, and standard deviation values in most cases, except functions F_6, F_{10}, F_{11} y F_{14} where the memetic algorithm using DVIIC is better. In the first two functions, both algorithms obtained zeros in the three statistics presented, so it cannot be said that there are significant differences.

Table 5.33: Memetic algorithm - 500D

500D	**DELoCos-GGA**			**DELoCos-DVIIC**			
	Best	**Median**	**Std**	**Best**	**Median**	**Std**	**W**
F1	**0.00E+00**	0.00E+00	0.00E+00	**0.00E+00**	0.00E+00	0.00E+00	=
F2	**0.00E+00**	0.00E+00	0.00E+00	**0.00E+00**	0.00E+00	0.00E+00	=
F3	**0.00E+00**	0.00E+00	0.00E+00	3.56E-04	3.40E-02	1.39E-03	✓
F4	**0.00E+00**	0.00E+00	0.00E+00	8.74E-03	1.20E-01	5.64E-02	✓
F5	**2.50E-02**	7.89E-01	3.57E-01	1.20E+00	1.56E+00	5.89E-01	✓
F6	1.22E+01	1.55E+01	2.48E+00	**7.15E+00**	2.86E+01	1.07E+01	×
F7	**5.62E+01**	1.29E+02	1.53E+01	2.34E+02	1.56E+03	2.68E+02	✓
F8	**4.56E+01**	6.59E+01	1.16E+01	8.96E+01	1.01E+02	5.64E+00	✓
F9	**2.36E+01**	7.92E+01	3.14E+01	7.54E+01	9.63E+01	6.45E+00	✓
F10	5.60E+01	9.92E+01	1.05E+01	**4.50E+01**	1.01E+02	4.02E+01	×
F11	7.12E+01	7.86E+01	3.45E+00	**2.56E+00**	8.96E+01	2.34E+01	×
F12	**1.28E+01**	4.83E+01	2.05E+01	9.94E+01	1.47E+02	3.14E+01	✓
F13	**1.65E+01**	8.42E+01	3.13E+01	3.56E+01	5.21E+01	1.64E+01	✓
F14	6.15E+01	1.56E+02	3.18E+01	**2.92E+01**	1.75E+02	4.58E+01	×
F15	**5.61E+02**	7.82E+02	9.92E+01	9.12E+02	1.03E+03	2.18E+02	✓
F16	**1.27E+02**	9.43E+02	3.51E+02	1.11E+03	1.57E+03	9.05E+01	✓
F17	**1.56E+02**	6.75E+02	1.49E+02	1.87E+03	2.16E+03	9.73E+01	✓
F18	**9.84E+02**	1.02E+03	8.43E+01	3.10E+03	3.45E+03	2.01E+02	✓

The results of the following experiment that uses CC for problems with 500 variables are presented in Table 5.34. Similarly, smaller best, median, and standard deviation values are obtained for most functions using the GGA-VD decomposition.

Table 5.34: CC algorithm - 500D

500D	CC-GGA			CC-DVIIC			
	Best	**Median**	**Std**	**Best**	**Median**	**Std**	**W**
F1	**0.00E+00**	0.00E+00	0.00E+00	**0.00E+00**	0.00E+00	0.00E+00	=
F2	**0.00E+00**	0.00E+00	0.00E+00	**0.00E+00**	0.00E+00	0.00E+00	=
F3	**0.00E+00**	0.00E+00	0.00E+00	**0.00E+00**	0.00E+00	0.00E+00	=
F4	**2.00E-02**	1.50E-01	6.00E-02	8.40E-01	9.20E-01	2.00E-02	✓
F5	**4.80E-03**	6.20E-03	2.74E-02	1.84E-01	5.64E-01	9.40E-02	✓
F6	**5.46E+00**	8.56E+00	1.78E+00	1.49E+01	5.96E+01	1.28E+01	✓
F7	**6.12E+00**	1.42E+01	3.51E+00	8.41E+01	8.65E+01	2.10E+00	✓
F8	**1.24E+01**	3.15E+01	1.47E+01	5.17E+01	6.94E+01	9.51E+00	✓
F9	**3.61E+01**	8.41E+01	2.04E+01	9.56E+01	1.21E+02	9.47E+00	✓
F10	**2.18E+01**	4.96E+01	1.37E+01	7.45E+01	9.16E+01	1.23E+01	✓
F11	7.56E+01	9.43E+01	9.47E+00	**5.48E+01**	6.92E+01	8.40E+00	×
F12	**5.61E+01**	8.45E+01	1.24E+01	7.12E+01	8.23E+01	5.45E+00	✓
F13	**1.28E+01**	5.42E+01	2.08E+01	4.52E+01	9.91E+01	1.86E+01	✓
F14	**2.17E+01**	4.59E+01	1.96E+01	4.84E+01	5.64E+01	7.67E+00	✓
F15	**5.47E+02**	7.41E+02	9.40E+01	6.61E+02	7.51E+02	8.45E+01	✓
F16	**3.14E+02**	4.82E+02	6.86E+01	4.57E+02	6.21E+02	1.02E+02	✓
F17	8.45E+02	9.22E+02	4.96E+01	**4.77E+02**	1.59E+03	5.41E+02	×
F18	**1.26E+02**	6.41E+02	2.87E+02	3.33E+02	5.61E+02	9.94E+01	✓

For the case of the experiment with the DE-LoCoS algorithm using the functions evaluated in a thousand variables, we have Table 5.35, in which we can observe how smaller values of fitness function are obtained with the decomposition through the GGA-VD in most of the functions, speaking of the best value found in the independent runs. The six functions that have better performance with the DVIIC decomposition are F_6, F_{10}, F_{11} and F_{14}. In functions F_1 and F_2, the same results are obtained for both decompositions.

Table 5.35: Memetic algorithm - 1000D

1000D	**DELoCos-GGA**			**DELoCos-DVIIC**			
	Best	**Median**	**Std**	**Best**	**Median**	**Std**	**W**
F1	**0.00E+00**	0.00E+00	0.00E+00	**0.00E+00**	0.00E+00	0.00E+00	=
F2	**0.00E+00**	0.00E+00	0.00E+00	**0.00E+00**	0.00E+00	0.00E+00	=
F3	4.56E-05	8.40E-04	9.00E-04	**0.00E+00**	0.00E+00	0.00E+00	×
F4	8.40E-01	1.95E+01	5.48E+00	**1.89E+00**	5.56E+00	2.16E+00	✓
F5	**1.26E+01**	1.59E+01	1.25E+00	1.87E+01	5.62E+01	1.98E+01	✓
F6	**9.45E+00**	4.13E+01	1.54E+01	1.62E+01	8.16E+01	3.07E+01	✓
F7	**2.14E+00**	5.42E+01	1.67E+01	1.94E+01	9.12E+01	2.58E+01	✓
F8	**7.15E+00**	2.34E+01	5.16E+00	9.15E+01	1.87E+02	5.13E+01	✓
F9	**9.56E+01**	5.10E+02	1.84E+02	2.12E+02	2.56E+02	1.46E+01	✓
F10	**6.63E+01**	1.58E+02	4.96E+01	9.18E+01	1.64E+02	2.31E+01	✓
F11	**7.26E+01**	1.08E+02	1.95E+01	9.43E+01	2.13E+02	3.84E+01	✓
F12	5.51E+01	9.62E+01	1.21E+01	**4.86E+01**	1.25E+02	2.99E+01	×
F13	**1.26E+02**	4.20E+02	1.56E+02	4.01E+02	4.75E+02	2.32E+01	✓
F14	**2.07E+02**	2.63E+02	1.85E+01	2.64E+02	4.51E+02	9.95E+01	✓
F15	7.01E+02	9.14E+02	9.12E+01	**5.99E+02**	8.41E+02	7.92E+01	×
F16	**1.03E+03**	2.48E+03	5.46E+02	2.85E+03	3.18E+03	5.61E+02	✓
F17	**8.44E+02**	2.32E+03	6.77E+02	2.13E+03	2.95E+03	3.15E+01	✓
F18	**3.64E+03**	9.15E+03	2.89E+03	5.21E+03	7.45E+03	9.94E+02	✓

For the case of CC using both decompositions the results are in Table 5.36, we observed a higher count of best evaluations using GGA as the decomposition algorithm. In five functions of the total functions, the version that uses CC is better, and in the first three functions they remain the same with a fitness value of zero. It is necessary to emphasize how the median is relatively higher in most of the functions when DVIIC is used.

Table 5.36: CC algorithm - 1000D

1000D	CC-GGA			CC-DVIIC			
	Best	**Median**	**Std**	**Best**	**Median**	**Std**	**W**
F1	**0.00E+00**	0.00E+00	0.00E+00	**0.00E+00**	0.00E+00	0.00E+00	=
F2	**0.00E+00**	0.00E+00	0.00E+00	**0.00E+00**	0.00E+00	0.00E+00	=
F3	**0.00E+00**	0.00E+00	0.00E+00	**0.00E+00**	0.00E+00	0.00E+00	=
F4	3.32E-01	1.44E+00	2.00E-01	**1.00E-04**	4.50E-01	2.31E-02	×
F5	**2.34E+00**	5.46E+00	1.56E+00	3.26E+01	1.20E+02	8.42E+01	✓
F6	6.53E+01	3.20E+02	1.83E+02	**1.28E+01**	3.63E+01	2.24E+01	×
F7	**6.39E+01**	1.04E+02	1.24E+01	2.42E+02	9.34E+02	3.72E+02	✓
F8	**7.56E+02**	8.34E+02	1.27E+02	8.23E+02	1.56E+03	4.38E+02	✓
F9	**1.26E+02**	6.44E+02	3.81E+02	6.39E+02	9.13E+02	2.35E+01	✓
F10	**2.21E+02**	5.48E+02	3.22E+02	3.62E+02	4.56E+02	8.40E+01	✓
F11	**8.90E+01**	1.84E+02	3.67E+01	2.21E+02	3.45E+02	1.27E+02	✓
F12	**2.11E+02**	7.62E+02	2.86E+02	7.66E+02	9.23E+02	9.25E+01	✓
F13	9.34E+01	6.45E+02	4.10E+02	**7.49E+01**	3.22E+02	1.05E+02	×
F14	**6.83E+01**	7.65E+01	3.10E+00	1.74E+02	2.43E+02	6.28E+01	✓
F15	3.91E+02	9.34E+02	2.01E+02	**1.26E+02**	3.48E+02	5.60E+01	×
F16	**7.67E+02**	1.03E+03	9.95E+01	1.48E+03	1.87E+03	2.31E+02	✓
F17	**5.33E+02**	7.83E+02	8.64E+01	9.23E+02	2.01E+03	5.64E+02	✓
F18	3.42E+03	4.53E+03	9.86E+01	**2.68E+03**	6.22E+03	1.85E+02	×

5.4 Stage 4: Characterization

Based on the above results, we can confirm at first glance that our algorithm for decomposition works better than the other algorithms studied. Part of this work also consists of determining the problems that were more difficult to solve for both the decomposition and optimization algorithms. This, in turn, allows us to improve both algorithms and contribute to generating knowledge so that other large-scale optimization problems can also benefit from this decomposition strategy.

Table 5.37: Function and constraints characteristics

Function	**Dim**	**Obj-Separability**	**# constraints**
F1	100, 500, 1000	Highly separable	1
F2	100, 500, 1000	Highly separable	2
F3	100, 500, 1000	Highly separable	3
F4	100, 500, 1000	Partially separable - spliced	1
F5	100, 500, 1000	Partially separable - spliced	2
F6	100, 500, 1000	Partially separable - spliced	3
F7	100, 500, 1000	Partially separable - spliced	1
F8	100, 500, 1000	Partially separable - spliced	2
F9	100, 500, 1000	Partially separable - spliced	3
F10	100, 500, 1000	Partially separable - overlapping	1
F11	100, 500, 1000	Partially separable - overlapping	2
F12	100, 500, 1000	Partially separable - overlapping	3
F13	100, 500, 1000	Partially separable - spliced and overlapping	1
F14	100, 500, 1000	Partially separable - spliced and overlapping	2
F15	100, 500, 1000	Partially separable - spliced and overlapping	3
F16	100, 500, 1000	Nonseparable - spliced and overlapping	1
F17	100, 500, 1000	Nonseparable - spliced and overlapping	2
F18	100, 500, 1000	Nonseparable - spliced and overlapping	3

To do this, we first summarize the functions and their main characteristics in Table 5.37 to identify if there is a pattern or relationship with the obtained results.

Then, in the Tables 5.38, 5.39, and 5.40, we show which algorithm obtained the best optimization results for each function in each dimension. For this, each row is represented by a function, and the columns represent the four methods: first, the memetic one with both DELoCoS-GGA and DELoCoS-DVIIC decomposition methods, then the two methods focused on Cooperative Coevolution: CC-GGA and CC-DVIIC. The cells marked with "x" indicate that that algorithm (column) was the best in that function (row), and at the end of each column, a total is shown, which represents the number of times that the corresponding algorithm obtained

that result.

Table 5.38: Count of the algorithms with the best results for each of the functions in 100D

Function/Algorithm	DELoCos-GGA	DELoCos-DVIIC	CC-GGA	CC-DVIIC
F1	×	×	×	×
F2			×	
F3			×	
F4				×
F5	×			
F6			×	
F7			×	
F8			×	
F9			×	
F10			×	
F11				×
F12			×	
F13			×	
F14	×			
F15			×	
F16			×	
F17				×
F18	×			
Total	4	1	12	4

First, in Table 5.38, which summarizes the functions evaluated in 100 dimensions, we can observe that the algorithm with the best results was CC-GGA, with a frequency of 12, followed by the algorithms DELoCoS-GGA and CC-DVIIC with 4. No trend is observed in the functions, only that the last two, F_{17} and F_{18}, which are the most difficult, were solved by CC-DVIIC and DELoCoS-GGA, respectively. Another factor is that DELoCoS-DVIIC was not better in any of the cases since the other algorithms also solved the function that it solved optimally.

Table 5.39: Count of the algorithms with the best results for each of the functions in 500D

Function/Algorithm	DELoCos-GGA	DELoCos-DVIIC	CC-GGA	CC-DVIIC
F1	×	×	×	×
F2	×	×	×	×
F3	×		×	×
F4	×			
F5			×	
F6			×	
F7			×	
F8			×	
F9			×	
F10			×	
F11		×		
F12	×			
F13		×		
F14			×	
F15			×	
F16	×			
F17	×			
F18			×	
Total	7	4	12	3

In Table 5.39, we have a summary of the functions in 500D. Again, CC-GGA obtains the highest score with 12 functions, followed by DELoCoS-GGA with 7. As in the experiments with 100 variables, we have that in the functions F_4, F_{11}, and F_{17}, CC-GGA is not the best. In this case, the algorithm with the least best results was CC-DVIIC with 3 functions, two of which were solved satisfactorily by the other three methods.

Table 5.40: Count of the algorithms with the best results for each of the functions in 1000D

Function/Algorithm	DELoCos-GGA	DELoCos-DVIIC	CC-GGA	CC-DVIIC
F1	×	×	×	×
F2	×	×	×	×
F3			×	×
F4				×
F5			×	
F6	×			
F7	×			
F8	×			
F9	×			
F10	×			
F11	×			
F12		×		
F13				×
F14			×	
F15				×
F16			×	
F17			×	
F18				×
Total	8	3	7	7

Finally, the results of the functions in 1000D are shown in Table 5.40. There, we see a similar frequency in three of the four methods studied, with an advantage of one for DELoCoS-GGA. The four algorithms solved the most straightforward functions, F_1 and F_2, with zero in the evaluation value, and CC-GGA failed again in the solution of F_4 and F_{11}. It is also important to highlight how the CC-DVIIC algorithm solves three of the most complicated functions of the benchmark in this dimension.

With the frequencies in the tables above, we can observe that CC-GGA and DELoCoS-GGA are better overall. However, it is also necessary to highlight the per-

formance of DELoCoS-DVIIC, which was very competent in problems with higher dimensions, standing out those functions with more complex separability characteristics.

Another study that we considered important was a correlation analysis between the results obtained by the GGA-VD in the decomposition step (Equation 4.5) and by the optimization algorithm. For this, the numerical data of $grps_{diff}$ of the 18 functions were first taken, in this case, only evaluated in 1000 variables. They were compared against their respective 18 values obtained by the optimization algorithm.

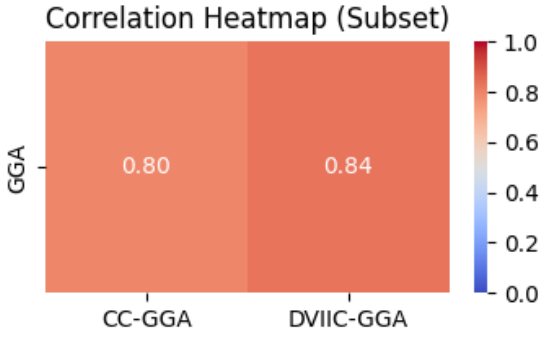


Figure 5.22: General correlation between the results obtained by the GGA-VD and the optimization approaches

The heatmap shown in Figure 5.22 contains the correlation coefficients between both optimization algorithms and the variable decomposition results. We see a correlation coefficient of 0.80 between CC-GGA and GGA-VD, while between DELoCoS-GGA and GGA-VD, there is a correlation coefficient of 0.84. Both coefficients are relatively high, indicating a high linear relationship between the decomposition and the algorithms. In other words, a good decomposition leads to a good optimization result and vice versa.

On the other hand, the correlations between the results of the 25 independent runs concerning the $grps_{diff}$ value of Equation 4.5 and the results of the optimization algorithms were analyzed to reinforce the conclusions about the relationship. Due to the handling of fixed seeds of random numbers and the values of $grps_{diff}$

equal to zero in the decomposition, the correlations are studied in the functions from F_{11} to F_{18} except F_{15} where $grps_{diff}$ is also zero. The graphs in Figures 5.23 and 5.24 show the correlation coefficients between the results of the GGA-VD and the CC-GGA and DE-LoCoS-GGA algorithms, respectively.

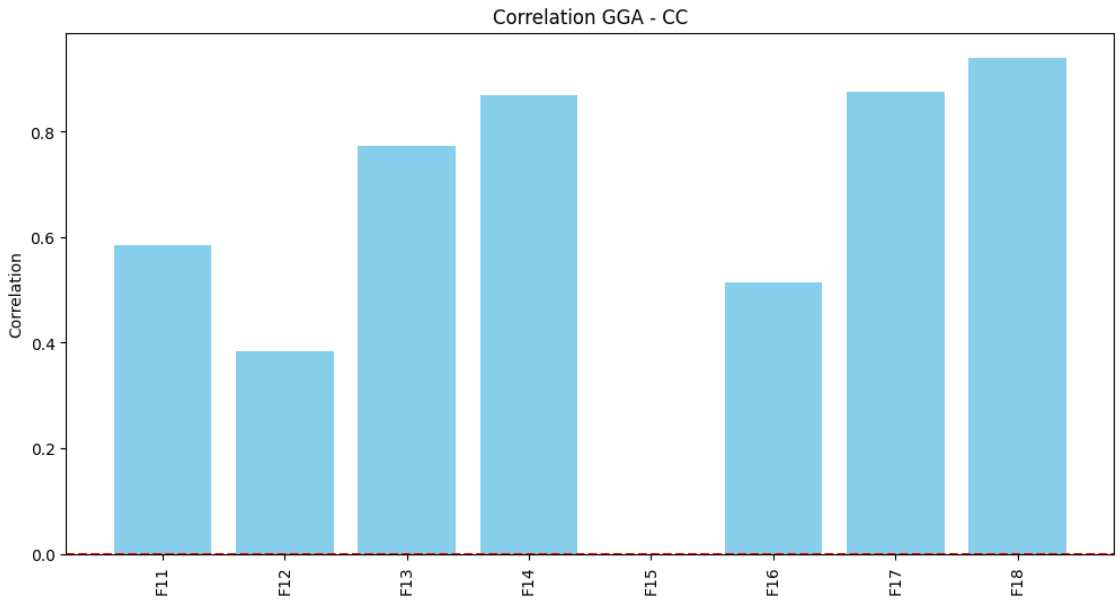


Figure 5.23: Correlation between the results obtained by the GGA-VD and those obtained by Cooperative Co-Evolution per function

In the first Figure (5.23), it is observed that the correlation coefficients are mostly greater than 0.5, which indicates a significant linear relationship between the results of both algorithms (GGA-VD and CC-GGA). The highest coefficient is found in the function F_{18}, which has non-separable characteristics and three constraints.

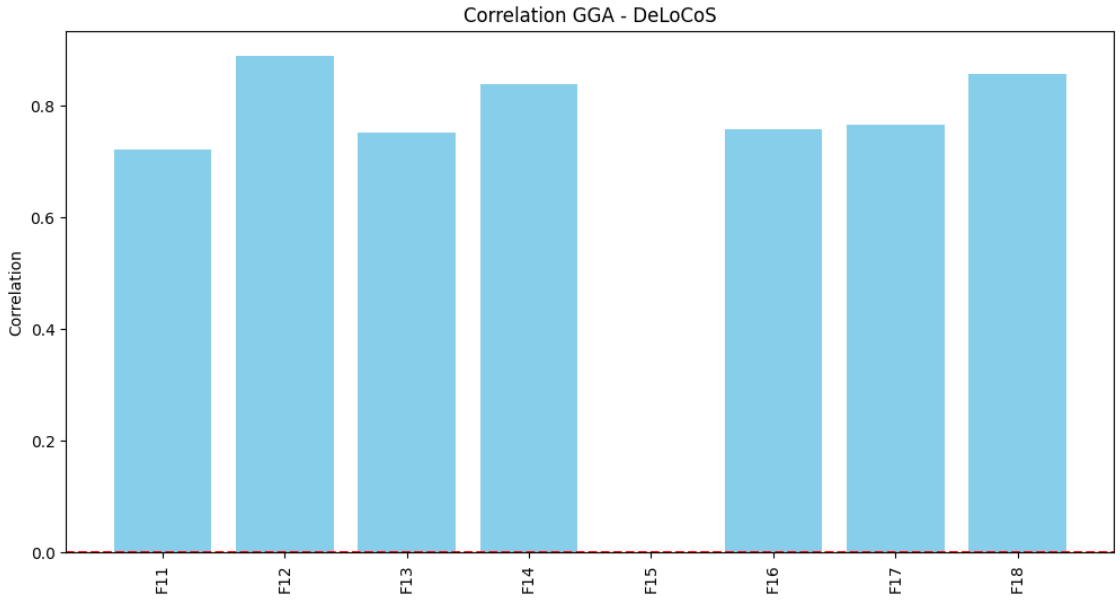


Figure 5.24: Correlation between the results obtained by the GGA-VD and those obtained by DE-LoCoS per function

The second Figure (5.24) shows the correlation coefficients between GGA-VD and DELoCoS-GGA. In this case, the coefficients are greater than 0.7, which indicates an even stronger linear relationship than with CC-GGA. In other words, good decompositions lead to good optimization results and vice versa.

With the tables and graphs above, we reinforce the hypothesis that a group-based variable decomposition algorithm can help improve optimization results, and it is confirmed that there is indeed a high linear relationship between the values returned by the decomposition algorithm and the numerical values of the optimizer.

Chapter 6

Conclusions and future work

In this work, a Grouping Genetic Algorithm was proposed to perform variable decomposition in numerical optimization problems where the number of variables in the decision vector is high. These problems are known as large-scale optimization problems, and some existing methods to solve them involve the creation of subproblems of the original problem into smaller subproblems.

In this Section, we provide the general discussion, conclusions, and results of the proposal, its development, and results, as well as the areas of work opportunity identified to be developed in the future.

6.1 Variable Decomposition

First, regarding the variable decomposition, our proposal demonstrated, in most cases, to obtain a better evaluation of the decomposition, i.e., lower values of $grps_{diff}$ given by Equation 4.5, which indicates that better decompositions are obtained compared to other algorithms.

However, the best assessment of the decomposition was received in the functions with characteristics of complete or partial separability. In those with overlapping or spliced variables, very high evaluations were obtained, which could indicate two facts: first, functions with complex separability properties should not necessarily

obtain low values of Equation 4.5 because this difference is based on partially or completely separable functions.

Hence, the interaction between variables in those functions is also high. Second, the decomposition in difficult-to-separable problems should be evaluated in an alternative way, considering other factors in the evaluation function of the decomposition. Finally, despite the high values of decomposition evaluation, when we optimized the problems, the results were better with the proposed decomposition algorithm.

Although the proposed algorithm proved competent compared to the strategies reported in the literature, convergence studies helped to study and notice that it could be improved by using other types of selection, crossover, mutation, and replacement strategies.

For this, different techniques were proposed, such as tournament and roulette selection, replacement of the worst individual or controlled replacement, and changes in the crossover and mutation operators, exchanging them one by one in the algorithm to measure their effect on the solutions obtained according to the evaluation of the decomposition, i.e., Equation 4.5.

Eight experiments were conducted on the three most complex functions (D=100, 500, and 1000) in the test set to compare the results with those obtained from the original algorithm, specifically the first version of the GGA-VD. The study revealed that certain elements contributed to improved convergence of the algorithm and enhanced diversity in the solutions.

Ultimately, it was decided to work with the version containing tournament selection, replacement of the worst, and improved mutation.

6.2 Large-scale Optimization

Regarding large-scale optimization problems, the literature review of this work supports the importance of developing more proposals focused on problems with constraints because most of the works focused on large-scale have been proposed for

problems without constraints. This work studied constrained optimization problems by adding a penalty factor that allowed the constraint to be part of the objective function. These problems are some degrees more difficult to solve in general, but if we talk about LSO problems, they are even more challenging. Therefore, it is essential to study it and propose more techniques for this kind of optimization problems and also evaluate other handling constraints methods.

In the optimization or collaboration stage of the partial solutions to create the final solution of the problem, two optimization approaches were used, and they were evaluated in three values of the dimension of the problem (D=100, 500, and 1000).

The first one was a memetic algorithm proposed in the literature, which combines the properties of a memetic and the properties of algorithms based on decomposition. The algorithm was implemented with two DVIIC decomposition methods and our GGA-VD proposal. The statistical results showed that the optimization problem obtains smaller fitness values in most cases when decomposed with the Grouping Genetic Algorithm, which is our proposed GGA-VD.

On the other hand, the second optimization algorithm implemented was the CC strategy, which is the basis of the algorithms based on decomposition, just as the two decomposition algorithms mentioned above were used before. In this experiment, the numerical results show, as in the previously mentioned experiment, that the version of the algorithm whose decomposition is performed using GGA-VD obtains better values of fitness in most benchmark functions. Given these results and the conditions under which the experiments were performed, we could confirm in the first instance that the Grouping Genetic Algorithm decomposes the optimization problem better, benefiting the optimization algorithm's performance. To reinforce this theory, some statistical analyses were added, as well as the Wilcoxon test under a 95% of confidence.

An exploratory analysis of the data obtained was used to measure the relationship between the decomposition results and the final solutions provided by the optimization algorithm. First, the frequencies in which each algorithm obtained the best results for each benchmark function were counted, resulting in the algorithms

that used GGA-VD obtaining the highest frequencies in all the dimensions studied.

On the other hand, a study was carried out to measure the correlation between the decomposition results using the GGA-VD of the 18 functions with the results of the optimization algorithms, that is, the results under the memetic and CC. Then, a correlation study was carried out by function; that is, for each problem, the correlation coefficient between the results of the evaluation function of the decomposition and the optimization results was measured. These experiments showed correlations of more than 0.8 in most cases. This indicates that the decomposition obtained by our proposal significantly influences the results obtained by the optimization algorithms.

6.3 Future work

During its development and analysis, several possible improvements to the algorithm were detected. These improvements are proposed as future work and are mentioned below. First, constraints in large-scale problems could be handled through other strategies so that they are not only treated as a penalty.

Then, the Grouping Genetic Algorithm has various factors to study, such as adjustment of the algorithm parameters, study and improvement of the crossover and mutation operators, and the selection of candidate individuals for these processes. An essential element to consider is the evaluation function of the variable decomposition problem. This function is beneficial in problems that are completely or partially separable. Still, in more complex or even non-separable problems, the evaluation function returns very high values, which can contribute to the misinterpretation of the results.

In addition, the optimization process could be improved by using algorithms with a dynamic approach, where the solutions are enhanced as necessary, and even adding machine learning strategies and other soft computing strategies.

It is essential to evaluate proposals with dimensions larger than 1000 and to analyze the algorithm's time complexity. Another test set is essential to determine

whether the algorithm works similarly on other problems.

Characterization of the algorithm and the large-scale problems is essential, too, intending to create the adaptation of the algorithm to the difficulty of the algorithm, in other words, adding statistically, algorithmically, and mathematically based information in the process of decomposing the problem according to its characteristics to facilitate the process of collaboration and optimization of the sub-problems.

Bibliography

[1] Z. Cao, L. Wang, Y. Shi, X. Hei, X. Rong, Q. Jiang, and H. Li, "An effective cooperative coevolution framework integrating global and local search for large-scale optimization problems," in *2015 IEEE Congress on Evolutionary Computation (CEC)*, pp. 1986–1993, IEEE, 2015.

[2] G. Carmona-Arroyo, M. Quiroz-Castellanos, and E. Mezura-Montes, "Variable decomposition for large-scale constrained optimization problems using a grouping genetic algorithm," *Mathematical and Computational Applications*, vol. 27, no. 2, p. 23, 2022.

[3] G. Carmona-Arroyo, J. B. Vázquez-Aguirre, and M. Quiroz-Castellanos, "One-dimensional bin packing problem: An experimental study of instances difficulty and algorithms performance," *Fuzzy Logic Hybrid Extensions of Neural and Optimization Algorithms: Theory and Applications*, pp. 171–201, 2021.

[4] Y. Liu, X. Yao, Q. Zhao, and T. Higuchi, "Scaling up fast evolutionary programming with cooperative coevolution," in *Proceedings of the 2001 Congress on Evolutionary Computation (IEEE Cat. No. 01TH8546)*, vol. 2, pp. 1101–1108, Ieee, 2001.

[5] J. Tang, M. H. Lim, and Y. S. Ong, "Diversity-adaptive parallel memetic algorithm for solving large scale combinatorial optimization problems," *Soft Computing*, vol. 11, pp. 873–888, 2007.

[6] K. Tang, X. Li, P. N. Suganthan, Z. Yang, and T. Weise, "Benchmark functions for the cec'2007 special session and competition on large-scale global optimization," *Nature inspired computation and applications laboratory, USTC, China*, vol. 24, pp. 1–18, 2007.

[7] K. Tang, X. Li, P. N. Suganthan, Z. Yang, and T. Weise, "Benchmark functions for the cec'2010 special session and competition on large-scale global optimization," *Nature inspired computation and applications laboratory, USTC, China*, vol. 24, pp. 1–18, 2007.

[8] A. R. Conn, N. Gould, and P. L. Toint, *Large-scale nonlinear constrained optimization: a current survey*. Springer, 1994.

[9] K. Deb, *Optimization for engineering design: Algorithms and examples*. PHI Learning Pvt. Ltd., 2012.

[10] A. Singh and N. Dulal, "A survey on metaheuristics for solving large scale optimization problems," *Int. J. Comput. Appl*, vol. 170, no. 5, pp. 1–7, 2017.

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

[12] K. A. De Jong, "Evolutionary computation: a unified approach," in *Proceedings of the Companion Publication of the 2015 Annual Conference on Genetic and Evolutionary Computation*, pp. 21–35, 2015.

[13] D. B. Fogel, *Artificial intelligence through simulated evolution*. Wiley-IEEE Press, 1998.

[14] T. Back, *Evolutionary algorithms in theory and practice: evolution strategies, evolutionary programming, genetic algorithms*. Oxford university press, 1996.

[15] D. E. Golberg, "Genetic algorithms in search, optimization, and machine learning," *Addion wesley*, vol. 1989, no. 102, p. 36, 1989.

[16] R. Storn and K. Price, "Differential evolution-a simple and efficient adaptive scheme for global optimization over continuous spaces," *International computer science institute*, 1995.

[17] M. Pant, H. Zaheer, L. Garcia-Hernandez, A. Abraham, *et al.*, "Differential evolution: A review of more than two decades of research," *Engineering Applications of Artificial Intelligence*, vol. 90, p. 103479, 2020.

[18] M. A. Potter and K. A. De Jong, "A cooperative coevolutionary approach to function optimization," in *Parallel Problem Solving from Nature—PPSN III: International Conference on Evolutionary Computation The Third Conference on Parallel Problem Solving from Nature Jerusalem, Israel, October 9–14, 1994 Proceedings 3*, pp. 249–257, Springer, 1994.

[19] X. Ma, X. Li, Q. Zhang, K. Tang, Z. Liang, W. Xie, and Z. Zhu, "A survey on cooperative co-evolutionary algorithms," *IEEE Transactions on Evolutionary Computation*, vol. 23, no. 3, pp. 421–441, 2018.

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

[21] Z. Yang, K. Tang, and X. Yao, "Self-adaptive differential evolution with neighborhood search," in *2008 IEEE congress on evolutionary computation (IEEE World Congress on Computational Intelligence)*, pp. 1110–1116, IEEE, 2008.

[22] Z. Yang, K. Tang, and X. Yao, "Multilevel cooperative coevolution for large scale optimization," in *2008 IEEE congress on evolutionary computation (IEEE World Congress on Computational Intelligence)*, pp. 1663–1670, IEEE, 2008.

[23] M. N. Omidvar, X. Li, Z. Yang, and X. Yao, "Cooperative co-evolution for large scale optimization through more frequent random grouping," in *IEEE Congress on Evolutionary Computation*, pp. 1–8, IEEE, 2010.

[24] J. Liu and K. Tang, "Scaling up covariance matrix adaptation evolution strategy using cooperative coevolution," in *International Conference on Intelligent Data Engineering and Automated Learning*, pp. 350–357, Springer, 2013.

[25] K. Weicker and N. Weicker, "On the improvement of coevolutionary optimizers by learning variable interdependencies," in *Proceedings of the 1999 Congress on Evolutionary Computation-CEC99 (Cat. No. 99TH8406)*, vol. 3, pp. 1627–1632, IEEE, 1999.

[26] M. A. Potter, *The design and analysis of a computational model of cooperative coevolution*. George Mason University, 1997.

[27] W. Chen, T. Weise, Z. Yang, and K. Tang, "Large-scale global optimization using cooperative coevolution with variable interaction learning," in *Parallel Problem Solving from Nature, PPSN XI: 11th International Conference, Kraków, Poland, September 11-15, 2010, Proceedings, Part II 11*, pp. 300–309, Springer, 2010.

[28] M. N. Omidvar, X. Li, Y. Mei, and X. Yao, "Cooperative co-evolution with differential grouping for large scale optimization," *IEEE Transactions on evolutionary computation*, vol. 18, no. 3, pp. 378–393, 2013.

[29] Y. Sun, M. Kirley, and S. K. Halgamuge, "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*, pp. 313–320, 2015.

[30] Y. Mei, M. N. Omidvar, X. Li, and X. Yao, "A competitive divide-and-conquer algorithm for unconstrained large-scale black-box optimization," *ACM Transactions on Mathematical Software (TOMS)*, vol. 42, no. 2, pp. 1–24, 2016.

[31] J. Sun and H. Dong, "Cooperative co-evolution with correlation identification grouping for large scale function optimization," in *2013 IEEE Third*

International Conference on Information Science and Technology (ICIST), pp. 889–893, IEEE, 2013.

[32] F. Wei, Y. Wang, and T. Zong, "A novel cooperative coevolution for large scale global optimization," in *2014 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*, pp. 738–741, IEEE, 2014.

[33] F. Wei, Y. Wang, and T. Zong, "Variable grouping based differential evolution using an auxiliary function for large scale global optimization," in *2014 IEEE Congress on Evolutionary Computation (CEC)*, pp. 1293–1298, IEEE, 2014.

[34] R. Lan, Y. Zhu, H. Lu, Z. Liu, and X. Luo, "A two-phase learning-based swarm optimizer for large-scale optimization," *IEEE transactions on cybernetics*, vol. 51, no. 12, pp. 6284–6293, 2020.

[35] P. Xu, W. Luo, X. Lin, Y. Chang, and K. Tang, "Difficulty and contribution-based cooperative coevolution for large-scale optimization," *IEEE Transactions on Evolutionary Computation*, vol. 27, no. 5, pp. 1355–1369, 2022.

[36] A. Vakhnin and E. Sopov, "A novel self-adaptive cooperative coevolution algorithm for solving continuous large-scale global optimization problems," *Algorithms*, vol. 15, no. 12, p. 451, 2022.

[37] A. Kelkawi, M. El-Abd, and I. Ahmad, "Gpu-based cooperative coevolution for large-scale global optimization," *Neural Computing and Applications*, vol. 35, no. 6, pp. 4621–4642, 2023.

[38] A. Kelkawi, I. Ahmad, and M. El-Abd, "Spark-based cooperative coevolution for large scale global optimization," *Cluster Computing*, vol. 27, no. 2, pp. 1911–1926, 2024.

[39] Q. Duan, C. Shao, G. Zhou, M. Zhang, Q. Zhao, and Y. Shi, "Distributed evolution strategies with multi-level learning for large-scale black-box optimization," *IEEE Transactions on Parallel and Distributed Systems*, 2024.

[40] P. Xu, W. Luo, X. Lin, J. Zhang, Y. Qiao, and X. Wang, "Constraint-objective cooperative coevolution for large-scale constrained optimization," *ACM Transactions on Evolutionary Learning and Optimization*, vol. 1, no. 3, pp. 1–26, 2021.

[41] K. Zhang and B. Li, "Cooperative coevolution with global search for large scale global optimization," in *2012 IEEE Congress on Evolutionary Computation*, pp. 1–7, IEEE, 2012.

[42] F. J. Solis and R. J.-B. Wets, "Minimization by random search techniques," *Mathematics of operations research*, vol. 6, no. 1, pp. 19–30, 1981.

[43] H.-q. Li and L. Li, "A novel hybrid particle swarm optimization algorithm combined with harmony search for high dimensional optimization problems," in *The 2007 International Conference on Intelligent Pervasive Computing (IPC 2007)*, pp. 94–97, IEEE, 2007.

[44] S.-Z. Zhao, J. J. Liang, P. N. Suganthan, and M. F. Tasgetiren, "Dynamic multi-swarm particle swarm optimizer with local search for large scale global optimization," in *2008 IEEE congress on evolutionary computation (IEEE world congress on computational intelligence)*, pp. 3845–3852, IEEE, 2008.

[45] S. Muelas, A. La Torre, and J.-M. Peña, "A memetic differential evolution algorithm for continuous optimization," in *2009 Ninth International Conference on Intelligent Systems Design and Applications*, pp. 1080–1084, IEEE, 2009.

[46] L.-Y. Tseng and C. Chen, "Multiple trajectory search for large scale global optimization," in *2008 IEEE congress on evolutionary computation (IEEE world congress on computational intelligence)*, pp. 3052–3059, IEEE, 2008.

[47] D. Molina, M. Lozano, A. M. Sánchez, and F. Herrera, "Memetic algorithms based on local search chains for large scale continuous optimisation problems: Ma-ssw-chains," *Soft Computing*, vol. 15, pp. 2201–2220, 2011.

[48] A. LaTorre, S. Muelas, and J.-M. Peña, "Multiple offspring sampling in large scale global optimization," in *2012 IEEE Congress on Evolutionary Computation*, pp. 1–8, IEEE, 2012.

[49] F. Zhang, T. Zhang, R. Wang, and H. Lei, "Memetic algorithm based on self-adaptive differential evolution and improved simplex crossover for large scale global optimization," in *2016 Tsinghua University-IET Electrical Engineering Academic Forum*, pp. 1–8, IET, 2016.

[50] A. E. Aguilar-Justo and E. Mezura-Montes, "A local cooperative approach to solve large-scale constrained optimization problems," *Swarm and Evolutionary Computation*, vol. 51, p. 100577, 2019.

[51] E. Mezura-Montes and C. A. C. Coello, "Constraint-handling in nature-inspired numerical optimization: past, present and future," *Swarm and Evolutionary Computation*, vol. 1, no. 4, pp. 173–194, 2011.

[52] K. Deb, "An efficient constraint handling method for genetic algorithms," *Computer methods in applied mechanics and engineering*, vol. 186, no. 2-4, pp. 311–338, 2000.

[53] A. E. Smith, D. W. Coit, T. Baeck, D. Fogel, and Z. Michalewicz, "Penalty functions," *Handbook of evolutionary computation*, vol. 97, no. 1, p. C5, 1997.

[54] E. Sayed, D. Essam, R. Sarker, and S. Elsayed, "Decomposition-based evolutionary algorithm for large scale constrained problems," *Information Sciences*, vol. 316, pp. 457–486, 2015.

[55] X. Li, K. Tang, M. N. Omidvar, Z. Yang, K. Qin, and H. China, "Benchmark functions for the cec 2013 special session and competition on large-scale global optimization," *gene*, vol. 7, no. 33, p. 8, 2013.

[56] E. Sayed, D. Essam, and R. Sarker, "Dependency identification technique for

large scale optimization problems," in *2012 IEEE Congress on Evolutionary Computation*, pp. 1–8, IEEE, 2012.

[57] F. Van den Bergh and A. P. Engelbrecht, "A cooperative approach to particle swarm optimization," *IEEE transactions on evolutionary computation*, vol. 8, no. 3, pp. 225–239, 2004.

[58] Z. Yang, K. Tang, and X. Yao, "Differential evolution for high-dimensional function optimization," in *2007 IEEE Congress on evolutionary computation*, pp. 3523–3530, IEEE, 2007.

[59] M. N. Omidvar, X. Li, and X. Yao, "Cooperative co-evolution with delta grouping for large scale non-separable function optimization," in *IEEE congress on evolutionary computation*, pp. 1–8, IEEE, 2010.

[60] B. Xu, Y. Zhang, D. Gong, Y. Guo, and M. Rong, "Environment sensitivity-based cooperative co-evolutionary algorithms for dynamic multi-objective optimization," *IEEE/ACM transactions on computational biology and bioinformatics*, vol. 15, no. 6, pp. 1877–1890, 2017.

[61] M. N. Omidvar, M. Yang, Y. Mei, X. Li, and X. Yao, "Dg2: A faster and more accurate differential grouping for large-scale black-box optimization," *IEEE Transactions on evolutionary computation*, vol. 21, no. 6, pp. 929–942, 2017.

[62] Y. Sun, M. Kirley, and S. K. Halgamuge, "A recursive decomposition method for large scale continuous optimization," *IEEE Transactions on evolutionary computation*, vol. 22, no. 5, pp. 647–661, 2017.

[63] Y. Sun, M. N. Omidvar, M. Kirley, and X. Li, "Adaptive threshold parameter estimation with recursive differential grouping for problem decomposition," in *Proceedings of the genetic and evolutionary computation conference*, pp. 889–896, 2018.

[64] Y. Sun, X. Li, A. Ernst, and M. N. Omidvar, "Decomposition for large-scale optimization problems with overlapping components," in *2019 IEEE congress on evolutionary computation (CEC)*, pp. 326–333, IEEE, 2019.

[65] M. Yang, A. Zhou, C. Li, and X. Yao, "An efficient recursive differential grouping for large-scale continuous problems," *IEEE Transactions on Evolutionary Computation*, vol. 25, no. 1, pp. 159–171, 2020.

[66] E. Sopov, "Large-scale global optimization using a binary genetic algorithm with eda-based decomposition," in *Advances in Swarm Intelligence: 7th International Conference, ICSI 2016, Bali, Indonesia, June 25-30, 2016, Proceedings, Part I 7*, pp. 619–626, Springer, 2016.

[67] H. Mühlenbein and G. Paass, "From recombination of genes to the estimation of distributions i. binary parameters," in *International conference on parallel problem solving from nature*, pp. 178–187, Springer, 1996.

[68] S. I. Valdez, A. Hernández, and S. Botello, "A boltzmann based estimation of distribution algorithm," *Information Sciences*, vol. 236, pp. 126–137, 2013.

[69] J. D. Schaffer and A. Morishima, "An adaptive crossover distribution mechanism for genetic algorithms," in *Proceedings of the Second International Conference on Genetic Algorithms on Genetic algorithms and their application*, pp. 36–40, 1987.

[70] C.-K. Goh and K. C. Tan, "A competitive-cooperative coevolutionary paradigm for dynamic multiobjective optimization," *IEEE Transactions on Evolutionary Computation*, vol. 13, no. 1, pp. 103–127, 2008.

[71] C. K. Goh, K. C. Tan, D. Liu, and S. C. Chiam, "A competitive and cooperative co-evolutionary approach to multi-objective particle swarm optimization algorithm design," *European Journal of Operational Research*, vol. 202, no. 1, pp. 42–54, 2010.

[72] S. Strasser, J. Sheppard, N. Fortier, and R. Goodman, "Factored evolutionary algorithms," *IEEE Transactions on Evolutionary Computation*, vol. 21, no. 2, pp. 281–293, 2016.

[73] L. Li, W. Fang, Y. Mei, and Q. Wang, "Cooperative coevolution for large-scale global optimization based on fuzzy decomposition," *Soft Computing*, vol. 25, no. 5, pp. 3593–3608, 2021.

[74] X. Guan, X. Zhang, J. Wei, I. Hwang, Y. Zhu, and K. Cai, "A strategic conflict avoidance approach based on cooperative coevolutionary with the dynamic grouping strategy," *International Journal of Systems Science*, vol. 47, no. 9, pp. 1995–2008, 2016.

[75] A. E. Aguilar-Justo and E. Mezura-Montes, "Towards an improvement of variable interaction identification for large-scale constrained problems," in *2016 IEEE Congress on Evolutionary Computation (CEC)*, pp. 4167–4174, IEEE, 2016.

[76] A. E. Aguilar-Justo, E. Mezura-Montes, S. M. Elsayed, and R. A. Sarker, "Decomposition of large-scale constrained problems using a genetic-based search," in *2016 IEEE International Autumn Meeting on Power, Electronics and Computing (ROPEC)*, pp. 1–6, IEEE, 2016.

[77] A. Vakhnin and E. Sopov, "Investigation of the icc framework performance for solving constrained lsgo problems," *Algorithms*, vol. 13, no. 5, p. 108, 2020.

[78] O. Ramos-Figueroa, M. Quiroz-Castellanos, E. Mezura-Montes, and O. Schütze, "Metaheuristics to solve grouping problems: A review and a case study," *Swarm and Evolutionary Computation*, vol. 53, p. 100643, 2020.

[79] A. H. Kashan, A. A. Akbari, and B. Ostadi, "Grouping evolution strategies: An effective approach for grouping problems," *Applied Mathematical Modelling*, vol. 39, no. 9, pp. 2703–2720, 2015.

[80] R. V. Book, "Michael r. garey and david s. johnson, computers and intractability: a guide to the theory of np-completeness," 1980.

[81] A. Martinez-Sykora, R. Alvarez-Valdés, J. A. Bennell, R. Ruiz, and J. M. Tamarit, "Matheuristics for the irregular bin packing problem with free rotations," *European Journal of Operational Research*, vol. 258, no. 2, pp. 440–455, 2017.

[82] Z. Pei, Z. Wang, and Y. Yang, "Research of order batching variable neighborhood search algorithm based on saving mileage," in *3rd International Conference on Mechatronics Engineering and Information Technology (ICMEIT 2019)*, pp. 196–203, Atlantis Press, 2019.

[83] T. Kämpke, "Simulated annealing: Use of a new tool in bin packing," *Annals of Operations Research*, vol. 16, pp. 327–332, 1988.

[84] D. Schermer, M. Moeini, and O. Wendt, "A hybrid vns/tabu search algorithm for solving the vehicle routing problem with drones and en route operations," *Computers & Operations Research*, vol. 109, pp. 134–158, 2019.

[85] G. Evtimov and S. Fidanova, "Ant colony optimization algorithm for 1d cutting stock problem," in *Advanced Computing in Industrial Mathematics: 11th Annual Meeting of the Bulgarian Section of SIAM December 20-22, 2016, Sofia, Bulgaria. Revised Selected Papers*, pp. 25–31, Springer, 2018.

[86] I. A. L. Sanchez, J. M. Vargas, C. A. Santos, M. G. Mendoza, and C. J. M. Moctezuma, "Solving binary cutting stock with matheuristics using particle swarm optimization and simulated annealing," *Soft Computing*, vol. 22, pp. 6111–6119, 2018.

[87] M. H. Xue, T. Z. Wang, and S. Mao, "Double evolutsional artificial bee colony algorithm for multiple traveling salesman problem," in *MATEC Web of Conferences*, vol. 44, p. 02025, EDP Sciences, 2016.

[88] E. Falkenauer, "A new representation and operators for genetic algorithms applied to grouping problems," *Evolutionary computation*, vol. 2, no. 2, pp. 123–144, 1994.

[89] O. Ramos-Figueroa, M. Quiroz-Castellanos, E. Mezura-Montes, and R. Kharel, "Variation operators for grouping genetic algorithms: A review," *Swarm and Evolutionary computation*, vol. 60, p. 100796, 2021.

[90] A. E. Eiben and C. A. Schippers, "On evolutionary exploration and exploitation," *Fundamenta Informaticae*, vol. 35, no. 1-4, pp. 35–50, 1998.

[91] D. Mosk-Aoyama and D. Shah, "Fast distributed algorithms for computing separable functions," *IEEE Transactions on Information Theory*, vol. 54, no. 7, pp. 2997–3007, 2008.

[92] T. Ray and X. Yao, "A cooperative coevolutionary algorithm with correlation based adaptive variable partitioning," in *2009 IEEE congress on evolutionary computation*, pp. 983–989, IEEE, 2009.

[93] T. Blickle and L. Thiele, "A comparison of selection schemes used in evolutionary algorithms," *Evolutionary Computation*, vol. 4, no. 4, pp. 361–394, 1996.

[94] B.-T. Zhang and J.-J. Kim, "Comparison of selection methods for evolutionary optimization," *Evolutionary optimization*, vol. 2, no. 1, pp. 55–70, 2000.

[95] J. Smith, "On replacement strategies in steady state evolutionary algorithms," *Evolutionary Computation*, vol. 15, no. 1, pp. 29–59, 2007.

[96] J. H. Holland, *Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control, and artificial intelligence*. MIT press, 1992.

[97] K. A. De Jong, *An analysis of the behavior of a class of genetic adaptive systems*. University of Michigan, 1975.

[98] A. Wetzel, "Evaluation of the effectiveness of genetic algorithms in combinatorial optimization," *University of Pittsburgh*, vol. 1, no. 983, pp. 1932–4537, 1983.

[99] M. Quiroz-Castellanos, L. Cruz-Reyes, J. Torres-Jimenez, C. Gómez, H. J. F. Huacuja, and A. C. Alvim, "A grouping genetic algorithm with controlled gene transmission for the bin packing problem," *Computers & Operations Research*, vol. 55, pp. 52–64, 2015.