CENTER FOR RESEARCH IN MATHEMATICS

MASTER THESIS

Metaheuristics in the Optimization of Cryptographic Boolean Functions

---

Author:
Isaac López López

Supervisor:
Dr. Carlos Segura González
Dr. Guillermo Sosa Gómez

A thesis submitted in fulfillment of the requirements for the degree of Master of Science in Computing and Industrial Mathematics

in the

Computer Science Department

April 3, 2019

Declaration of Authorship

I, Isaac López López , declare that this thesis titled, “Metaheuristics in the Optimization of Cryptographic Boolean Functions ” and the work presented in it are my own. I confirm that:

- This work was done wholly or mainly while in candidature for a research degree at this Center for Research.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- I have acknowledged all main sources of help.
- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Signed:

Date:

"We need heuristic reasoning when we construct a strict proof as we need scaffolding when we erect a building. "

George Pólya

CENTER FOR RESEARCH IN MATHEMATICS

Abstract

Computer Science Department

Master of Science in Computing and Industrial Mathematics

Metaheuristics in the Optimization of Cryptographic Boolean Functions

by Isaac López López

The aim of the research presented in this thesis is to proof that diversity-based metaheuristics are able to generate Boolean Functions (BFs) with high nonlinearity. The best results known for 10-variable BFs correspond to algebraic constructions and the metaheuristics have never been successful generating such BFs. Metaheuristics have obtained BFs with suboptimal nonlinearity and this thesis shows that by incorporating a proper control of diversity and adequately designed cost function, results are improved further. We have proposed a new cost function to guide the search based on considering more information from the Walsh Hadamard Transform (WHT). Instead of using only one value of the WHT, we employ the two maximum absolute values that appear and their number of appearances. It is also proposed a diversity-based metaheuristic that allows us to explore a large amount of regions in the search space.

The thesis focuses on the design of a novel evolutionary memetic algorithm that incorporates a clustering technique coupled with a diversity-based replacement strategy. Typical diversity management techniques force to all the BFs to contribute to the diversity, however, in the technique designed for the thesis is allowed that some BFs form clusters to promote both exploration and exploitation at each phase of the algorithm. We call this algorithm as: Memetic Algorithm with Clusters coupled with a Replacement with Elite based Dynamic Diversity Control with Clustering (MAC-REDDCC). The MAC-REDDCC algorithm is the first of its kind according to the best of our knowledge.

The MAC-REDDCC method proposed is able to overcome all the existing results found by traditional metaheuristics and is able to reach the results obtained with algebraic constructions by incorporating a simple algebraic technique such as initialization. This implies that a diversity-based method is able to generate BFs with high nonlinearity. The MAC-REDDCC method is even better than all traditional methods to generate BFs with high nonlinearity and has a successful performance when comparing with algebraic constructions.

Acknowledgements

This thesis is possible thanks to the support and encouragement of my principal supervisor, Dr. Carlos Segura González, so, I am very grateful to him, I am also grateful with my another supervisor, Dr. Guillermo Sosa Gómez, due this problem was proposed by him.

Special thanks must go to my family, especially my mother, for her loving support, not only during the time spent on this thesis, but also throughout the many years of education which preceded it.

Thanks must also go to CIMAT and CONACyT for all the economical support and the knowledge received during the time spent on this master degree.

Contents

List of Figures

List of Tables

List of Algorithms

List of Abbreviations

AI	Algebraic Immunity
AC	Auto Correlation
ANF	Algebraic Normal Form
BF	Boolean Function
CBF	Cryptographic Boolean Function
CF	Cost Function
CGP	Cartesian Genetic Programming
CI	Correlation Immunity
CLONALG	Clonal Selection Algorithm
DCN	Distance to the Closest Neighbor
DCNC	Distance to the Closest Neighbor in a Cluster
EA	Evolutionary Algorithm
EDA	Estimation of Distribution Algorithm
ES	Evolution Strategy
FAR	Fast Algebraic Resistance
FHC	Full Hill Climbing
FIQTS	First Improvement Quasi-Tabu Search
FWHT	Fast Walsh Hadamard Transform
GA	Genetic Algorithm
GP	Genetic Programming
HC	Hill Climbing
ILS	Iterated Local Search
LMA	Lamarckian Memetic Algorithm
MA	Memetic Algorithm
MAC	Memetic Algorithm with Clusters
NTB	Nonlinearity Theoretical Bound
PSO	Particle Swarm Optimization
REDDC	Replacement with Elite based Dynamic Diversity Control
REDDCC	Replacement with Elite based Dynamic Diversity Control with Clustering
RGE	Replacement Generational with Elitism
RMDDC	Replacement with Multiobjective based Dynamic Diversity Control
SA	Simulated Annealing
SHC	Smart Hill Climbing
WHT	Walsh Hadamard Transform

To life itself for everything lived...

Chapter 1

Introduction

1.1 Introduction

Cryptography is a fundamental issue for the attainment of secure communications in the presence of third parties that might have access to the communication channel. For instance, in the case of Internet, cryptography is specially important because public channels that might be accessed by many parties are used and important information is transferred. In this sense, the basic principle of cryptography is that instead of sending the original message — which is called the *plaintext* — through the channel, a different message that might be used to recover the original one is sent. The message that is sent over the channel is called the *ciphertext*. In order to convert the ciphertext into the plaintext, a secret code which is called the *key* is used. The aim of cryptography in this sense is to avoid that any party that is not involved in the communication might recover the plaintext even if the ciphertext is accessed. The transformation of plaintext to ciphertext is called encryption and the opposite step is called decryption. Encryption-decryption is considered the most ancient cryptographic activity. As an example we can consider the *Caesar cipher*, which is a simple system used and developed by Julius Caesar to send secure information to his troops. It was all about substituting certain letters for others [27]. This algorithm is what we would call a cipher. Nowadays, there are more sophisticated cipher algorithms that allow a higher level of security. In recent years, since the access to computation power and the amount of important information transferred through public channels have dramatically increased, the use of secure channels with advanced encryption-decryption schemes has been specially important. As a result, a lot of research on this topic have been performed in the last decades and the nature of encryption-decryption schemes has deeply changed.

The concept of power has changed along the years. Once it was said that who has the land has the power. Then, it changed to who has the information has the power. Currently, it is said that who knows how to handle information has the power. Many companies and people save important information in digital devices, so in order to avoid information leakage, cryptography is used. Obviously, with the proper decryption key

such an information might be accessed. Cryptanalysis is a power tool to analyze cryptographic systems. One of the main aims of cryptanalysis is to find weaknesses of cryptographic systems so that the plaintext can be recovered even if the decryption key is unknown. Thus, it is used to violate authentication schemes, to break cryptographic protocols, and, more benignly, to find and correct weaknesses of encryption algorithms. Finally, another objective related to cryptography is to enable digital communications over noisy channel in such a way that the errors in the information transmission can be detected and corrected by the receiver [7].

In many cryptographic systems and in the use of noisy channels, Boolean Functions (BFs) play an important role. Specifically, BFs are used in the internal operation of many cryptographic algorithms and it is known that in order to avoid the success of cryptanalysis on such systems, increasing the degree of nonlinearity of BFs is really important. The research in Cryptographic Boolean Functions (CBFs) has increased significantly in the last few decades. Particularly, the cryptographic community has been working widely in the problem of generating BFs with good cryptographic properties. One of the first authors on studying this topic showed the important implication of the feature called correlation immunity using algebraic procedures [54]. CBFs are currently used in symmetric-key cryptography (used to encrypt and decrypt data with the same key) both in block and stream ciphers [20]. In block ciphers, algorithms encrypt and decrypt data in blocks of certain size whereas in stream ciphers this is done bitwise. The common feature is that in both of these types of ciphers, the only nonlinear element are usually the BFs applied for the stream ciphers and the vectorial BFs (better known as substitution boxes or s-boxes) for block ciphers. Without the use of BFs in stream ciphers or s-boxes in block ciphers, it would be trivial to break the cryptographic systems. As a result, designing proper BFs is a crucial step.

Since many years ago, efforts were concentrated in the problem of finding proper CBFs to avoid cryptanalytic attacks [31]. Thus, cryptographers studied which were the desired properties for CBFs. Particularly, some desired features are the high nonlinearity, balancedness, algebraic degree and low autocorrelation. Among the previous ones, attaining a high nonlinearity for a balanced BF is one of the most challenging tasks [25]. BFs with these properties hinder the application of linear and differential cryptanalysis. In fact, when BFs with proper properties are used, these kinds of cryptanalysis do not succeed faster than exhaustive key search. Currently, many ways to generate CBFs with proper features have been designed. The three main approaches that are currently used are the following [40]: algebraic constructions, random generation and metaheuristic constructions. Algebraic constructions use mathematical procedures based on algebraic properties to create BFs with good cryptographic properties. Random generation is easy and fast, but the resulting BFs usually have suboptimal properties for cryptographic

applications, so many BFs must be checked and even if a large computational effort is used, the probability to attain proper BFs is low. Finally, metaheuristic constructions are heuristic techniques that have attained quite promising BFs and are easier to design than algebraic methods. However, up to now, metaheuristic techniques have not been able to attain so good results as the algebraic techniques. For instance, attending to the nonlinearity for balanced CBFs, metaheuristics attain CBFs with lower nonlinearity than the best-known CBFs that have been generated with algebraic constructions. It is also important to note that several general metaheuristics have been used. Among them, Genetic Algorithms, Hill Climbing, Simulated Annealing and Cartesian Genetic Programming are quite popular.

Regarding metaheuristics, its application for solving optimization problems dates back to the 1960 with the design of the firsts Evolutionary Algorithms (EAs). The history of metaheuristics is briefly described in [49]. The word "metaheuristic" is used to denote a high-level framework (a set of concepts and strategies that blend together offering a perspective on the development of optimization algorithms) and for referring to a specific implementation of an algorithm based on such a framework designed to solve a specific optimization problem. At least since two decades ago, the firsts researchers started to study the design of CBFs from the metaheuristics point of view. The first Genetic Algorithm that maximizes the nonlinearity of BFs was proposed in [30]. A smart version of a hill climbing method to find proper BFs [33] was also proposed. Subsequently, nonlinearity was optimized for balanced BFs with 8 variables by proposing a novel crossover operator and realizing that small population are good enough to find BFs with high nonlinearity [32]. More recently, it was noted that the applied fitness function significantly affects the performance of the optimization methods [11]. This last study was developed by applying the trajectory-based method called Simulated Annealing. This work was extended by designing a more sophisticated search technique of Simulated Annealing called "vanilla" [13] in order to obtain CBFs with some additional properties such as resilience. Authors note that the huge search space hinders the attainment of better results. Obviously, this is not a drawback specific to this technique. In order to partially avoid this drawback some authors consider some of the advances performed with algebraic constructions. Particularly, [5, 19] proposed trajectory-based search methods that operate with a kind of functions that are called the bent BFs. They randomly adjust the bent BF to convert it on a balanced BF. In this way they decrease the nonlinearity of a bent BF instead of increasing the nonlinearity of a randomly created BF. First, a bent BF is constructed with the method given in [28]. Then, a trajectory-based heuristic is used to attain a balanced BF. These methods show really good results. In fact, at the starting point of this thesis, these methods were the state-of-the-art in the generation of BFs with high nonlinearity by applying metaheuristics. A different alternative to deal with the so large search space in the case of maximizing the nonlinearity for balanced BFs is to take into account the symmetries that

appear in the fitness landscape when using the bit string representation [41]. Some other recent works [39] include comparisons among different metaheuristics such as Genetic Algorithms (GAs), Evolution Strategies (ESs) and Cartesian Genetic Programming (CGP). Among the tested approaches CGP was the best one to generate 8-variable CBFs with high nonlinearity. In fact, it is the EA with best performance among those that do not apply the concept of bent BFs. Another contribution of this paper is to compare three different fitness functions with the aim of increasing the nonlinearity. Finally, some notions on the difficulty of generating balanced functions with high nonlinearity are discussed in [37]. This last study is developed using Estimation of Distribution Algorithms (EDAs). They note some undesired properties of the basic fitness function that is usually applied and prove that the problem is not order-1 deceptive.

Taking into account the difference among the best-known BFs and the ones attained by state-of-the-art metaheuristics, the main aim of this thesis has been to advance further on the development of metaheuristic construction techniques. Particularly, the thesis focuses on the development of novel memetic algorithms incorporating explicit control of diversity.

1.2 Motivation

The main motivation of this dissertation is to improve further the quality of metaheuristic techniques for the construction of CBFs with high nonlinearity. Particularly, the desire was to design techniques that were able to generate BFs with a quality similar to the ones that have been found with algebraic methods. Another motivation of this paper has been to test the generality of a set of metaheuristics design principles that have provided important advances recently [45]. The principle is based on modifying the internal operation of metaheuristics by taking into account the stopping criterion and diversity with the aim of properly balancing the exploration and exploitation capabilities. Diversity-based algorithms have shown its superiority only in executions with long time, but taking into account the increase of technological capabilities in the last decades it has become an interesting topic. Since the search space considered in the problem addressed in this paper is huge, it is important to avoid fast convergence to a reduced region of the search space, so using such a principle might be beneficial. Note that this idea might be incorporated in several ways. In [46] a proposal that follows this principle was designed. Particularly, the *Replacement with Multiobjective based Dynamic Diversity Control* (RMDDC) strategy was devised. RMDDC is a novel diversity-based replacement strategy to avoid premature convergence in EAs. RMDDC is simple and it allowed the attainment of high-quality results. In fact, RMDDC has obtained the best-known results for several instances of the Frequency Assignment Problem (FAP) and the Two-dimensional Packing Problem (2DPP). Additionally, it was able to solve the most difficult Sudoku puzzles

known to date with a 100% of success rate. In order to study this principle, RMDDC method was modified in this thesis with some specific methods designed for the generation of CBFs with high nonlinearity.

Note that methods developed for the generation of CBFs with high nonlinearity have not considered the diversity in an explicit way. Overall, this thesis pretends to show the viability and utility of considering a diversity-based scheme motivated by the RMDDC method for the generation of CBFs with high nonlinearity. This can be viewed as an extension of the work presented in [45] and will serve as a validation with more complex combinatorial problems.

1.3 Problem Description and Hypothesis

The generation of CBFs with high nonlinearity is not an easy problem. One of the main difficulties to handle is that the search space $\mathcal{F}_n$ is immensely large: $|\mathcal{F}_n| = 2^{2^n}$. In fact, for problems with more than five variables it is not possible to do an exhaustive search nowadays. Furthermore, for larger sizes of BFs, not only the search space grows, but also the computational cost of calculating the various important properties increases. Specifically the calculus of the nonlinearity is based on the usage of the Walsh Hadamard Transform, so there are more BFs and estimating the quality of each BF is more costly. Table 1.1 shows the number of n-variable BFs for n ranging between 4 and 12.

TABLE 1.1: Number of n-variable BF

n	4	5	6	7	8	9	10	11	12
$\lvert\mathcal{F}_n\rvert$	2^{16}	2^{32}	2^{64}	2^{128}	2^{256}	2^{512}	2^{1024}	2^{2048}	2^{4096}
$\approx$	10^4	10^9	10^{19}	10^{38}	10^{77}	10^{154}	10^{308}	10^{616}	10^{1233}

Suppose that visiting an n-variable BF takes one picosecond (10^{-12} seconds). Then, exhaustive search would need four months to visit all functions for $n = 6$, and about 10^8 times the age of the universe (13.7×10^9 years) to visit all those for $n = 7$. The number of 8-variable BFs is $0.0012\times$ the number of atoms in the visible universe (10^{80}) [7]. Trying to find good CBFs at random is not plausible for these values of n. This dissertation focuses on the n-variable BF problem, with $n = 8, 10$. The main efforts are concentrated in improving further the highest nonlinearity achievable by metaheuristics for these n-variable BF problems.

Due to the large efforts that have been dedicated to this problem, some researchers consider that balanced BFs for 8 variables with nonlinearity equal to 118 do not exist, so the best-known BFs might already be known. This function has been searched for at least since 30 years ago but its existence has not been proved. A similar situation appears for the case of 10-variable BFs with nonlinearity equal to 494. In any case, the main

purpose is to reduce the gap between algebraic methods and metaheuristics approaches.

In this thesis the following hypothesis is studied:

> **Diversity-based metaheuristic methods can provide a better way of generating CBFs with high nonlinearity, improving further the best current results obtained in literature by other metaheuristic methods and reducing the gap between algebraic and metaheuristic approaches.**

1.4 Objectives

The **general objective** of this thesis is to improve the performance of metaheuristics for the generation of CBFs with high nonlinearity. In order to achieve this objective, the state-of-the-art is analyzed and extended by incorporating diversity-based strategies. The **specific objectives** are the definition of proper cost functions, the design of novel diversity-based memetic algorithms and the achievement of CBFs with a quality similar to the ones attained by algebraic methods. A particular objective in the design of the novel diversity-based memetic algorithm is to generate a method able to combine intensification and exploration in a proper way by enabling some degree of intensification in the whole optimization process. This last one is the main novelty when compared to current diversity-based methods and the purpose of this modification is to reduce the time required to attain high-quality BFs.

1.5 Contributions

This thesis proposes the first algorithm that considers the diversity in an explicit way in order to generate balanced CBFs with high nonlinearity. The main novelty appears in the replacement phase, where concepts related to clustering are applied. Additionally, two new cost functions that guide the search towards proper regions are devised. Regarding the diversity management, two algorithms that involve two different ways of controlling the diversity are proposed. The first one enforces a large contribution to diversity for every member of the population, whereas the other one considers the definition of clusters, meaning that some close members are maintained in the population. Finally, a novel population initialization method is also proposed which considers some algebraic constructions with the aim of speeding up the attainment of higher quality BFs.

Concerning to the results we remark that this metaheuristic methods have allowed to reduce the gap between algebraic and heuristic constructions. Algebraic strategies are the methods that have allowed that attainment of

the best known BFs up to date, and for many years many researchers of the field of heuristics have tried to match the results of algebraic constructions. However, this has not been possible. With the methods proposed in this thesis, the results obtained by algebraic constructions for 8 and 10 variables have been matched.

1.6 Brief Overview of the Thesis Chapters

This dissertation is composed of five chapters where it is clearly described the state-of-the-art, the work developed for this thesis, the results obtained, the conclusions and some lines of future work. The thesis is structured as follows.

Chapter 2 starts with a brief explanation about the main metaheuristics that have been used in the generation of BFs with good cryptographic properties. The main concepts involved in the metaheuristic field are introduced. The theory concerning to BFs and some motivations are exposed briefly. Then the related work that has been done with metaheuristics is summarized.

Chapter 3 starts with the importance of the election of an appropriate cost function and the particular motivation for the design of a novel cost function for this problem is discussed. Then the cost functions and algorithms proposed are fully described.

In chapter 4 the main results obtained with the algorithms described in chapter 3 are exposed. Results for different parameters are discussed and compared with the best-known results attained in literature. These studies include statistical validation, tables and graphs with the aim of better describing the attained results.

Finally, chapter 5 exposes the conclusions based on the results obtained and some lines of future work.

Chapter 2

Theoretical Framework

This chapter describes the set of metaheuristics that has been applied in the search of BFs with high nonlinearity, the main concepts involved in the metaheuristic field, the theory concerning to BFs and the related work that has been done with metaheuristics to date for balanced BFs with high nonlinearity.

2.1 Metaheuristics

Metaheuristic is a term used to refer to a subfield of optimization with stochastic methods and encompasses a set of general approximate algorithms used mainly to solve optimization problems. These algorithms are usually applied to solve problems where there is no much analytic information about the function to optimize and where other techniques such as mathematical programming and/or exact techniques cannot be applied. Although the techniques are general, in most cases problem-dependent decisions are required to solve satisfactorily. These kinds of approaches have been specially successful in the most complex optimization problems, usually with huge search spaces.

Metaheuristics are one of the optimization technologies most widely used nowadays. Their success is partially due to the simplicity both in design and implementation and because in many complex problems they are the methods that have attained the best-known results currently. A common drawback of metaheuristics is that they can not guarantee to solve problems exactly. In fact, usually there is no guarantee at all in relation to the quality of the attained results. However, as it has been mentioned, in practice it is used when other methods fail so practitioners are usually satisfied with non-optimal but high-quality solutions, which are in most cases obtained by metaheuristics in a reasonable time.

Regarding the design of metaheuristics, two somewhat contradictory criteria must be taken into account:

- **Diversification:** some of the candidate solutions evaluated should be distant to any of the previously evaluated candidate solutions with the aim of discovering new promising regions.
- **Intensification:** some of the candidate solutions evaluated should be close to some of the best solutions previously evaluated with the aim of refining such solutions.

It is known that inducing a proper balance between diversification and intensification is fundamental in order to generate high-quality solutions and to widely explore the search space. In many cases, practitioners do not design metaheuristics taking into account the balance between intensification and diversification in an explicit way. Particularly, the stopping criterion is not usually taken into account to perform such a balance. However, some recent techniques have shown the importance of relating diversification, intensification and the stopping criterion. This thesis proposes a novel explicit mechanism to induce a proper balance between intensification and diversification that relates its control with the stopping criterion.

Generally, three of the main purposes that are taken into account when designing metaheuristics are:

1. Solving problems faster
2. Solving larger problems
3. Obtaining robust algorithms, i.e. methods that in each execution attain high-quality results

This thesis is specially focused on solving large problems and in obtaining robust algorithms. While an effort was performed to implement the algorithms in an efficient way, no complex data structures were used, so probably faster implementation might be plausible for the algorithms proposed in this thesis.

Several components must be selected when applying metaheuristics. Many of these components depend on the particular metaheuristic that is used. However, there are others, that are used in practically every metaheuristic. Two of them are the following:

Representation of solutions

The variables in metaheuristics are a set of quantities that need to be determined in order to solve certain problem. The variables are sometimes called *decision variables* because the problem is to decide what value each variable should take. An assignment of values to all decision variables in a problem is called a solution $\mathcal{S}$. Any metaheuristic needs an encoding of a solution. In order to have a good representation it should be taken into account how the solution will be evaluated and how the search operators will operate on it. A good representation must have *completeness* (all the

solutions are represented), *connexity* (any solution can be transformed into another when a neighborhood is defined) and *efficiency* (easy to manipulate by the search operators). In this thesis the terms candidate solution $\mathcal{S}$ and individual $\mathcal{I}$ are used interchangeably.

Objective, Fitness and Cost Functions

When we work with metaheuristics the terms *objective function* and *fitness function* are strongly related but there is a slight difference between them. The objective function is the actual objective of the problem and the fitness function is a quality measure of the solution that is defined with the aim of guiding the search towards high-quality solutions. Sometimes the fitness function is the same than the objective function and in other cases they differ.

When the aim is to maximize the values associated with each candidate solution, the quality measure of a candidate solution is called *fitness function* (maximization problem), whereas when the aim is to minimize the values associated with each candidate solution, it is more usual to use the term *cost function* (minimization problem).

An illustrative example of the importance of differentiating between these functions is the following. In linear programming, there are problems with constraints that must be satisfied and the objective function is to minimize the cost and fulfills the constraints. In such constrained problem fitness functions might contain two parts: the objective function and a penalty function which penalizes the value when constraints are violated. Another example arises in combinatorial problems. In many cases the objective function takes just a few distinct values. This is a drawback for the proper operation of metaheuristics because the search is not properly guided towards higher-quality regions. As a help for this drawback another fitness function can be defined to distinguish among solutions with the same objective function.

Some general ways to alter the fitness function have been defined [2] by scaling the objective function: *linear static scaling*, *linear dynamic scaling*, *logarithmic scaling*, *exponential scaling* and *sigma truncation*. For some metaheuristics these kinds of transformation are quite important and might be the key to success. Some studies that relate selection probabilities and these kinds of transformations are included in [9].

2.1.1 Trajectory-Based Metaheuristics

Trajectory-based metaheuristics, also called *single solution based*, are a set of metaheuristics that operate in each step with a single solution $\mathcal{S}$. This kind of metaheuristics are exploitation oriented, i.e. they are usually proper to attain good solutions of a specific region of the search space. However, they

Algorithm 1: General Operation of Trajectory-Based Metaheuristics

Input : Initial solution $\mathcal{S}_0$

$t = 0$

repeat

  /* Generate candidate solutions $\mathcal{S}'$ from $\mathcal{S}_t$ */

  Generate($\mathbf{C}(\mathcal{S}_t)$)

  /* Select a solution $\mathcal{S}'$ from ($\mathtt{C}(\mathcal{S}_t)$) to replace $\mathcal{S}_t$ */

  $\mathcal{S}_{t+1} = \mathcal{S}'$

  $t = t + 1$

until stopping criterion satisfied

Output: Best solution found

are not so global, so many regions might remain unexplored even if long executions are performed. It operates by transforming a single solution $\mathcal{S}$ by performing "walks" through neighborhoods or search trajectories through the search space in order to improve it. Thus, a trajectory-based metaheuristic iteratively applies *generation* and *replacement* procedures from the current single solution $\mathcal{S}$ until a given stopping criterion is reached. The generation procedure also called *move operator* is responsible for generating a set of candidate solutions $\mathbf{C}(\mathcal{S})$ from the current solution $\mathcal{S}$, this process is usually defined based on a neighborhood of the current solution $\mathcal{S}$, although it can also be done with other kind of mechanisms. The replacement procedure is responsible for selecting a new solution $\mathcal{S}'$ belonging to the candidate solution set ($\mathcal{S}' \in \mathbf{C}(\mathcal{S})$) to replace the current solution. The generation and replacement phases may be *memoryless*, which indicates that only take into account the current solution to operate. Algorithm 1 illustrates the general operation of trajectory-based metaheuristics [51]. In base of the general framework of trajectory-based metaheuristics, many different proposals have been defined. Some of the most popular ones are: *Hill Climbing* (HC) or *Local Search* (LS), *Simulated Annealing* (SA), *Iterated Local Search* (ILS), *Variable Neighborhood Search* (VNS) and *Guided Local Search* (GLS).

Common Concepts

Neighborhood. The definition of the neighborhood set plays an important role in the performance of trajectory-based metaheuristic. In combinatorial optimization, *the neighborhood $N(\mathcal{S})$ of a solution $\mathcal{S}$ is usually represented by*

$$N(\mathcal{S}) = \{\mathcal{S}' : d(\mathcal{S}', \mathcal{S}) \leq \epsilon\}, \tag{2.1}$$

where d represents a given distance between solutions. The natural neighborhood for binary representations — the one used in this thesis — is based on the Hamming distance d_H. In many cases, $\epsilon = 1$ is used, meaning that the neighborhood of a solution $\mathcal{S}$ consists in flipping one decision variable of the solution.

Local Optimum. Relatively to a given neighborhood N, a solution $\mathcal{S} \in N(\mathcal{S})$ is a local optimum if it has a better or equal quality than all of its neighbors, i.e.

$$cost(\mathcal{S}) \leq cost(\mathcal{S}') \quad \forall \mathcal{S}' \in N(\mathcal{S}). \tag{2.2}$$

Large Neighborhoods. In general, the larger the neighborhoods, the best results are obtained when applying a local search. However, due to the time limitations is not feasible to use very large neighborhoods. In order to deal with large neighborhoods, it is usual to apply heuristic search (sampling only one part) or methods that allow to find the best neighbor without exploring them all. For example, *dynamic programming* [3] and/or *branch and bound* [43] are typical methods used to explore large neighborhoods.

Initialization of Solutions. Two main strategies are used to generate the initial solution: *random* and *greedy* approaches. Random initial solutions are obtained quickly, but the metaheuristic might take many iterations to converge. In order to speed up the search, a greedy heuristic might be used for generating the initial solution. One of the inconveniences of this last approach is that when executed several times, even if a stochastic greedy approach is used, all the initial solutions might be in close regions.

Search Space. The search space of a problem can be visualized as a directed graph $G = (V, E)$, where the set of vertices V corresponds to the solutions of the problem, and the set of edges E is generated in base of the move operator. For instance, when the move operator is based on a neighborhood there is an edge between neighbor solutions.

Fitness Landscape. The fitness landscape can be defined by the tuple (G, f), where the graph G represents the search space and f represents the objective function that guides the search. The analysis of the fitness landscape is a very important aspect in the design of a metaheuristic in order to solve any optimization problem. There exist a relationship between the effectiveness of metaheuristics and the properties of the landscape. We can describe a landscape using some geographical terms. In this way considering the search space as the ground floor, we elevate each solution to an altitude equal to its quality. We obtain a landscape made of *valleys, plains, plateaus*, and so on.

Plateaus. Plateaus are connected regions of the search space that are equally fitted. This represents a tedious problem for the performance of metaheuristics because the fitness function is blind in such regions so there is no information to guide the search toward better regions. When a fitness function generates too many and large plateaus, it is usual to redefine the fitness function so it is important to check if this issue is arising.

Algorithm 2: Hill Climbing (HC)

Input : Initial solution $\mathcal{S} = \mathcal{S}_0$

```
1 while not stopping criterion do
      /* Generate candidate neighbors                    */
2     Generate(N(S))
3     if there is not better neighbor then
4         stop
5     S = Select a better candidate solution from N(S)
```

Output: Final solution found (local optimum)

Hill Climbing

Hill Climbing (HC) is the oldest and simplest metaheuristic method [1]. It starts at a given initial solution and at each iteration HC replaces the current solution by a neighbor that improves the fitness function. The search stops when no neighbor is better than the current solution, which means that a local optimum is reached. Algorithm 2 illustrates the operation of HC.

Starting with an initial solution $\mathcal{S}_0$, HC generates a sequence $\mathcal{S}_1, \mathcal{S}_2, \ldots, \mathcal{S}_k$ of solutions with the following characteristics:

- The size of the sequence k is unknown a *priori*.
- $\mathcal{S}_{i+1} \in N(\mathcal{S}_i),\ \forall i \in [0, k-1]$.
- $cost(\mathcal{S}_{i+1}) < cost(\mathcal{S}_i),\ \forall i \in [0, k-1]$.
- $\mathcal{S}_k$ is a local optimum: $cost(\mathcal{S}_k) \leq cost(\mathcal{S}),\ \forall \mathcal{S} \in N(\mathcal{S}_k)$.

Variants. In the previous pseudo-code the way of selecting a neighbor is not specified. There are basically three distinct strategies for performing such a selection.

1. Simple Hill Climbing: The first neighbor which improves the current cost function is selected. The neighborhood is explored using a deterministic pre-specified order.
2. Steepest Hill climbing: It evaluates the complete neighborhood an it selects the neighbor that produces the largest improvement. If there is a tie, random selection among them is usually considered.
3. Stochastic hill climbing: the neighborhood is explored in a random order and the first found solution that improves the current solution is selected.

Algorithm 3: Iterated Local Search (ILS)

Input : Initial solution $\mathcal{S}_0$

```
/* Apply a given local search algorithm                    */
```

1 $\mathcal{S}_* =$ Local Search($\mathcal{S}_0$)

2 **repeat**

```
    /* Perturb the obtained local optimum                  */
```

3 $\quad \mathcal{S}' =$ Perturb($\mathcal{S}_*$, search history)

```
    /* Apply local search on the perturbed solution        */
```

4 $\quad \mathcal{S}'_* =$ Local Search($\mathcal{S}'$)

```
    /* Accepting criteria                                  */
```

5 $\quad \mathcal{S}_* =$ Accept($\mathcal{S}_*, \mathcal{S}'_*$, search memory)

6 **until** Stopping criterion satisfied

Output: Best solution found

Iterated Local Search

Iterated Local Search (ILS) is based on the simple principle of perturbing the current local optimum and then apply local search again. Stutzle [50] generalized this kind of heuristic. First, a local search is applied to an initial solution. Then, a perturbation of the obtained local optimum is carried out. Finally, a local search is applied again to the perturbed solution. The new generated solution is accepted as the new current solution under some conditions. This process iterates until a given stopping criterion is reached. Algorithm 3 describes the general operation of ILS. The most important design decisions in ILS are the selection of the neighborhood and the **perturbation method**. Additionally, instead of using simple local search, another trajectory-based method might be applied. The **perturbation method** can be seen as a large random move of the current solution keeping a percentage of the solution and perturbing the rest. In many cases, ILS is integrated with population-based approaches. In such cases, perturbations that are not very disruptive are usually applied. Finally, the **acceptance criteria** defines the conditions that the new local optimum must satisfy to replace the current solution.

Simulated Annealing

Simulated Annealing (SA) is a trajectory-based metaheuristic [22] that is inspired by the annealing process in metallurgy. Its main feature is that it enables the acceptance of worsening solutions (with certain probability) in order to escape from local optima. Algorithm 4 describes the SA operation. In SA algorithm the temperature update is diminished step by step. Most of the time, it is performed in a way such that

$$T_i > 0, \quad \forall i \quad \text{and} \quad \lim_{i \to \infty} T_i = 0. \tag{2.3}$$

Algorithm 4: Simulated Annealing (SA)

Input : Cooling schedule

$\mathcal{S} = \mathcal{S}_0$; /* Generation of the initial solution */
$T = T_{max}$; /* Starting temperature */
repeat
  /* At a fixed temperature */
  repeat
    Generate a random neighbor s' from s
    $\Delta E = cost(\mathcal{S}') - cost(\mathcal{S})$
    if $\Delta E \leq 0$ **then**
      $\mathcal{S} = \mathcal{S}'$ /* Accept the neighbor solution */
    else
      Accept $\mathcal{S}'$ with a probability $\exp(-\Delta E/T)$
  until Equilibrium condition
  /* e.g. a given number of iterations executed at each T */
  $T = g(T)$; /* Temperature update */
until Stopping criterion satisfied (*e.g.* $T < T_{min}$)

Output: Best solution found

The temperature can be updated in several ways:

- **Linear**: $T_i = T_0 - i \times \beta, \quad \beta$ is a constant value.
- **Geometric**: $T = \alpha T, \quad \alpha \in [0, 1]$. This is the most popular one.
- **Logarithmic**: $T_i = T_0 / \log(i)$.
- **Very slow decrease**: $T_{i+1} = T_i / (1 + \beta T_i)$.
- **Non monotonic**: it might increase the temperature to encourage the diversification in the search space.
- **Adaptive**: dynamic decreasing rate depending on information obtained during the search.

It is easily to see that the first four procedures to the temperature update guarantees that the 2.3 conditions hold.

In this thesis we focus in the three trajectory search methods described above. They were implementing with the aim of integrating them in a population-based metaheuristic because in order to achieve proper solutions, long executions were performed. In such circumstances, population-based metaheuristics are required to further balance the search towards exploration. The details of each implementation and adaptation to the optimization problem at hand are given in chapter 3.

Algorithm 5: General Operation of Population-Based Metaheuristics

Input : Initial population $P = P_0$

1 $t = 0$
2 **repeat**
  /* Generate a new population */
3 Generate(P'_t)
  /* Select new population */
4 P_{t+1} =Select_Population($P_t \cup P'_t$)
5 $t = t + 1$
6 **until** stopping criterion satisfied

Output: Best solution(s) found

2.1.2 Population-Based Metaheuristics

Population-based metaheuristics can be viewed as an iterative improvement in a population of solutions P. The main purpose is to improve iteratively a whole population of individuals — candidate solutions — by applying a set of operators. First, the population is initialized usually with random initialization. Then, a new population of solutions is generated and this new population is incorporated to the current one using some replacement selection procedures. The process ends when a stopping criterion is satisfied. In the reproduction phase, a new population of solutions P' is created. In the replacement phase, a selection is carried out from the current and the new populations. The reproduction and replacement phases may be *memoryless*, which indicates that the procedures are based only in the current population. Algorithm 5 illustrates the general operation of population-based metaheuristics.

Algorithms such as *Evolutionary Algorithms* (EAs), *Scatter Search* (SS), *Estimation of Distribution Algorithms* (EDAs), *Particle Swarm Optimization* (PSO), *Bee Colony* (BC), and *Artificial Immune Systems* (AISs) belong to the population-based metaheuristics.

Evolutionary Algorithms

In this thesis we focus the research in EAs, which are the most studied population-based algorithms. Their success in solving difficult optimization problems in various domains has promoted the field known as *Evolutionary Computation* (EC). An EA uses mechanisms inspired by biological evolution of the species, such as *selection* (competition between individuals), *reproduction* (crossover, mutation) and *replacement* (survival of the best individuals). EAs often attain well approximate solutions to many optimization problems and are easy to implement. Algorithm 6 illustrates the template of an EA. There are different types of EAs, such as: *Genetic Algorithm* (GA), *Evolution Strategy* (ES) and *Genetic Programming* (GP) among others.

Algorithm 6: Template of an Evolutionary Algorithm (EA)

```
1 Initialization(P(0))          /* Initial population with N individuals */
2 Evaluation(P_0)                /* Set quality to each individual */
3 Set t = 0                      /* Generation counter */
4 while not stopping criterion do
5     P'(t) = Selection(P(t))    /* Select parents to generate the
        offspring */
6     P'(t) = Reproduction(P'(t))   /* Create the child population */
7     Evaluate(P'(t))            /* Set quality to each offspring */
8     P(t+1) =Replacement(P(t), P'(t)   /* Select individuals for next
        generation */
9     t = t + 1
```

Genetic Algorithms (GAs) are the most popular class of EAs. This metaheuristic was developed by J. Holland in 1970s [17]. Traditionally, GAs are associated with the use of a binary representation, but nowadays one can find GAs that use other types of representations. Most typical GAs work by choosing the parents by the proportional selection method, which give to the best individuals a higher probability to become a parent for the offspring. The reproduction phase in GAs is performed by applying a crossover operator to two parents, any time that the crossover operator is applied to two parents, two offspring are generated. Then a mutation that randomly modifies the offspring is applied in order to promote diversity. The most usual replacement technique in GAs is generational, that is, the parents are replaced systematically by the offsprings. However, nowadays including elitism is also very typical, meaning that the best individual found along the search always remains in the population.

Evolution Strategy (ES) are another kind of EAs. This metaheuristics were originally developed by Rechenberg and Schewefel in 1964 [42]. This kind of metaheuristics are based on the principle of string causality: *small changes have small effects* and are mostly applied to continuous optimization problems where representations are based on real-valued vectors. The selection operator is deterministic and is based on the fitness ranking, so the best individuals have higher probability to become parents. In an ES, there is a distinction between the population of parents of size μ and the population of the offsprings of size $\lambda \geq \mu$. An individual is composed of the float decision variables plus some other parameters guiding the search. Thus, an ES facilitates a kind of self-adaptation by evolving the solution as well as the strategy parameters (e.g., mutation step size) at the same time. ESs usually apply a normally (Gaussian) distributed mutation and an elitist replacement. Crossover is rarely used, at least in the most standard variants.

Genetic Programming (GP) is a more recent evolutionary approach, which extends the generic model of learning to the space of programs [23]. The main difference between GP and other EAs is that instead of using a linear

representation for the solutions in the population it uses programs (nonlinear representation based on trees) to solve the problems. GP is a form of program induction that allows to automatically generate programs that solve a given task. In general, the parent selection is a fitness proportional, the crossover operator is based on subtrees exchange, the mutation is based on random change in the tree and the replacement is generational. One of the problems in GP is the uncontrolled growth of trees. Nowadays GP is widely used in machine learning and data mining tasks such as prediction and classification.

Memetic Algorithms

A very successful way to improve the performance of EAs is to hybridize with local search or other trajectory-based techniques. In fact, combinations of genetic algorithms and trajectory-based search heuristics, often called ***Memetic Algorithms*** (MAs), have been applied successfully to many combinatorial optimization problems. According to Moscato [34] MAs are an hybridization between EAs and refinement procedures. Norman and Moscato [36] implemented the first MA.

As it is said at the beginning of the chapter, the two main goals desired to achieve in the design of metaheuristics is to perform the *exploitation* and *exploration* in a proper way. A memetic algorithm allows to do this efficiently. This is because the part concerning to the population method is oriented to explore the search space and the part concerning to the trajectory method is oriented on intensifying the search.

According to its hybridization, the MAs can be categorized into three different generations [35]:

- 1^{st} **generation**: refers to hybrid algorithms between evolutionary based (global search) and trajectory based (local search). Some examples of this category are Lamarckian Memetic Algorithm (LMA) and Baldwin Memetic Algorithm (BMA).
- 2^{nd} **generation**: refers to multiple individual learning methods within an evolutionary system exhibiting the principles of memetic *transmission* and *selection* in their design. Some examples of this category are Hyper-Heuristic, Meta-Lamarckian and Multi-Meme.
- 3^{rd} **generation**: refers a rule-based representation of local search co-adapted alongside candidate solutions within the evolutionary systems, thus capturing regular repeated features in the problem space, one example of this is Co-evolution of MAs.

Algorithm 7: Lamarckian Memetic Algorithm (LMA)

```
1  Initialize(P_0)
2  Evaluate(P_0)
3  Improvement(P_0)
4  t = 0
5  while not stopping criterion do
6      P'_t = Mating Selection(P_t)
7      P'_t = Reproduction(P'_t)
8      Evaluate(P'_t)
9      P'_t = Improvement(P'_t)
10     P_{t+1} = Replacement (P_t, P'_t)
11     t = t + 1
```

In a MA the individuals are improved locally by the refinement procedure: *local search, lifetime learning* or *individual learning*. In base of the modifications arising from the refinement procedure we can classify the MAs in two groups [8]:

- **Lamarckian Memetic Algorithm** (LMA): modifications are written back in every individual representation. Algorithm 7 illustrates how a LMA works.
- **Baldwin Memetic Algorithm** (BMA): modifications change the fitness of the individuals without altering its representation.

Description of LMA components

Initialize. This phase is responsible for generating N individuals in the population, the most usual is random generation.

Evaluate. This phase is responsible for assigning fitness or quality to each individual in the population. This operator makes use of the objective function.

Improvement. This phase is responsible for improving each individual belonging to the population. The most commonly is to use a local search procedure to improve the individuals.

Mating Selection. This phase is responsible for selecting those individuals among the entire population that will be the parents of the offspring. The parents are selected according to their fitness using one of the following strategies:

- *Roulette Wheel Selection:* assigns to each individual a selection probability and the best individuals have more probability to be chosen. This is a very elitist procedure and in many cases it does not provide enough diversity to attain high-quality solutions.
- *Tournament Selection:* consist in randomly select k individuals (size of the tournament) and select the best one. This procedure is less elitist and provides more diversity. A binary tournament ($k = 2$) is used in the implementations for this thesis.

Reproduction. This phase is responsible for generating the offspring population. Valid solutions are produced by inheriting some characteristics from their parents through the application of variation operators such as *crossover* and *mutation*.

- **Crossover.** This procedure is applied to a pair of parents. For binary representations the common crossover operators are based on interchanging segments of their parents with a certain probability p_c. Three popular procedures to perform this exchange are the following:
  1. *1-point:* cut the parents in a random position k to do the exchanges. The resulting offspring is obtained up to the point from one parent and from the point from the another parent.
  2. *n-point:* cut the parents in p random positions $(k_1, ..., k_p)$ to do the exchanges. The p positions are treated equally.
  3. *Uniform:* each element of the offspring is selected randomly from either parent. Each parent contribute equally generating the offspring. A slight variant of this method is used in this thesis.
- **Mutation.** This operator acts on a single individual doing small changes with certain probability p_m. An usual value for p_m is $1/k$ where k is the number of decision variables, so in average only one variable is mutated.

  The fundamental aspects to be taken into account in the mutation are: *ergodicity* (allow every solution of the search space to be reached), *validity* (produce valid solutions) and *locality* (produce minimal changes in the solutions with a high probability). In EAs the mutation is related to neighborhood operators of trajectory metaheuristics and for binary representations the commonly used mutation is the flip operator. In this thesis we use two flips from truth table positions containing 0 and 1, so it can be seen as a swap between them.

Replacement. This phase is responsible for selecting the individuals that pass to the following generation. Modifications in this phase plays a very important role in our proposal for the diversity management, further details are described below and in the chapter 3.

Algorithm 8: RMDDC Survivor Selection Technique

Input : Population, offspring
Output: New Population;
Current = Population ∪ Offspring;
foreach $I \in$ Current **do**
    I_{cost} = Fitness associated to individual I.
Best = Individual with lowest fitness in Current;
NewPop = {Best} ;
Current = Current \ {Best} ;
/* Update D taking into account the elapsed time (T_e), stopping criterion (T_s) and initial value of D (D_I). */
$D = D_I - D_I \times T_e / T_s$;
while |NewPop|<N **do**
    foreach $I \in$ Current **do**
        I_{dist} = distance to the closest individual of I in New Pop;
        if $I_{dist} < D$ **then**
            $I_{cost} = \infty$;
    ND = Non-dominated individuals of Current (without repetitions) taking into account I_{dist} and I_{cost} as the two objectives.;
    Selected = Randomly select an individual from ND;
    NewPop = NewPop ∪ Selected ;
    Current = Current \ {Selected};
Population = NewPop;
Output: Population

Diversity Management

In EAs premature convergence is a drawback which represent a problem of efficiency. Due to this the EA could converge prematurely to a sub-optimal region. In order to mitigate this, some techniques have been developed. The different methods are classified in base of the component of the EA that is modified: *selection-based*, *population-based*, *reproduction-based*, *fitness-based* and *replacement-based*. The fundamental guideline used for replacement-based techniques is to change the replacement phase by diversifying the survivors, so more exploration is induced. Following this, Segura [46] designed the *Replacement with Multiobjective based Dynamic Diversity Control* (RMDDC) strategy. This is carried out by adopting multi-objective concepts and considering the diversity as an explicit objective. Algorithm 8 illustrates how the RMDDC strategy works for a minimization problem.

We need to clarify that increasing the diversity by itself is not a real objective when performing an optimization. However, it is an auxiliary objective that might enable to attain better results. For more details about the different procedures in diversity management the reader is referred to [47].

2.2 Cryptographic Boolean Functions

History

The work concerning to cryptographic BFs started with George Boole and Claude Elwood Shannon [14]. With his most important work *An Investigation into the Laws of Thought, on Which are Founded the Mathematical Theories of Logic and Probabilities* (1854), Boole redefined logic in a different way, transforming it into algebra. In his honor, the discipline of mathematics that studies logical expressions is called *Boolean Algebra*. The work of Boole was ignored for many years, until the 1940s when Claude Elwood Shannon picked up Boole's work and recognized its relevance to electronics design. The paper *Communication Theory of Secret Systems* (1949) by Shannon is the first analysis of cryptography [48].

2.2.1 Boolean Functions and their Representations

In Boolean Algebra {0,1} is the set of interest. The symbols 0 and 1 represents status of information, and they basically can be considered to be as a True (1) or False (0). These symbols can be combined by using electronic devices that perform operations on them. The most typical logic circuits take two values as an input and generate a new value. This is the case of the logic gates OR, AND and XOR. The logic gate NOT takes as input a single value and generates a new value. A way to explain the operation of logic gates is by using truth tables. Let A and B be Boolean variables representing the input of a logic gate. Then, the output of the logic gates is illustrated in table 2.1.

TABLE 2.1: Truth table of logic gates

A	B	AND (A$\wedge$B)	OR (A$\vee$B)	NOT ($\neg A$)	XOR (A$\bigoplus$B)
0	0	0	0	1	0
0	1	0	1	1	1
1	0	0	1	0	1
1	1	1	1	0	0

The set $\{0,1\}$ will be most often endowed with the structure of field (and denoted by $\mathbb{F}_2$) and the set $\mathbb{F}_2^n$ of all binary vectors of length n will be viewed as a $\mathbb{F}_2$ vector space. The null vector of $\mathbb{F}_2^n$ is $\mathbf{0}$. $\mathbb{F}_2^n$ is endowed with field structure to form the well-known Galois Field $\mathbf{GF}(2^n)$ [16]. A BF of n variables is a function

$$f : \mathbb{F}_2^n \to \mathbb{F}_2. \tag{2.4}$$

It is usually written as $f(\mathbf{x}) = f(x_1, x_2, ..., x_n)$, where $\mathbf{x}$ is the shorthand writing of vector $(x_1, x_2, ..., x_n)$. As an example, consider the 4-variable BF

$$f(\mathbf{x}) = f(x_1, x_2, x_3, x_4) = x_1 \bigoplus x_2 x_3 \bigoplus x_4 x_1. \tag{2.5}$$

There are several ways of representing BFs. The three representations exposed in the following are universal for BFs, meaning that each BF are represented uniquely in each kind of representation. For further information about conversions between representations of BFs, the interested reader is referred to [53].

Truth Table Representation

One of the most basic representation of a BF is by its *binary truth table* which is the *vector* of all outputs of the BF f lexicographically ordered. The binary truth table has length 2^n. The lexicographical order of the binary vectors $\mathbf{x}$ follow a particular order when going through all the possible values in $\mathbb{F}_2^n$. Let $k_\mathbf{x}$ denote the integer representation of $\mathbf{x} = (x_1, x_2, ..., x_n)$, *i.e.*,

$$k_\mathbf{x} = to\_int(\mathbf{x}) = \sum_{i=1}^{n} x_i 2^{n-i}. \tag{2.6}$$

We can see the function *to_int* as the function that maps a vector $\mathbf{x}$ from $\mathbb{F}_2^n$ to an integer $k_\mathbf{x} \in \mathcal{N}_n = \{0, \cdots, 2^n - 1\}$. When the integer $k_\mathbf{x}$ takes all the values from 0 incrementally to $2^n - 1$, then the corresponding binary vector $\mathbf{x}$ goes through all the elements in $\mathbb{F}_2^n$. This order is the lexicographical one.

Very linked to this representation is the *polarity truth table* or *polar form*. Let f denote a BF, then $\hat{f}$ is used to define the *polarity truth table*. In this case $\hat{f}(\mathbf{x}) \in \{-1, 1\}$ and each element of $\hat{f}$ is obtained as $\hat{f}(\mathbf{x}) = (-1)^{f(\mathbf{x})}$. Table 2.2 has the truth table and the polarity truth table of f 2.5.

TABLE 2.2: All the $\mathbf{x} \in \mathbb{F}_2^4$ with their corresponding integer values $k_\mathbf{x}$, the truth table f and the polarity truth table $\hat{f}$ of 2.5

$k_\mathbf{x}$	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
x_4	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1
x_3	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
x_2	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1
x_1	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
f	0	1	0	1	0	1	1	0	0	0	0	0	0	0	1	1
$\hat{f}$	1	-1	1	-1	1	-1	-1	1	1	1	1	1	1	1	-1	-1

Algebraic Normal Form Representation

In coding theory and cryptography a natural representation of BFs is to write the BF as a multivariate polynomial. A BF can be written as

$$f(\mathbf{x}) = c_0 \bigoplus_{1 \le i \le n} c_i x_i \bigoplus_{1 \le i < j \le n} c_{ij} x_i x_j \bigoplus \cdots \bigoplus c_{1,\ldots,n} x_1 x_2 \cdots x_n, \tag{2.7}$$

and it is called the *algebraic normal form* representation of f, or the ANF in brief, where $c_0, c_i, c_{ij}, \cdots, c_{1,\ldots,n}$ are coefficients having value in $\mathbb{F}_2$. It has been proved that every BF in n variables can be represented uniquely in the form of equation 2.7, i.e., with a unique set of coefficients. Thus, the representation of equation 2.7 is universal [6].

As an example, f 2.5 is the algebraic normal form of a BF in four variables. In such a case, almost all the coefficients are zero. The only exceptions are c_1, c_{23}, c_{41}.

Support Representation

A different way to store a BF is by recording the values of the variables where the BF takes value 1. The *support* of f — denoted as $supp(f)$ — is defined as

$$supp(f) = \{\mathbf{x} : f(\mathbf{x}) = 1, \mathbf{x} \in \mathbb{F}_2^n\}. \tag{2.8}$$

The support of f is a set of vectors of length n. For example, the support for f 2.5 is

$$supp(f) = \{(0001), (0011), (0101), (0110), (1110), (1111)\}.$$

If all these vectors are listed as a matrix X_f, the X_f is called the *characteristic matrix* of f which is unique when the rows are in an specific order, e.g., with the order defined above in expression 2.6.

Enumeration of Boolean Functions

$\mathcal{F}_n$ denotes the set of all n-variable BFs. The number of n-variable BFs is

$$|\mathcal{F}_n| = 2^{2^n}. \tag{2.9}$$

Affine Boolean Functions. An affine BF $L_{\mathbf{w},c}$ on $\mathcal{F}_n$ is a BF that takes the form

$$L_{\mathbf{w},c}(\mathbf{x}) = \mathbf{w} \cdot \mathbf{x} \bigoplus c = w_1 x_1 \bigoplus \cdots \bigoplus w_n x_n \bigoplus c, \tag{2.10}$$

where $\mathbf{w} \in \mathbb{F}_2^n$ and $c \in \mathbb{F}_2$. If $c = 0$, then $L_{\mathbf{w},0}$ ($L_{\mathbf{w}}$) is a linear BF.

The sets of n-variable affine and linear BFs are denoted by $\mathcal{A}_n$ and $\mathcal{L}_n$ respectively. The number of n-variable affine and linear BFs are

$$|\mathcal{A}_n| = 2^{n+1}, \tag{2.11}$$

$$|\mathcal{L}_n| = 2^n. \tag{2.12}$$

Basic Definitions

Hamming Weight. The numbers of $1's$ in the truth table of f is called the *Hamming weight* of f and is denoted by $w_H(f)$. As a consequence $w_H(f)$ is the number of elements in $supp(f)$.

$$w_H(f) = \sum_{\mathbf{x} \in \mathbb{F}_2^n} f(\mathbf{x}). \tag{2.13}$$

This definition is also true for vectors $\mathbf{x} \in \mathbb{F}_2^n$. Particularly, the hamming weight of a vector $\mathbf{x} \in \mathbb{F}_2^n$ is the amount of nonzero positions.

Hamming Distance. Let $f(\mathbf{x})$, $g(\mathbf{x})$ be two n-variable BFs, the Hamming distance between $f(\mathbf{x})$ and $g(\mathbf{x})$, denoted by $d_H(f,g)$, is the number of coordinates with different values in their truth tables. It can be written as

$$d_H(f,g) = w_H(f \oplus g) = \sum_{\mathbf{x} \in \mathbb{F}_2^n} f(\mathbf{x}) \oplus g(\mathbf{x}). \tag{2.14}$$

2.2.2 The Walsh Hadamard Transform

The Walsh Hadamard Transform (WHT) can be seen as another kind of n-variable BF representation. In this dissertation this is the most important representation of n-variable BFs, because it is used to calculate relevant cryptographic properties of BFs, between them the **nonlinearity**. Let f be a n-variable BF, the WHT of f is the function $W_n : \mathbb{F}_2^n \to \mathbb{Z}$ defined by

$$W_n(f)(\mathbf{w}) = \sum_{\mathbf{x} \in \mathbb{F}_2^n} (-1)^{f(\mathbf{x}) \oplus L_{\mathbf{w}}(\mathbf{x})}, \qquad \mathbf{w} \in \mathbb{F}_2^n. \tag{2.15}$$

Another intuitive way to calculate the WHT is by using the polarity truth tables of f and $L_{\mathbf{w}}$

$$W_n(\hat{f})(\mathbf{w}) = \sum_{\mathbf{x} \in \mathbb{F}_2^n} \hat{f}(\mathbf{x}) \hat{L}_{\mathbf{w}}(\mathbf{x}), \qquad \mathbf{w} \in \mathbb{F}_2^n. \tag{2.16}$$

The Walsh Hadamard inverse transform corresponding to 2.16 is

$$\hat{f}(\mathbf{x}) = 2^{-n} \sum_{\mathbf{w} \in \mathbb{F}_2^n} W_n(\hat{f})(\mathbf{w}) \hat{L}_{\mathbf{w}}(\mathbf{x}), \qquad \mathbf{x} \in \mathbb{F}_2^n. \tag{2.17}$$

The WHT of f measures the correlation between f and each $L_{\mathbf{w}}$. According to Millan [33], the correlation with $L_{\mathbf{w}}$ is given by

$$c(f, L_{\mathbf{w}}) = \frac{|W_n(f)(\mathbf{w})|}{2^n}. \tag{2.18}$$

The correlation is a real number $0 \leq c(f, L_{\mathbf{w}}) \leq 1$, that represents the degree of similarity between f and $L_{\mathbf{w}}$. A correlation value of zero indicates that f and $L_{\mathbf{w}}$ are completely uncorrelated. Differently, a value equal to 1 means a perfect correlation between f and $L_{\mathbf{w}}$.

Parseval's Equation

The values in the WHT of f are constrained by a square sum relationship which implicitly limits the magnitude and frequency of those values, this is known as *Parseval's Equation* (2.19) from [24]:

$$\sum_{\mathbf{w} \in \mathbb{F}_2^n} (W_n(f)(\mathbf{w}))^2 = 2^{2n}. \tag{2.19}$$

This value is constant for all n-variable BFs, *i.e.*, the values in the WHT of every BF must satisfy Parseval's equation. However, a WHT of a function can satisfy Parseval's equation and not necessarily be Boolean.

Hadamard Matrices

A Hadamard matrix H_n is a square matrix of dimension 2^n, such that

$$H_n H_n^T = 2^n I_{2^n}, \tag{2.20}$$

where H_n^T is the transpose of H_n, and I_{2^n} is the $2^n \times 2^n$ identity matrix. The entries of the matrix are either $+1$ or -1 and its rows and columns are orthogonal.

By defining $H_0 = (1)$, Hadamard matrices can be constructed with the following formula:

$$H_n = \begin{pmatrix} H_{n-1} & H_{n-1} \\ H_{n-1} & -H_{n-1} \end{pmatrix}. \tag{2.21}$$

When $n = 1, 2$, it is easy to formulate the matrices

$$H_1 = \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} \quad \text{and} \quad H_2 = \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \end{pmatrix}. \tag{2.22}$$

Let $\hat{f}$ be the polarity truth table representation of f and $W_n(\hat{f})$ the WHT of $\hat{f}$. Then, the WHT and the inverse WHT can be represented by means of matrices as follows:

$$W_n(\hat{f}) = \hat{f} \cdot H_n, \tag{2.23}$$

$$\hat{f} = 2^{-n} W_n(\hat{f}) \cdot H_n. \tag{2.24}$$

When equations 2.23 and 2.24 are used to compute the WHT, it is faster than to use the normal WHT 2.15 and 2.17. The Hadamard matrix representation can be used to formulate fast computation of WHT.

Fast Walsh Hadamard Transform

Direct calculation of W_n would require about 2^{2n} operations. This is because there exists 2^n linear functions and computing the correlation 2.18 with each linear function requires 2^n operations, which give us the total amount of operations required by the direct calculation of the WHT. Fortunately there is a faster way to obtain W_n, —the Fast Walsh Hadamard Transform (FWHT)— which is a discrete version of the so-called Fast Fourier Transform (FFT). This is possible because H_n can be written as a product of n matrices with dimensions $2^n \times 2^n$. Each matrix has only two non-zero elements per column. Thus only $n2^n$ operations are needed to evaluate W_n. In order to explain the decomposition of H_n the Kronecker products must be introduced.

Kronecker product of matrices

If A is an $m \times n$ matrix and B is a $p \times q$ matrix, then the Kronecker product $A \bigotimes B$ is the $mp \times nq$ matrix obtained from A by replacing every entry a_{ij} by $a_{ij}B$

$$A \bigotimes B = (a_{ij}B). \tag{2.25}$$

The Kronecker product is not commutative, i.e., in general $A \bigotimes B \neq B \bigotimes A$. However, it is associative, $(A \bigotimes B) \bigotimes C = A \bigotimes (B \bigotimes C)$ and distributive, $(A + B) \bigotimes C = A \bigotimes C + B \bigotimes C$.

Hadamard matrices can be defined using the Kronecker product

$$H_n = H_1 \bigotimes H_{n-1}. \tag{2.26}$$

As a consequence any Hadamard matrix H_n can be decomposed as

$$H_n = M_n^{(1)} M_n^{(2)} \cdots M_n^{(n)}, \tag{2.27}$$

where $M_n^{(i)} = I_{2^{n-i}} \bigotimes H_1 \bigotimes I_{2^{i-1}}$, and I_m is the $m \times m$ identity matrix. The proof of 2.27 is easy by induction and can be found in [24]. Example, for $n = 2$

$$M_2^{(1)} M_2^{(2)} = \begin{pmatrix} 1 & 1 & 0 & 0 \\ 1 & -1 & 0 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & -1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 1 & 0 & -1 & 0 \\ 0 & 1 & 0 & -1 \end{pmatrix}$$

$$M_2^{(1)} M_2^{(2)} = \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \end{pmatrix}$$
$$M_2^{(1)} M_2^{(2)} = H_2.$$

An interesting remark is that the matrices $M_n^{(.)}$ commute with each other, i.e., $M_n^{(i)} M_n^{(j)} = M_n^{(j)} M_n^{(i)}$, for any i, j.

The sparse matrix method of the equation 2.27 allows to compute the W_n using only $n2^n$ operations. This can be exemplified when is used the concept of butterfly diagram. A butterfly diagram of size two, takes two inputs (x_0, x_1) and gives two outputs (y_0, y_1): $y_0 = x_0 + x_1$, $y_1 = x_0 - x_1$.

FIGURE 2.1: Fast WHT applied to the BF 2.5 to obtain its WHT

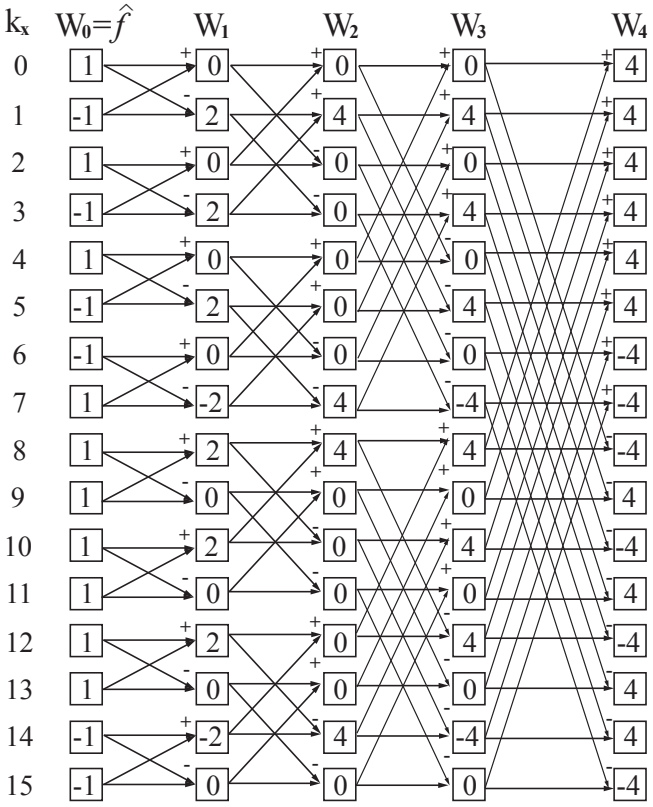


Figure 2.1 shows an example of use of the Fast WHT applied to f 2.5. Intuitively, we can see any W_i ($i \in \{1, \cdots, n\}$) like a i-phase in the computation of the WHT (W_0 is the 0-phase and corresponds to the polar form of the BF). It is easy to see that we need to have computed the $(i-1)$-phase to compute the i-phase. So the FWHT works by computing n phases orderly. More details about this can be found in the section 3.1.3.

2.2.3 Cryptographic Properties of Boolean Functions and Special Functions

Boolean functions used in stream ciphers must have some required properties. These properties are designed to make ciphers secure against some attacks. Some of the most important cryptographic properties of BFs are briefly explained in the following.

Algebraic Degree

Let $deg(f)$ denote the algebraic degree of f. The algebraic degree is defined to be the number of variables in the largest product term of the algebraic normal form 2.7 of f having a non-zero coefficient. The affine BFs which includes the linear BFs have an algebraic degree of 1.

Balancedness

The most basic of all cryptographic properties desired to be exhibited by BFs is balance, f is said to be balanced if $w_H(f) = 2^{n-1}$. In terms of its WHT, a BF f is balanced if and only if

$$W_n(f)(\mathbf{0}) = 0. \tag{2.28}$$

If the magnitude of $w_H(f)$ deviates from 2^{n-1}, the function can be better approximated with a constant function. This is one of the reasons for the importance of balancedness Thus, the most basic of all cryptographic properties desired to be exhibited by BFs is balance. The set of n-variable balanced BF is denoted as $\mathcal{B}_n$ and its size is

$$|\mathcal{B}_n| = \binom{2^n}{2^{n-1}}. \tag{2.29}$$

It is important to remark that we use this search space along this thesis.

Nonlinearity

The nonlinearity $N_n(f)$ of f is calculated using the maximum absolute value of the WHT and represents the minimum Hamming distance between f and the affine BFs set $\mathcal{A}_n$, i.e.

$$N_n(f) = \min_{\phi \in \mathcal{A}_n} d_H(f, \phi). \tag{2.30}$$

Mathematically, the relationship between the nonlinearity of an n-variable BF f and the WHT of f is given by the following equation:

$$N_n(f) = \frac{1}{2}\left(2^n - \max_{\mathbf{w}\in\mathbb{F}_2^n} |W_n(f)(\mathbf{w})|\right), \tag{2.31}$$

where $\max_{\mathbf{w}\in\mathbb{F}_2^n} |W_n(f)(\mathbf{w})|$ represents the maximum absolute value in the WHT. By Parseval's equation, we can obtain an upper bound of nonlinearity in general case (when n is even), that is

$$N_n(f) \leq 2^{n-1} - 2^{n/2-1}. \tag{2.32}$$

Auto Correlation

The auto correlation of f $AC(f)$ is the maximum absolute value of $r_f(\mathbf{x})$, $\mathbf{x} \neq \mathbf{0}$ such that r_f is the auto correlation spectrum of f

$$AC(f) = \max_{\mathbf{x}\in\mathbb{F}_2^n\setminus\{\mathbf{0}\}} |r_f(\mathbf{x})| \quad \text{s.t.} \quad r_f(\mathbf{w}) = \sum_{\mathbf{x}\in\mathbb{F}_2^n} (-1)^{f(\mathbf{x})+f(\mathbf{x}\oplus\mathbf{w})}. \tag{2.33}$$

Algebraic Immunity

The algebraic immunity (AI) is defined as the minimum degree of the nonzero function g such that g is known as the *annihilator* of f (or $f \oplus 1$)

$$AI(f) = \min\{deg(g) : fg = 0 \vee (f \oplus 1)g = 0, g \in \mathcal{F}_n\}. \tag{2.34}$$

Fast Algebraic Resistance

The fast algebraic resistance (FAR) of f is defined as the minimum value of $deg(g) + deg(h)$ for all $(deg(g), deg(h))$- relations on f, *i.e.*

$$FAR(f) = \min\{deg(g) + deg(h) : fg = h \wedge deg(g) < deg(h)\}. \tag{2.35}$$

FAR is upper bounded by $deg(f)$. This means that any cost function dealing with FAR also deals to some extent with algebraic degree.

Bent Functions

Bent BFs are a very special class of BFs. A BF $f(\mathbf{x})$ is bent if and only if

$$|W_n(f)(\mathbf{w})| = 2^{n/2}, \ \forall \mathbf{w} \in \mathbb{F}_2^n. \tag{2.36}$$

Bent BFs are not balanced and they are usually not applied in cryptosystems. It can be noticed that bent BFs only exists when n is even.

Correlation Immune Functions

Correlation immunity is an interesting cryptographic property that measures the level of resistance against correlation attacks. A BF with low order correlation immunity is more susceptible to a correlation attack than a BF of high order correlation immunity. The correlation immunity m of a BF of algebraic degree d of n variables satisfies $m + d \leq n$; for a given set of input variables, this means that a high algebraic degree will restrict the maximum possible correlation immunity. Furthermore, if the BF is balanced then $m + d \leq n - 1$.

A BF is correlation immune of order m (in brief $CI_f(m)$) if the output of the BF is statistically independent of the combination of any m of its inputs. According to Picek [39], for the WHT it holds that

$$W_n(f)(\mathbf{w}) = 0; \quad \text{for } 1 \leq w_H(\mathbf{w}) \leq m, \qquad \mathbf{w} \in \mathbb{F}_2^n. \tag{2.37}$$

Sarkar and Maitra [44] showed that if a $CI(m)$ BF f has an even number of variables n and $k \leq n/2 - 1$, then its nonlinearity $N_n(f)$ has the upper bound as follows:

$$N_n(f) \leq 2^{n-1} - 2^{n/2-1} - 2^k, \tag{2.38}$$

where k equals $m + 1$ if f is balanced or has Hamming weight divisible by 2^{m+1} and k equals m otherwise. In the case when $k > n/2 - 1$ then the nonlinearity has the upper bound

$$N_n(f) \leq 2^{n-1} - 2^k. \tag{2.39}$$

In the following, we concentrate only on BFs where $m + 1 \leq n/2 - 1$. This bound is called Sarkar-Maitra divisibility bound and due to this we know that the maximum possible nonlinearity for 8-variable BFs is 118, that occurs when $m = 0$. In this way the nonlinearity upper bound for any n-variable BF with n even is

$$N_n(f) \leq 2^{n-1} - 2^{n/2-1} - 2. \tag{2.40}$$

Resilient Functions

When correlation immune BFs are balanced, they are also called resilient BFs. It is said that a balanced BF with $CI(m)$ is m-resilient. In other words, the BF f is called m-resilient if $w_H(f') = 2^{n-m-1}$ for any its sub-function f' of $n - m$ variables. From this point of view we can consider formally any balanced BF as 0-resilient. The concept of an m-resilient function was introduced in [10].

2.3 Metaheuristics for Cryptographic Boolean Functions

There are many works related to finding BFs with high nonlinearity using metaheuristics. These methods can be classified in three groups: trajectory-based metaheuristics, population-based metaheuristics and hybrid metaheuristics that combine algebraic constructions and (mainly) local search procedures in order to construct BFs with high nonlinearity.

2.3.1 Trajectory-Based Metaheuristics

Smart Hill Climbing

The Smart Hill Climbing (SHC) method developed by Millan [33] is a quite simple strategy. It does not use any cost function explicitly, *i.e.* it uses the maximum absolute value in the WHT directly, which is implicitly related with the nonlinearity, so we can deduct that they made use of it to guide the search. They consider defining some sets involving certain linear functions $L_{\mathbf{w}}(\mathbf{x})$ that have minimum Hamming distance to $f(\mathbf{x})$. The purpose of defining these sets is to ensure that a change always improve the nonlinearity of f. Thus, the method identifies the linear functions that have more correlation with f and by properly complementing certain truth table positions the correlation to those linear functions is diminished.

Let X_α denote the maximum absolute value in the WHT of the BF f.

$$X_\alpha = \max_{\mathbf{w}\in\mathbb{F}_2^n} |W_n(\mathbf{w})|.$$

SHC works by defining the following sets

$$\begin{aligned}
\mathcal{A}_1^+ &= \{\mathbf{w} : W_n(\mathbf{w}) = \quad X_\alpha, \quad \mathbf{w} \in \mathbb{F}_2^n\} \\
\mathcal{A}_1^- &= \{\mathbf{w} : W_n(\mathbf{w}) = -X_\alpha, \quad \mathbf{w} \in \mathbb{F}_2^n\} \\
\mathcal{A}_2^+ &= \{\mathbf{w} : W_n(\mathbf{w}) = \quad X_\alpha - 4, \mathbf{w} \in \mathbb{F}_2^n\} \\
\mathcal{A}_2^- &= \{\mathbf{w} : W_n(\mathbf{w}) = -X_\alpha + 4, \mathbf{w} \in \mathbb{F}_2^n\}.
\end{aligned}$$

The main idea behind this method is to decrease the maximum absolute values that appear in the WHT. It is easy to proof that when two different truth table values are complemented the change in the WHT $W_n(\mathbf{w})$ is -4,0 or 4, this is why we need to identify the WHT entries where the value $|X_\alpha - 4|$ appear. In order to improve the nonlinearity it is required to find a pair of truth table positions $\mathbf{x}_1, \mathbf{x}_2$, such that $f(\mathbf{x}_1) \neq f(\mathbf{x}_2)$. When these truth table values are complemented

$$f(\mathbf{x}_1) = f(\mathbf{x}_1) \oplus 1, \qquad f(\mathbf{x}_2) = f(\mathbf{x}_2) \oplus 1, \tag{2.41}$$

Algorithm 9: Smart Hill Climbing (SHC) - Related Work

Input : n-variable BF f

```
1  LOF=False;                                   /* Local optimum found flag */
2  while LOF=False do
3      Calculate the WHT W_n ;
4      Obtain the sets A_1^+, A_1^-, A_2^+, A_2^- ;
5      Set C_0 = ∅, C_1 = ∅ ;
6      foreach x ∈ F_2^n do
7          if f(x) = L_w(x), ∀w ∈ A_1^+ then
8              if f(x) ≠ L_w(x), ∀w ∈ A_1^- then
9                  C_{f(x)} = C_{f(x)} ∪ x ;
10     if C_0 = ∅ ∨ C_1 = ∅ then
11         LOF = True;
12     flag=1;
13     foreach x ∈ C_0 & flag=1 do
14         foreach y ∈ C_1 & flag=1 do
15             if f(x) ≠ L_w(x) & f(y) ≠ L_w(y), ∀w ∈ A_2^+ then
16                 LOF = True;
17             else
18                 if f(x) = L_w(x) & f(y) = L_w(y), ∀w ∈ A_2^- then
19                     LOF = True;
20                 else
21                     f(x) = f(x) ⊕ 1 ;
22                     f(y) = f(y) ⊕ 1 ;
23                     LOF = False;
24                     flag = 0;
```

Output: Local optimum f

the changes in the WHT do not increase the quantity $|X_\alpha - 4|$ and decreases the quantity $|X_\alpha|$. The algorithm 9 illustrates how the SHC works. The sets C_0, C_1 in the 5th line stores the truth table positions that can be complemented in order to diminish $|X_\alpha|$. Since the values corresponding to $|X_\alpha - 4|$ could increase and remain the same nonlinearity, we need to identify those pairs $\mathbf{x}, \mathbf{y}$ such that when are complemented (lines 21,22), the values $|X_\alpha|$ decreases and $|X_\alpha - 4|$ do not increases. In order to identify those where $|X_\alpha - 4|$ do not increases we need to make sure that the conditions on lines 15 and 18 are not met and in this way the nonlinearity of f is increased. The best results obtained by Millan with the SHC are BFs with nonlinearity $N_8(f) = \mathbf{112}$ for 8 variables and $N_{10}(f) = \mathbf{474}$ for 10 variables after starting from 10000 BFs.

Algorithm 10: Simulated Annealing (SA) - Related Work

Input : Solution $\mathcal{S}_0$, cooling schedule (T_0, α),*MIL*,*MaxIL*

```
S = S_0;
T = T_0;
for it = 0; it < MaxIL; it++ do
    for i = 0; i < MIL; i++ do
        Select neighbor S';
        δ = C_XR(S') − C_XR(S);
        if δ ≤ 0 then S = S';
        else
            Generate u = U(0,1);
            if u < exp(−δ/T) then S = S';
    T = T × α;
```

Output: Best solution found $\mathcal{S}$

Simulated Annealing

The first known SA method to address this problem was proposed by Clark [11]. Clark works with many versions of SA, the SA version which provides him the best results can be found in [13], however, additional results concerning to this method can be found in [12]. The main aim of the application of SA is to avoid getting stuck into low-quality local optima. The SA method proposed by Clark encourage improving moves but allows some worsening moves to be accepted as a mean to escape from local optima. Algorithm 10 illustrates how the SA proposed by Clark works. The initial temperature T_0 is not described in the works of Clark, where the remaining parameters setting for SA is $MaxIL = 400$ which is the maximum number of outer iterations for the search, $MIL = 400$ which is the maximum number of inner iterations at fixed temperature and $\alpha = 0.95$ which is the geometric cooling parameter.

The main contribution from Clark is the development of another cost function to guide the search. The cost function C_{XR} that he developed can be derived by considering the Parseval's equation 2.19 and it is as follows

$$C_{XR} = \sum_{\mathbf{w} \in \mathbb{F}_2^n} \left(|W_n(\mathbf{w}) - X| \right)^R, \tag{2.42}$$

where X and R are parameters to set, Clark found that proper values for these parameters are $R = 3.0$ and $X = 12$. Thanks to this new cost function, Clark was able to overcome the results obtained by Millan for the 10-variable case. Clark was able to reach BFs with nonlinearity $N_8(f) = \mathbf{116}$ and $N_{10}(f) = \mathbf{486}$ for 8 and 10 variables respectively, employing a two-stage optimization method, which basically consists of an annealing-based search using the cost function C_{XR} followed by SHC method 9 to improve the nonlinearity.

Following the work from Clark, Kavut [21] develop another cost function 2.43 to be used in the SA method proposed by Clark.

$$C_{RS} = \sum_{\mathbf{w}\in\mathbb{F}_2^n} |W_n(\mathbf{w})|^R + \sum_{\mathbf{w}\in\mathbb{F}_2^n\setminus\{\mathbf{0}\}} |r_f(\mathbf{w})|^S, \tag{2.43}$$

where R and S are parameters to set and $r_f(\mathbf{w})$ is the auto correlation spectrum 2.33 of the BF f. The values corresponding to these parameters are $R = 3, S = 3$. Kavut works with a three-stage optimization method which is based on applying simulated annealing using the cost function 2.43, then use a hill climbing technique with respect to the cost function 2.43 and finally use the SHC 9 to improve the nonlinearity. Kavut was able to find BFs with nonlinearity $N_8(f) = \mathbf{116}$ and $N_{10}(f) = \mathbf{486}$ for 8 and 10 variables respectively.

Another researcher that works using SA is McLaughlin [29]. The novelty of the work by McLaughlin is that his experiments do not make use of cost function focus in nonlinearity. He employed the SA along with the algebraic immunity (AI 2.34) and the fast algebraic resistance (FAR 2.35)

$$fit(f) = AI(f) + FAR(f). \tag{2.44}$$

The computation of the AI and FAR is exponential in memory and time. However, McLaughlin found BFs with nonlinearity $N_8(f) = \mathbf{116}$ and $N_{10}(f) = \mathbf{488}$ for 8 and 10 variables respectively. He was able to obtain these results for $n = 10$ in the 32% of the functions tasted after several days of execution (is not mentioned how many).

2.3.2 Population-Based Metaheuristics

Genetic Algorithm

The genetic algorithm developed by Millan [32] is the first known population-based metaheuristic to find BFs with high nonlinearity. It uses the balanced BFs search space $\mathcal{B}_n$ (see 2.29) and the nonlinearity $N_n(f)$ as the fitness function to guide the search. In order to generate the offspring they define a merging operator that combines two parents (which should be close) to produce a single offspring. The offspring is also balanced and is modified with a random mutation incorporated in the merging operator. Algorithm 11 describes the working operation of the GA developed by Millan. The novelties of this GA is the merging operator and the incorporation of the SHC to improve the offspring generated. This method can be considered as the first LMA used to find BFs with high nonlinearity. The SHC method was described before. Algorithm 12 illustrates how the merging operator works. The condition in the line 3 is used to ensure that only parents which are close to each other are allowed to breed. It should be noted that complementing the truth table of a BF does not alter its

Algorithm 11: Genetic Algorithm (GA) - Related Work

Input : Population size N

```
1 Initialize(P_0) ;                          /* Generate N random balanced BFs */
2 Evaluate(P_0) ;                  /* Calculate the nonlinearity of each BF */
3 for i = 0; i < MAXITER; i++ do
4     P'_i =Merging(P_i);                    /* Generate N(N-1)/2 offspring */
5     Evaluate(P'_i);
6     (Optional SHC) ;    /* Apply the SHC (alg 9) to each offspring */
7     P_{i+1} =Replacement(P_i, P'_i); /* Select the N best as the new pop */
```

Output: Best solution found

Algorithm 12: Merging - Related Work

Input : Two BFs as the parents f_1, f_2

```
1  Set c = f_1;
2  Set n_1 = 0 and k = 0;
3  if d_H(f_1, f_2) > 2^n/2 then Complement f_1 or f_2 ;
4  foreach x ∈ F_2^n do
5      if f_1(x) = f_2(x) then c(x) = f_1(x) ;
6      else
7          if n_1 = d_H(f_1, f_2)/2 then c(x) = 0;
8          else if n_1 + d_H(f_1, f_2) − k = d_H(f_1, f_2)/2 then c(x) = 1;
9          else c(x) =random bit;
10     k++; if c(x) = 1 then n_1++;
```

Output: Offspring c

nonlinearity. The conditions in the lines 7 and 8 are used to force the offspring to be balanced and the condition in the line 9 can be seen as the random mutation incorporated.

The results obtained by Millan in his GA are BFs with $N_8(f) = \mathbf{112}$ for 8 variables and $N_{10}(f) = \mathbf{476}$ for 10 variables when no local search is employed and when the SHC method is employed to improve the offspring the results obtained are BFs with $N_8(f) = \mathbf{116}$ for 8 variables and $N_{10}(f) = \mathbf{484}$ for 10 variables. It can be seen that the hybridization between the GA and SHC performs much better than the pure GA.

Evolution Strategy

Picek [39] implemented an $(\mu + \lambda)$-ES algorithm. In each generation parents compete with the offspring and from their joint set, the μ fittest individuals are kept. There is not information about the use of some mutation operator. The best results obtained by Picek with ES are BFs with nonlinearity $N_8(f) = \mathbf{112}$ for 8 variables (no results were presented for 10 variables) with $\lambda = 10$, $\mu = 500$ and $N_n(f)$ as the fitness function.

Genetic Programming

Picek [39] use a tree-based GP in which a BF is represented by a tree of nodes. BFs are represented in XOR and AND operators. GP uses tournament selection with tournament size 3. The crossover is performed with five different tree-based crossover operators selected at random: a simple tree crossover with 90% bias for functional nodes, uniform crossover, size fair, one-point and context preserving crossover. The best results obtained by Picek with GP are BFs with nonlinearity $N_8(f) = \mathbf{114}$ for 8 variables (no results were presented for 10 variables), with tree depth size equal to 7, population size equal to 10000 and $N_n(f)$ as the fitness function.

Cartesian Genetic Programming

Picek [39] uses a Cartesian Genetic Programming (CGP) which represents the BFs as a directed graph. Setting the number of rows to be one and levels-back parameter to be equal to the number of columns is regarded as the best and most general choice. CGP usually uses small population sizes and has no crossover operator. Determining the best combination of maximum number of nodes and mutation rate is crucial for a well performance of CGP. The number of input connections n_i for each node is set equal to 2 and the number of program output connections n_o is set equal to 1. The mutation operator is one-point where the mutation point is chosen with a fixed probability. The best results obtained by Picek with CGP are BFs with nonlinearity $N_8(f) = \mathbf{116}$ for 8 variables (no results were presented for 10 variables), with genotype size equal to 1500, mutation rate equal to 13% and population size equal to 5. Additionally CGP uses a $(1 + 4)$-ES individual selection in which offspring are favored over parents when they have a fitness better than or equal to the fitness of the parent. $N_n(f)$ is used as the fitness function.

Clonal Selection Algorithm

The Clonal Selection Algorithm (CLONALG), which belongs to the field of Artificial Immune Systems, is inspired by the clonal selection theory of acquired immunity that explains how B and T lymphocytes improve their response to antigens over time called affinity maturation.

Picek [38] implemented CLONALG method along with $N_n(f)$ as the fitness function, however, additional results obtained are not promising. The results obtained by Picek with CLONALG are BFs with $N_8(f) = \mathbf{114}$ for 8 variables and $N_{10}(f) = \mathbf{476}$ for 10 variables, with a population size of 50, the number of the clones is defined by

$$N_c = \sum_{i=1}^{n} \left\lceil \frac{\beta \cdot n}{i} \right\rceil, \tag{2.45}$$

where β is a parameter set equal to 6. The mutation intensity value M is calculated by

$$M = (1 + l \cdot \rho\,(1 - \exp(-r/\tau))), \tag{2.46}$$

where l is the bit string length, r is the rank of the clone to be mutated in the current population, $\rho = 0.2$ is the mutation parameter and no information is given about the τ variable.

Estimation of Distribution Algorithm

The main idea of estimation of distribution algorithms (EDAs) is to extract patterns shared by the best solutions, represent these patterns using a probabilistic model and generate new solutions from this model. In contrast to GAs, EDAs apply learning and sampling of distributions instead of crossover and mutation operators.

Picek [37] uses three types of probabilistic graphical models: univariate model, 1-order Markov model and tree model. The research was focused on analyzing the influence of three factors: the probabilistic model type, the truncation selection factor and the population size. Picek realizes that the probabilistic model was not a decisive factor in the success of the algorithm, however the probabilistic model that gives the slightly best results is the tree model. In brief, Picek was able to find BFs with $N_8(f) = \mathbf{116}$ for 8 variables and $N_{10}(f) = \mathbf{482}$ for 10 variables, with a population size equal to 500 and a truncation value equal to 0.4.

Particle Swarm Optimization

Another population-based metaheuristic found in the literature review was the work by Mariot [26], in which the particle swarm optimization (PSO) algorithm is employed. The main idea in this job is to apply Kennedy and Eberhart's discrete PSO to explore the space of balanced boolean functions. The PSO algorithm is coupled with the SHC 9 method to locally improve the nonlinearity of the particles.

The positions of the particles are 2^n-bit vectors representing the truth tables of BFs of n variables. Mariot designed a *swap-based* operator in such a way that a bit flips its value by swapping with another different bit value. The velocity vector of a particle is in turn updated using the classical PSO velocity equation, which is normalized through the *logistic function* to get meaningful probability values to perform a swap. Mariot tested the PSO algorithm with a coupled fitness function between the nonlinearity and the auto correlation 2.33.

$$fit(f) = N_n(f) + AC(f). \tag{2.47}$$

In order to improve the performance of the PSO algorithm after the application of the position update operator each particle is optimized with

Algorithm 13: Hill Climbing for Bent BFs - Related Work

Input : Bent BF f, MAXCHANGES, $N_{n_{min}}$

```
1  Set f' = f;
2  Define T_i = {x : f(x) = i, x ∈ F_2^n}      i ∈ {0,1};
3  for i = 0; i < MAXCHANGES; i++ do
       /* Check the Hamming weight (2.13) of f                    */
4      if w_H(f) < 2^{n-1} then
5          Select x_1, x_2 ∈ T_0;
6      else if w_H(f) > 2^{n-1} then
7          Select x_1, x_2 ∈ T_1;
8      else
9          Break;
10     f'(x_1) = f'(x_1) ⊕ 1;
11     f'(x_2) = f'(x_2) ⊕ 1;
12     if N_n(f') < N_{n_min} then
13         f' = f;
14     else
15         i = 0;
```

Output: Balanced BF f'

the SHC 9 method. The results obtained by Mariot are BFs with $N_8(f) = \mathbf{116}$ for 8 variables and $N_{10}(f) = \mathbf{480}$ for 10 variables, with 200 particles and 400 iterations for the algorithm.

2.3.3 Hybrid Methods with Algebraic Components

There also exist hybridizations between algebraic methods and metaheuristics, these are simpler methods that work with bent BFs and by a hill climbing procedure the BFs are balanced trying to lose the least possible nonlinearity.

Hill Climbing for Bent BFs

Burnett and Millan [5] implemented a HC method that randomly adjust a bent BF until convert it in a balanced BF. Algorithm 13 shows how this method works.

The HC method implemented by Burnett and Millan, receives a bent boolean function f, the maximum number of distinct 2-bit changes to be considered for each bent function *MAXCHANGES* and the minimum nonlinearity value which is acceptable for the resulting BF $N_{n_{min}}$.

The bent BFs for the algorithm 13 are based on the construction by Maiorana-McFarland [28]. As bent BFs are not weight balanced and have a Hamming weight of $2^{n-1} \pm 2^{n/2-1}$ the purpose of this method is to achieve balance while retaining high nonlinearity. The results that they obtain are the best known for metaheuristic methods, since they are able to find BFs with $N_8(f) = \mathbf{116}$ and BFs with $N_{10}(f) = \mathbf{488}$ for 8 and 10 variables respectively.

Another method very similar to the Burnett method is the concerning to the work of Izbenko [19]. Izbenko applies a modified SHC 9 procedure to a bent BF, in this way Izbenko was able to find BFs with $N_8(f) = \mathbf{116}$ for 8 variables and $N_{10}(f) = \mathbf{488}$ for 10 variables.

2.3.4 Summary of the Previous Work

Table 2.3 summarize the best results known up to the date for the previous work in metaheuristics reported. Additionally it is showed the best results known, which are obtained by algebraic constructions and the nonlinearity theoretical bound (NTB) for BFs with 8 and 10 variables.

TABLE 2.3: Comparison between the best results obtained for metaheuristics up to date

Metaheuristic	Method	f_{ob}	$N_8(f)$	$N_{10}(f)$
Trajectory-based	SHC [33]	$N_n(f)$	112	474
	SA-SHC [13]	C_{XR} & $N_n(f)$	116	486
	SA-SHC [21]	C_{RS} & $N_n(f)$	116	486
	SA-HC [29]	$AI(f)$+$FAR(f)$	116	488
Population-based	GA [32]	$N_n(f)$	112	476
	GA-SHC [32]	$N_n(f)$	116	484
	ES [39]	$N_n(f)$	112	-
	GP [39]	$N_n(f)$	114	-
	CGP [39]	$N_n(f)$	116	-
	CLONALG [38]	$N_n(f)$	114	476
	EDA [37]	$N_n(f)$	116	482
	PSO [26]	$N_n(f)$+$AC(f)$	116	480
Hybrid	HC-Bent [5]	$N_n(f)$	116	488
	SHC-Bent [19]	$N_n(f)$	116	488
Algebraic Constructions [18, 15]			116	**492**
Nonlinearity Theoretical Bound (NTB) [44]			**118**	**494**

We can see that when bent boolean functions are used or when the objective function employed to guide the search is different to the nonlinearity (N_n), better results are obtained. However, none of the metaheuristics presented achieves the results obtained for algebraic constructions for $n = 10$. Needless to say about achieving the NTB for $n = 8, 10$, which stills unknown.

Chapter 3

Proposals and Methods

This chapter is devoted to present the main algorithm proposals designed for this research. Firstly, the common concepts concerning to the proposals are described: the nonlinearity computation and the representation of solutions. Then, the new cost functions, trajectory methods, population methods and hybrid with algebraic procedures are introduced with a detailed description of them.

3.1 Components Applied in all Proposals

3.1.1 Representation of Solutions

In this thesis each solution $\mathcal{S}$ is encoded using the binary representation composed by a vector with 2^n boolean decision variables $\mathcal{S}[k]$, where n is the number of variables of the BF f and $k \in \mathcal{N}_n$. In order to introduce the relationship between the solution $\mathcal{S}$ and f, let us introduce the analogous function to *to_int* 2.6 that maps an integer k to a binary vector $\mathbf{x} \in \mathbb{F}_2^n$, thus, $to\_bin : \mathbb{N} \to \mathbb{F}_2^n$, such that

$$\mathbf{x} = (x_1, \cdots, x_n) = to\_bin(k) = \left(\left\lfloor \frac{k}{2^{n-1}} \right\rfloor \%2, \cdots, \left\lfloor \frac{k}{2^{n-n}} \right\rfloor \%2 \right). \tag{3.1}$$

Now, we are ready to introduce the relationship between the solution $\mathcal{S}$ and f, each decision variable $\mathcal{S}[k]$ is set equal to the corresponding truth table value $f(\mathbf{x})$, where $k = to\_int(\mathbf{x}), \mathbf{x} \in \mathbb{F}_2^n$ (or $\mathbf{x} = to\_bin(k), k \in \mathcal{N}_n$)

$$f(\mathbf{x}) = \mathcal{S}[k]. \tag{3.2}$$

Since, we only work with balanced BFs, 2^{n-1} decision variables will be set equal to 0 and 2^{n-1} decision variables will be set equal to 1. Let us denote the positions sets $\mathcal{P}_0$ and $\mathcal{P}_1$ such that

$$\begin{aligned} \mathcal{S}[p_0] &= 0 \qquad \forall\ p_0 \in \mathcal{P}_0, \\ \mathcal{S}[p_1] &= 1 \qquad \forall\ p_1 \in \mathcal{P}_1. \end{aligned}$$

Thus, the sets $\mathcal{P}_0$ and $\mathcal{P}_1$ have the following properties:

$$|\mathcal{P}_0| = |\mathcal{P}_1| = 2^{n-1}, \quad \mathcal{P}_0 \cap \mathcal{P}_1 = \emptyset, \quad \mathcal{P}_0 \cup \mathcal{P}_1 = \mathcal{N}_n.$$

Each solution has associated its $\mathcal{P}_0$, $\mathcal{P}_1$ sets and an integer vector $\mathcal{W}$ of length 2^n which denotes the WHT of the BF f, analogously to the equality 3.2 we have that the following holds for a vector $\mathcal{W}$ denoting the WHT of the BF f

$$W_n(f)(\mathbf{w}) = \mathcal{W}[k] \quad \text{s.t} \quad k \in \mathcal{N}_n, \quad \mathbf{w} = to\_bin(k), \tag{3.3}$$

where $\mathcal{W}[k]$ denotes an integer value and $W_n(f)(\mathbf{w})$ the WHT value associated to the linear function $L_{\mathbf{w}}$, which is also an integer.

As an example of the representation, consider the following 4-variable balanced BF f in truth table form

$$f = \boxed{1\ 0\ 1\ 0\ 1\ 0\ 1\ 0\ 1\ 0\ 1\ 0\ 1\ 0\ 1\ 0} \tag{3.4}$$

The solution $\mathcal{S}$ and its attributes, the $\mathcal{P}_i$ sets and the $\mathcal{W}$ vector are represented as

$$\begin{aligned}
\mathcal{S} &= \boxed{1}\boxed{0}\boxed{1}\boxed{0}\boxed{1}\boxed{0}\boxed{1}\boxed{0}\boxed{1}\boxed{0}\boxed{1}\boxed{0}\boxed{1}\boxed{0}\boxed{1}\boxed{0} \\
\mathcal{S}.\mathcal{P}_0 &= \{\ 1\ \ 3\ \ 5\ \ 7\ \ 9\ \ 11\ \ 13\ \ 15\ \} \\
\mathcal{S}.\mathcal{P}_1 &= \{\ 0\ \ 2\ \ 4\ \ 6\ \ 8\ \ 10\ \ 12\ \ 14\ \} \\
\mathcal{S}.\mathcal{W} &= \boxed{0}\boxed{-16}\boxed{0}\boxed{0}\boxed{0}\boxed{0}\boxed{0}\boxed{0}\boxed{0}\boxed{0}\boxed{0}\boxed{0}\boxed{0}\boxed{0}\boxed{0}\boxed{0}
\end{aligned} \tag{3.5}$$

We can associate to each solution its nonlinearity too, in this case, we have that

$$\mathcal{S}.N_4 = \frac{1}{2}(2^4 - 16) = 0.$$

As the balanced BF f 3.4 is affine, its nonlinearity is equal to 0.

Neighborhood.

In order to introduce the neighborhood employed in this thesis it is necessary to define an operator that transforms a solution $\mathcal{S}$ into another one $\mathcal{S}'$. We call this operator *Bit Swapping* (BS), it receives a solution $\mathcal{S}$ and two positions p_0 and p_1 as inputs and returns another solution $\mathcal{S}'$, such that the decision variables $\mathcal{S}[p_0]$ and $\mathcal{S}[p_1]$ are exchanged (or complemented) and $d_H(\mathcal{S}, \mathcal{S}') = 2$, *i.e.*

$$\mathcal{S}'[k] = \begin{cases} 1 \oplus \mathcal{S}[k] & \text{if } k = p_0, p_1 \\ \mathcal{S}[k] & \text{if } k \in \mathcal{N}_n \setminus \{p_0, p_1\}. \end{cases} \tag{3.6}$$

The Bit Swapping operator is defined as follows

$$BS(\mathcal{S}, p_0, p_1) := swap(\mathcal{S}[p_0], \mathcal{S}[p_1]). \tag{3.7}$$

Algorithm 14: Random Construction of a Balanced BF

input : $\mathcal{S}$ (vector with length 2^n)

1 *Set* $\mathcal{S}[k] = 0 \quad \forall k \in \mathcal{N}_n$;
2 *Set* $\mathcal{S}.\mathcal{P}_0 = \mathcal{N}_n$; /* $\mathcal{S}.\mathcal{P}_0$ starts with size 2^n */
3 *Set* $\mathcal{S}.\mathcal{P}_1 = \emptyset$;
4 **for** $j = 0; j < 2^{n-1}; j++$ **do**
5 *Select some* $k \in \mathcal{S}.\mathcal{P}_0$;
6 $\mathcal{S}[k] = 1$;
7 $\mathcal{S}.\mathcal{P}_1 = \mathcal{S}.\mathcal{P}_1 \cup \{k\}$;
8 $\mathcal{S}.\mathcal{P}_0 = \mathcal{S}.\mathcal{P}_0 \setminus \{k\}$; /* $\mathcal{S}.\mathcal{P}_0$ finishes with size 2^{n-1} */

output: Solution $\mathcal{S}$

As an example, consider the solution $\mathcal{S}$ given previously in 3.5 and the positions $p_0 = 3 \in \mathcal{S}.\mathcal{P}_0$ and $p_1 = 8 \in \mathcal{S}.\mathcal{P}_1$, we have that the new neighbor solution is

$$\mathcal{S}' = BS(\mathcal{S}, p_0 = 3, p_1 = 8)$$

$$\mathcal{S}' = \boxed{1 \mid 0 \mid 1 \mid 1 \mid 1 \mid 0 \mid 1 \mid 0 \mid 0 \mid 0 \mid 1 \mid 0 \mid 1 \mid 0 \mid 1 \mid 0} \tag{3.8}$$

$$\mathcal{S}'.\mathcal{P}_0 = \{\ 1\ \ 8\ \ 5\ \ 7\ \ 9\ \ 11\ \ 13\ \ 15\ \}$$

$$\mathcal{S}'.\mathcal{P}_1 = \{\ 0\ \ 2\ \ 4\ \ 6\ \ 3\ \ 10\ \ 12\ \ 14\ \}$$

$$\mathcal{S}'.\mathcal{W} = \boxed{0 \mid -12 \mid 4 \mid 0 \mid 0 \mid 4 \mid 4 \mid 0 \mid -4 \mid 0 \mid 0 \mid -4 \mid -4 \mid 0 \mid 0 \mid -4}$$

$$\mathcal{S}'.N_4 = \frac{1}{2}(2^4 - 12) = 2$$

It is easily to see that solutions $\mathcal{S}$ and $\mathcal{S}'$ only differ in two positions: the third and the eighth. However, $\mathcal{S}'$ has a better (higher) nonlinearity than $\mathcal{S}$.

Employing BS we can construct the full neighborhood for a single solution $\mathcal{S}$ as

$$N(\mathcal{S}) = \{BS(\mathcal{S}, p_0, p_1) : p_0 \in \mathcal{S}.\mathcal{P}_0 \wedge p_1 \in \mathcal{S}.\mathcal{P}_1\}. \tag{3.9}$$

The total amount of bit-swapping that can be performed is the neighborhood size

$$|N(\mathcal{S})| = |\mathcal{S}.\mathcal{P}_0| \times |\mathcal{S}.\mathcal{P}_1| = 2^{n-1} \times 2^{n-1} = 2^{2n-2}. \tag{3.10}$$

3.1.2 BFs Random Generator

The trajectory-based and population-based metaheuristics make use of the balanced BF random generator. The algorithm 14 describes how to generate a balanced BF at random. It generates a set $\mathcal{P}_0 = \mathcal{N}_n$ with 2^n elements and all the decision variables are set equal to 0, then 2^{n-1} different elements are extracted at random from $\mathcal{P}_0$, generating another set $\mathcal{P}_1$ and the decision variables associated to the elements in $\mathcal{P}_1$ are set equal to 1. In this way is ensured that the BF generated is balanced.

Algorithm 15: Computation of the FWHT

input : Solution $\mathcal{S}$

1 *Set* $W_0[k] = 1 - 2\mathcal{S}[k] \quad \forall k \in \mathcal{N}_n$; /* Zero phase */

/* Cycle to go through the different phases */

2 **for** $i = 0; \quad i < n; \quad i++$ **do**

3 $p = 2^i$; /* To identify which cells must sum */

4 $q = 2^{n-(i+1)}$; /* Gives the number of distinct sums (*blocks*) */

/* Cycles to fill the phase $i+1$ */

5 **for** $j = 0; \quad j < q; \quad j++$ **do**

6 **for** $k = 2jp; \quad k < 2jp + p; \quad k++$ **do**

/* Butterfly diagram of size two */

7 $W_{i+1}[k] = W_i[k] + W_i[k+p]$;

8 $W_{i+1}[k+p] = W_i[k] - W_i[k+p]$;

9 *Set* $\mathcal{S}.\mathcal{W} = W_n$;

output: Walsh Hadamard Transform $\mathcal{S}.\mathcal{W}$

3.1.3 Fast Walsh Hadamard Transform Implementation

As we previously discussed, the direct calculus of the WHT requires N^2 operations, with $N = 2^n$. The FWHT allows to compute the WHT in $N \log N$ operations, *i.e.* in only $n2^n$ (instead of 2^{2n}) operations. All the cost functions are based on using the WHT values, so calculating it in an efficient way is really important to be able to run several iterations of our optimizers. The algorithm 15 describes how to compute the FWHT using the *butterfly diagram* of size two (see subsection 2.2.2).

To properly understand the number of steps executed by algorithm 15, it is clear that n phases are carried out. In each phase, it performs pq operations and for every p it makes 2 operations, so the total number of operations (T_{op}) is

$$T_{op} = npq2 = n2^i 2^{n-(i+1)} 2 = n2^n.$$

Walsh Hadamard Transform Update

Several of the proposals are based on performing bit-swapping in the encoding, i.e. a position with value 0 is modified to contain a 1, and a position with value 1 is modified to contain a 0. Calculating again the entire WHT would be costly, so an effort to properly update the WHT efficiently was performed. By storing the set of phases $W_0, ..., W_n$ in the individual representation, we can define another algorithm to update the WHT (*WHTU*) faster when a bit-swapping of this kind is performed. The main principle is to update only the information that is altered because of the

Algorithm 16: Walsh Hadamard Transform Update (WHTU)

input : Solution $\mathcal{S}$ and positions p_0, p_1 to update

`/* Update` $\mathcal{S}$`, by bit-swapping` $\mathcal{S}[p_0]$ `with` $\mathcal{S}[p_1]$ `*/`

1 $BS(\mathcal{S}, p_0, p_1)$;

2 *Set* $W_0[k] = 1 - 2 \times \mathcal{S}[k] \quad \forall k \in \mathcal{N}_n$; `/* Zero phase */`

`/* Cycle to go through the different phases */`

3 **for** $i = 1; \quad i \leq n; \quad i++$ **do**

`/* Identify the blocks for any position at each phase */`

4 $b_0 = p_0/2^i$;

5 $b_1 = p_1/2^i$;

6 $p = 2^{i-1}$; `/* To identify which cells must sum */`

7 **for** $k = 2b_0p; \quad k < 2b_0p + p; \quad k++$ **do** `/*` b_0 `block update */`

8 $W_i[k] \quad = W_{i-1}[k] + W_{i-1}[k+p]$;

9 $W_i[k+p] = W_{i-1}[k] - W_{i-1}[k+p]$;

10 **if** $b_0 \neq b_1$ **then**

11 **for** $k = 2b_1p; \quad k < 2b_1p + p; \quad k++$ **do** `/*` b_1 `block update */`

12 $W_i[k] \quad = W_{i-1}[k] + W_{i-1}[k+p]$;

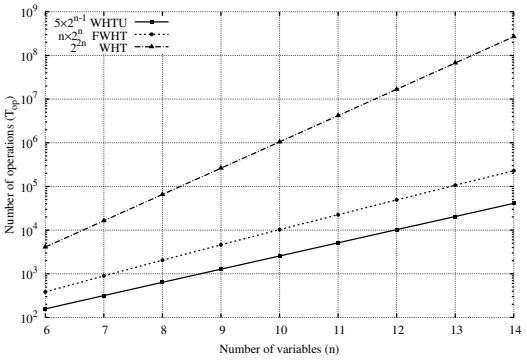
13 $W_i[k+p] = W_{i-1}[k] - W_{i-1}[k+p]$;

14 *Set* $\mathcal{S}.\mathcal{W} = W_n$;

output: Walsh Hadamard Transform Updated $\mathcal{S}.\mathcal{W}$

modification of these two positions. The algorithm 16 explains how to update the WHT when a bit-swapping is performed.

FIGURE 3.1: Comparison between the number of operations performed by the three different methods to obtain the WHT.



This algorithm allows to update the WHT in approximately $5 \times 2^{n-1}$ operations on average. Additionally, the total number of operations that the algorithm 16 performs is bounded by 2×2^n (when the condition in the line 10 is never fulfilled) and by 3×2^n (when the condition in the line 10 is fulfilled at most in $n-1$ phases) *i.e.* we have

$$2^{n+1} < 5 \times 2^{n-1} < 3 \times 2^n. \tag{3.11}$$

The number of operations required to compute WHT with different algorithms are shown in figure 3.1 for n ranging between 6 and 14. Three different algorithms are considered, the trivial WHT, the fast WHT and the WHT update . Obviously, this last one can only be used when a single bit-swapping is performed. As we can see in figure 3.1 the FWHT reduces the number of operations in several orders of magnitude with respect to the direct computation, and the WHTU reduce the number of operations in an additional half order, so in order to evolve several iterations of the optimizers, its application is important.

3.2 Proposals

3.2.1 Cost Functions

In order to properly measure the performance of the algorithms developed in this thesis, it is important to remark that our aim is to generate balanced BFs with high nonlinearity, i.e. our objective function is the nonlinearity 2.31

$$\mathbf{F_1}(\mathcal{S}) = \frac{1}{2}\left(2^n - \max_{k\in\mathcal{N}_n} |\mathcal{S}.\mathcal{W}[k]|\right). \tag{3.12}$$

We can analyze some properties of this function by taking into account that the values in the WHT $\mathcal{S}.\mathcal{W}$ are constrained by the Parseval's equation 2.19. We know that for a balanced BF all the values in its WHT are multiples of 4. This has as a consequence that the amount of different values that could take $\max_{k\in\mathcal{N}_n} |\mathcal{S}.\mathcal{W}[k]|$ is really small so the different values of $\mathbf{F_1}$ is also small. This relation between $\max_{k\in\mathcal{N}_n} |\mathcal{S}.\mathcal{W}[k]|$ and $\mathbf{F_1}$ is more clear if we transform the original objective function 3.12 (maximize the nonlinearity) into another one (minimize the maximum value in the WHT):

$$\mathbf{F_2}(\mathcal{S}) = \max_{k\in\mathcal{N}_n} |\mathcal{S}.\mathcal{W}[k]|. \tag{3.13}$$

As other researchers [12], we realized that using the objective function as the fitness function is not suitable to guide the search in EAs and it is necessary to define a new fitness function (in our case cost function). One of the drawbacks seen easily from the objective function 3.12 (or 3.13) is that it just takes a small amount of different values, so a lot of BFs are considered to be of equal value. However, even if they share the same nonlinearity, one of them could be used to easily improve the nonlinearity, by just making some modifications, while another ones could be far from better solutions.

Previous Cost Function

In order to tackle the problem of generating CBFs with high nonlinearity, the most usual alternative is to use the objective function $\mathbf{F_1}$ 3.12 as the fitness

function or the objective function $\mathbf{F_2}$ 3.13 as the cost function to guide the search. Many researchers use only $\mathbf{F_1}$ 3.12 (or $\mathbf{F_2}$ 3.13), in some cases is used together with an extra penalization to respect the balance of the BF. Since we work only with balanced boolean functions these kinds of penalizations are not required in our method so we discard them in our analyses.

The most popular cost function used with metaheuristics is the objective function $\mathbf{F_2}$ 3.13, that will be denoted as C_1 in the following.

$$C_1(\mathcal{S}) = \mathbf{F_2}(\mathcal{S}) = \max_{k \in \mathcal{N}_n} |\mathcal{S}.\mathcal{W}[k]|. \tag{3.14}$$

Also, the cost function designed by Clark [12] is widely used,

$$C_2(\mathcal{S}) = \sum_{k=0}^{2^n-1} ||\mathcal{S}.\mathcal{W}[k]| - \mathcal{X}|^R. \tag{3.15}$$

When using C_2 3.15 every value in the WHT influences the cost function of S, rather than just the maximum one as in the cost function C_1 3.14. The authors note "the parameters $\mathcal{X}$ and R provide freedom to experiment". Similarly to the authors, we use $R = 3$ and for the $\mathcal{X}$, the expected maximum value for the WHT when achieves the nonlinearity bound. This value is calculated using equations 2.31 and 2.40 by equalizing them and solving for $\mathcal{X}$

$$\begin{aligned} 2^{n-1} - 2^{-1}\mathcal{X} &= 2^{n-1} - 2^{n/2-1} - 2 \\ \mathcal{X} &= 2^{n/2} + 2^2. \end{aligned} \tag{3.16}$$

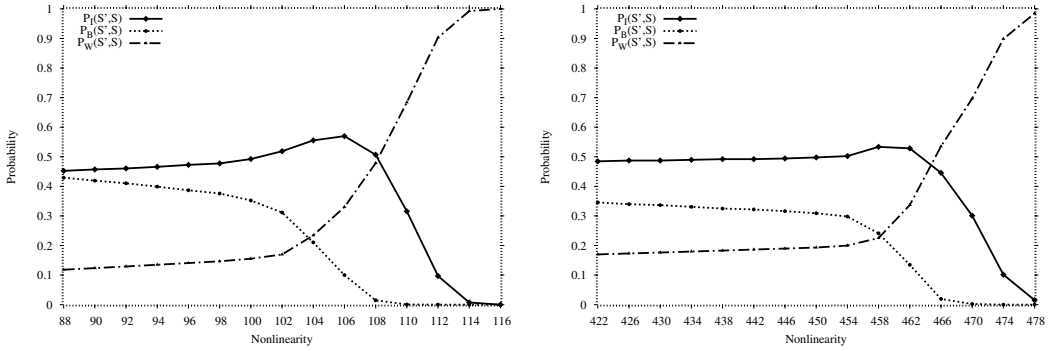
This is the value that we use for $\mathcal{X}$ in our experiments in order to compare the performance of the cost functions.

Motivation for New Cost Function

The main drawback for C_1 3.14 has been mentioned above and is based on the fact that too few different values are generated. The cost function C_2 3.15 might solve this issue. However, a solution $\mathcal{S}_1$ with a higher nonlinearity than a solution $\mathcal{S}_2$ might have a higher (worse) cost function than $\mathcal{S}_2$ which is a not desirable property. Moreover, despite the large amount of research on the application of EAs to the design of CBFs with high nonlinearity, their results are not so good as the ones obtained with algebraic constructions (see table 2.3). Thus, we consider that there is enough motivation for designing a new cost function that avoids the drawbacks of the previous ones. Particularly, we hypothesize that one of the issues that causes the poor performance of currently designed metaheuristics is due the definition of the cost function C_1 3.14 — the most commonly applied cost function — which has the drawback that it might contain huge plateaus because it returns integer values in a small range, so many BFs are mapped to the same value. To illustrate this we can explore the full neighborhood $N(\mathcal{S})$ of

a solution $\mathcal{S}$ and compute the probability that a neighbor $\mathcal{S}'$ has identical cost value $P_I(\mathcal{S}', \mathcal{S})$, better cost value (lower) $P_B(\mathcal{S}', \mathcal{S})$ or worse cost value (higher) $P_W(\mathcal{S}', \mathcal{S})$ than $\mathcal{S}$ when the cost function C_1 3.14 is employed to guide the search. The figure 3.2 shows the average of this probability values when 50 different solutions with the same nonlinearity are generated at random for 15 different nonlinearity values ranging between 88-116 and 422-478 for 8 and 10 variables respectively.

FIGURE 3.2: Probability that a neighbor $\mathcal{S}'$ has identical cost value $P_I(\mathcal{S}', \mathcal{S})$, better cost value $P_B(\mathcal{S}', \mathcal{S})$ or worse cost value $P_W(\mathcal{S}', \mathcal{S})$ than $\mathcal{S}$ when the cost function C_1 3.14 is employed to guide the search



(A) 8-variables (B) 10-variables

As we can see, the probability to have the same cost value when applying C_1 3.14 is large. In fact, in about half of the neighborhood, the cost value does not vary. For the best solutions, the probability to have the same cost value diminishes. However, this is because the probability to have a worse cost value (higher) increases.

New Cost Functions

One of the contributions of this thesis is the definition of new cost functions. We propose two new cost functions in order to address the problem properly. In order to explain our cost function, lets denote the absolute values in the WHT as $X_1, X_2, \cdots, X_\alpha$, such that

$$X_1 = 0 < X_2 < \cdots < X_\alpha = \max_{k \in \mathcal{N}_n} |\mathcal{S}.\mathcal{W}[k]|.$$

In addition, η_i is used to denote the amount of times that the X_i value appears in the WHT. Note that $X_1 = 0$, since $\mathcal{S}.\mathcal{W}[0] = 0$ (according to equation 2.28) and $\eta_1 > 0$.

Based on the above, we propose two new cost functions. In both we form a tuple of values

$$C_3(\mathcal{S}) = (\mathbf{F_2}(\mathcal{S}), \boldsymbol{\xi_1}(\mathcal{S})), \tag{3.17}$$
$$C_4(\mathcal{S}) = (\mathbf{F_2}(\mathcal{S}), \boldsymbol{\xi_2}(\mathcal{S})), \tag{3.18}$$

where the first one is the objective function $\mathbf{F}_2$ 3.13 and the second values $\boldsymbol{\xi_1}$, $\boldsymbol{\xi_2}$ — which are the novelty — help us to discern between solutions with the same nonlinearity. The second members ξ_1, ξ_2 of such cost functions are the following

1. In the second value of the cost function 3.17 we try to minimize the maximum absolute value X_α and the second maximum absolute value $X_{\alpha-1}$ in the WHT. In order to attain this aim, the second value of the cost function takes also into account the number of appearances of these values η_α and $\eta_{\alpha-1}$ in the WHT. This is implemented as follows:

$$\xi_1(\mathcal{S}) = (\eta_\alpha \times X_\alpha)^3 + (\eta_{\alpha-1} \times X_{\alpha-1}). \tag{3.19}$$

2. Let $\pm X_k$ belong to the WHT such that $X_{k-1} \leq \mathcal{X} < X_k$. In the second value of the cost function 3.18 we try to minimize the appearance of entries with absolute values greater than $\mathcal{X}$ 3.16. The penalty applied to the appearances is higher for the larger values.

$$\xi_2(\mathcal{S}) = \sum_{i=k}^{\alpha} (2 \times \eta_i \times X_i)^{i-k+1}. \tag{3.20}$$

Employing the cost functions previously defined, we define a new operator $\prec$ in such way to identify when a solution $\mathcal{S}'$ is better than the solution $\mathcal{S}$ ($\mathcal{S}' \prec \mathcal{S}$) and the operator $\preceq$ in which the solution $\mathcal{S}'$ is better or equal than the solution $\mathcal{S}$ ($\mathcal{S}' \preceq \mathcal{S}$). For example, for C_3 3.17, we have

$$\mathcal{S}' \prec \mathcal{S} : \mathbf{F_2}(\mathcal{S}') < \mathbf{F_2}(\mathcal{S}) \vee \left(\mathbf{F_2}(\mathcal{S}') = \mathbf{F_2}(\mathcal{S}) \wedge \boldsymbol{\xi_1}(\mathcal{S}') < \boldsymbol{\xi_1}(\mathcal{S})\right), \tag{3.21}$$
$$\mathcal{S}' \preceq \mathcal{S} : \mathbf{F_2}(\mathcal{S}') < \mathbf{F_2}(\mathcal{S}) \vee \left(\mathbf{F_2}(\mathcal{S}') = \mathbf{F_2}(\mathcal{S}) \wedge \boldsymbol{\xi_1}(\mathcal{S}') \leq \boldsymbol{\xi_1}(\mathcal{S})\right). \tag{3.22}$$

3.2.2 Trajectory-based Metaheuristics

Full Hill Climbing

This one is based on the general HC algorithm 2. Algorithm 17 illustrates how the FHC method works. This algorithm makes use of the neighborhood 3.9 and stops when a local optimum is reached.

The FHC algorithm 17 ensures that all the neighbors of a solution $\mathcal{S}$ are visited just once, this algorithm works as follows: first one decision variable set equal to 0 is selected at random (line 5) and is bit swapped with all the

Algorithm 17: Full Hill Climbing (FHC)

Input : Initial solution $\mathcal{S}$, T_{end}

```
1  T_elapsed = 0 ;
2  LOF= False ;                                   /* Local Optimum Found  */
3  while LOF= False do
       /* Check all the neighbors S' by fixing p_0 at random and swap
          with the whole set P_1 randomly                                  */
4      BSF= False ;                               /* Best Solution Found  */
5      foreach p_0 ∈ S.P_0 ∧ BSF= False do
6          foreach p_1 ∈ S.P_1 ∧ BSF= False do
7              S' = BS(S, p_0, p_1) ;      /* Select the current S' ∈ N(S) */
8              if S' ≺ S then
9                  S = S' ;
10                 BSF= True;
11             Update(T_elapsed) ;
12             if T_elapsed > T_end then
13                 BSF= True ;
14                 LOF= True ;
       /* If all the neighbor solutions S' are visited and BSF= False
          the solution S is a local optimum                                */
15     if BSF= False then
16         LOF= True ;
```

Output: *Local Optimum* $\mathcal{S}$

decision variables set equal to 1 (line 6), then if the neighbor generated is better than the current solution, this is selected to replace the current solution $\mathcal{S}$ (line 9) and both for cycles stops and start again. If no better neighbor is found and the neighborhood is fully visited, the solution is considered as local optimum and the algorithm stops (line 16). Another stopping criterion that can be used is time (line 12) returning the best solution found.

First Improvement Quasi-Tabu Search

Tabu Search is a very popular method that in each iteration moves to the non-tabu neighbor that attains the largest improvement. Since the neighborhood considered in this thesis is quite large, exploring the whole neighborhood is too costly. As a result, a novel method that applies some of the principles of Tabu Search was devised. However, this method does not explore the whole neighborhood. Instead, it moves to the first neighbor that improves the current solution. The algorithm 18 illustrates how the First Improvement Quasi-Tabu Search (FIQTS) works. First, the tabu bit-swapping positions $\mathcal{R}_0$ and $\mathcal{R}_1$ are initialized at random (line 2), $\mathcal{R}_0$ and $\mathcal{R}_1$ contain positions that

Algorithm 18: First Improvement Quasi-Tabu Search (FIQTS)

Input : Initial solution $\mathcal{S}$, T_{end}

1 $T_{elapsed} = 0$;

/* Tabu bit-swapping positions initialized randomly */

2 *Set* $\mathcal{R}_i = \{p_k : p_k \in \mathcal{S}.\mathcal{P}_i, k = 1, \cdots, 2^{n-4}\}, i \in \{0,1\}$;

/* Non-tabu bit-swapping positions */

3 $\mathcal{Q}_i = \mathcal{S}.\mathcal{P}_i \setminus \mathcal{R}_i, i \in \{0,1\}$;

/* Construct the reduced neighborhood */

4 $N_r(\mathcal{S}) = \{BS(S, q_0, q_1) : q_0 \in \mathcal{Q}_0, q_1 \in \mathcal{Q}_1\}$;

5 *Set* $N_v(\mathcal{S}) = \emptyset$; /* Neighbors already visited */

6 *Set* $Repeats = 0$; /* Repetitions of neighbors */

7 *LOF= False* ; /* Local Optimum Found */

8 **while** *LOF= False* **do**

/* Select a random neighbor $\mathcal{S}' \in N_r(\mathcal{S})$ */

9 $\mathcal{S}' = BS(\mathcal{S}, q_0, q_1), \quad q_0 \in \mathcal{Q}_0, q_1 \in \mathcal{Q}_1$;

/* Exchange the oldest element $r_{i_{old}} \in \mathcal{R}_i$ with $q_i \in \mathcal{Q}_i$ */

10 $swap(\mathcal{Q}_i[q_i], \mathcal{R}_i(r_{i_{old}})), \quad i \in \{0,1\}$;

/* Check if the neighbor has not been visited */

11 **if** $N_v(\mathcal{S}) \cap \{\mathcal{S}'\} = \emptyset$ **then**

12 $N_v(\mathcal{S}) = N_v(\mathcal{S}) \cup \{\mathcal{S}'\}$;

13 **if** $\mathcal{S}' \prec \mathcal{S}$ **then**

14 $\mathcal{S} = \mathcal{S}'$;

15 $N_v(\mathcal{S}) = \emptyset$;

16 $Repeats = 0$;

17 **else**

18 $Repeats++$;

/* Stopping criterion */

19 **if** $N_v(\mathcal{S}) = N(\mathcal{S})$ *or* $Repeats \geq 2|N_r(\mathcal{S})|$ **then**

20 $LOF = True$;

21 $Update(T_{elapsed})$;

22 **if** $T_{elapsed} > T_{end}$ **then**

23 $LOF = True$;

Output: *Best solution found* $\mathcal{S}$

are not allowed to bit-swap. Then the non-tabu bit-swapping positions $\mathcal{Q}_0$ and $\mathcal{Q}_1$ are initialized (line 3), such that

$$\mathcal{Q}_i \cup \mathcal{R}_i = \mathcal{S}.\mathcal{P}_i \qquad \mathcal{Q}_i \cap \mathcal{R}_i = \emptyset \quad i \in \{0,1\}.$$

The sets $\mathcal{Q}_0$ and $\mathcal{Q}_1$ contain positions such that the decision variables $\mathcal{S}[q_0]$, $\mathcal{S}[q_1]$ are set equal to 0, 1 respectively, and that might be bit-swapped. In base of these sets the reduced neighborhood $N_r(\mathcal{S})$ is constructed (line 4). Each time that a neighbor is created by bit-swapping the decision variables in the positions q_0 and q_1, such elements are moved to $\mathcal{R}_0$ and $\mathcal{R}_1$,

Algorithm 19: Iterated Local Search (ILS)

Input : $\mathcal{S}$, outer iterations *Out_iters*, percentage of perturbation p_p

1 *Set* $\rho = \lceil 2^{n-1} \times p_p \rceil$;
2 *Set* $Iters = 0$;
`/* Apply the FIQTS algorithm (alg 18) */`
3 $\mathcal{S}' = FIQTS(\mathcal{S})$;
4 $\mathcal{S} = \mathcal{S}'$; `/* Best solution found */`
5 **while** $Iters < Out\_iters$ **do**
6 $\quad \mathcal{S}'' = \mathcal{S}$;
$\quad$ `/* Perturb the obtained local optimum` $\mathcal{S}''$ `*/`
7 $\quad$ **for** $k = 0; k < \rho; k++$ **do**
8 $\quad\quad \mathcal{S}'' = BS(\mathcal{S}'', p_{0_k}, p_{1_k})$;
$\quad$ `/* Apply the FIQTS algorithm (alg 18) to` $\mathcal{S}''$ `*/`
9 $\quad \mathcal{S}' = FIQTS(\mathcal{S}'')$;
$\quad$ `/* Check if the new local optimum is better */`
10 $\quad$ **if** $\mathcal{S}' \prec \mathcal{S}$ **then**
11 $\quad\quad \mathcal{S} = \mathcal{S}'$;
12 $\quad Iter++$;

Output: *Best Solution Found* $\mathcal{S}$

respectively (lines 9, 10). Additionally, the elements that have been for a longer time in R_0 and R_1 are inserted in Q_0 and Q_1. The main principle is to promote the selection of different decision variables in each step when performing the bit-swapping. The stopping criterion is set in such a way that the iterative process ends when all the neighborhood has been visited and no better solution was found or when the random sampling process generates too many neighbors that have already been visited previously (line 19). Another stopping criterion that can be used is time (line 22).

Iterated Local Search

Iterated local search is basically the join of local search and perturbation, sometimes extended with additional memories. In this thesis, a simple ILS is used. First, it applies the FIQTS algorithm in order to find a relatively good solution in the region. Note that, since the whole neighborhood is not visited necessarily, this solution might not be a local optimum. Then, a perturbation based on performing a set of swaps is triggered and FIQTS is applied again. This is repeated until a stopping criterion is reached. This algorithm has one parameter to be fixed (the percentage of perturbation p_p), and it is described in algorithm 19.

The perturbation of the local optimum obtained at each step is carried out by simply bit-swapping 2ρ decision variables in the solution $\mathcal{S}$ (lines 7,8). This operation maintains the balancedness of the solutions because the same amount of elements from $\mathcal{S}.\mathcal{P}_0$ and $\mathcal{S}.\mathcal{P}_1$ are selected. The main difference

between ILS and the previous algorithms (FHC and FIQTS) is that the previous algorithms stop when a local optimum is reached. However, in ILS the process stops after a certain number of iterations and it might scape from local optima.

Simulated Annealing

Simulated Annealing (SA) is another trajectory-based search method that was designed with the aim of escaping from local optima. Its main feature with respect to the previous described methods is that it incorporates a strategy to accept worsening solutions with certain probability. It is a quite popular algorithm, and the main modification was the incorporation of the tabu method applied in FIQTS to reduce the neighborhood. Algorithm 20 illustrates the internal operation of this method.

As is known, the performance of the SA algorithm strongly depends on the choice of the cooling schedule and on the neighborhood structure. We use the classical version with a geometric cooling schedule following some ideas exposed by Ben-Ameur [4]. In order to set the initial temperature and the geometric value for α, we first calculate an estimate of the average Δ_E ($avg\Delta_E$) for uphill movements at the initial execution time and we found that this value $avg\Delta_E \approx 0.5$ (following the rules to calculate Δ_E in the lines 15 and 17 of the algorithm 20). In this way we set the initial temperature such that the acceptance probability of uphill movements is p_{u_0}, so we have

$$p_{u_0} = \exp\left(\frac{-avg\Delta_E}{Temp_0}\right), \tag{3.23}$$

solving the equation 3.23 for $Temp_0$, we obtain the value in the line 1

$$Temp_0 = \frac{-avg\Delta_E}{p_{u_0}}. \tag{3.24}$$

For the α value we have that after the 50% of the execution time, we have

$$Temp = \alpha^{0.5 \times Out\_iters} \times Temp_0, \tag{3.25}$$

and if we decide that after this execution time only the P_{u_1} percentage of the uphill movements will be accepted, then we have

$$p_{u_1} = \exp\left(\frac{-avg\Delta_E}{\alpha^{0.9 \times Out\_iters} \times Temp_0}\right). \tag{3.26}$$

Solving the equation 3.26 for α, we obtain the value in the line 2

$$\alpha = \left(\frac{-avg\Delta_E}{Temp_0 \times p_{u_1}}\right)^{\frac{1}{0.5Out\_iters}}. \tag{3.27}$$

Algorithm 20: Simulated Annealing (SA)

Input : $\mathcal{S}$, outer iterations *Out_iters*, cooling schedule ($avg\Delta_E$, p_{u_0}, p_{u_1})

```
1  Set Temp_0 = −avgΔ_E / p_{u_0} ;
2  Set α = [−avgΔ_E / (Temp_0 × p_{u_1})]^{1/0.5Out_iters};
3  Temp = Temp_0;
4  Set Iters_O = 0;
5  Set Inner_iters = 2^{2n−4};
6  Set S_best = S ;
7  while Iters_O < Out_iters do
8      Set N_v(S) = ∅;
9      Set Iters_I = 0;
10     while Iters_I < Inner_iters do
           /* Select a random neighbor S' ∈ N_r(S)                    */
11         S' = BS(S, q_0, q_1) ;
           /* Exchange the oldest element r_{i_old} ∈ R_i with q_i ∈ Q_i */
12         swap(S.Q_i[q_i], S.R_i(r_{i_old})) ;
           /* Check if the neighbor has not been visited              */
13         if N_v(S) ∩ {S'} = ∅ then
14             N_v(S) = N_v(S) ∪ {S'};
15             Δ_E = (Fob_2(S') − Fob_2(S)) ;
16             if Fob_2(S') = Fob_2(S) then
17                 Δ_E = (C_3(S') − C_3(S)) / max(C_3(S'), C_3(S)) ;
18             if Δ_E ≤ 0 then
19                 S = S' ;
20                 N_v(S) = ∅;
21             else
22                 if U(0,1) < exp(−Δ_E / Temp) then
23                     S = S' ;
24                     N_v(S) = ∅;
               /* Saves the best of the whole process                 */
25             if S ≺ S_best then
26                 S_best = S
27         Iters_O + + ;
28     Temp = α^{Iters_O} Temp_0;
```

Output: *Best Solution Found* $\mathcal{S}_{best}$

As in the ILS method, the SA stops after the outer iterations reach some fixed value.

Algorithm 21: Mating Selection

Input : Population P

```
/* Perform a binary tournament at each iteration                    */
```

1 **for** $i = 0; i < N; i++$ **do**

2 Select randomly $\mathcal{I}_j, \mathcal{I}_k \in P$; /* It could be that $\mathcal{I}_j = \mathcal{I}_k$ */

/* Assign the best individual to $\mathcal{I}'_i$ */

3 $\mathcal{I}'_i = best(\mathcal{I}_j, \mathcal{I}_k)$;

Output: Parents P'

3.2.3 Evolutionary-based Metaheuristics

Common Components

The majority of EAs make use of a set of components such as the operators: *initialization*, *evaluation*, *mating selection*, *crossover*, *mutation* and *survivor selection*. The operators used in this thesis are described below.

Initialization. The EAs implemented starts with an initial population P_0 generated at random. The population size is N, to generate the N individuals in the population the algorithm 14 is called N times.

Evaluation. The cost of each individual is calculated using the algorithm 15 for the calculus of the WHT, then the cost function C_3 3.17 is used to assign the cost value to each individual.

Mating Selection. Particularly, the mating selection is performed with a binary tournament, which is exemplified in algorithm 21.

Reproduction. After the mating selection 21, the offspring population is build. To generate it two variation operators are applied: *crossover* and *mutation*.

1. **Crossover**. Algorithm 22 describes how the crossover operator works. The crossover maintains balancedness in the offsprings. Lets see an example of how this operator works. Suppose that we have two parents $\mathcal{I}_1, \mathcal{I}_2$, each one representing a 4-variable BF. The elements in the intersection of the sets $\mathcal{I}_1.\mathcal{P}_1, \mathcal{I}_2.\mathcal{P}_1$ are shown in gray:

$\mathcal{I}_1 =$	1	1	0	1	0	1	1	0	1	0	1	0	0	0	1	0
$\mathcal{I}_2 =$	0	1	1	1	0	1	0	1	0	1	0	1	0	0	1	0

Algorithm 22: Crossover

Input : Parents $\mathcal{I}_j, \mathcal{I}_k$, cross probability p_c

1 *Set* $\mathcal{C}_j[i] = 0 \quad \forall i \in \mathcal{N}_n$;
2 *Set* $\mathcal{C}_k[i] = 0 \quad \forall i \in \mathcal{N}_n$;
/* With $\mathcal{I}_j, \mathcal{I}_k$ generate the offsprings $\mathcal{C}_j, \mathcal{C}_k$ */
/* First take the intersection of their $\mathcal{P}_1$ sets */
3 $\mathcal{Q}_{1_j} = \mathcal{I}_j.\mathcal{P}_1 \cap \mathcal{I}_k.\mathcal{P}_1$;
4 $\mathcal{Q}_{1_k} = \mathcal{I}_j.\mathcal{P}_1 \cap \mathcal{I}_k.\mathcal{P}_1$;
/* Take the differences */
5 $\mathcal{R}_{1_j} = \mathcal{I}_j.\mathcal{P}_1 \setminus \mathcal{Q}_{1_j}$;
6 $\mathcal{R}_{1_k} = \mathcal{I}_k.\mathcal{P}_1 \setminus \mathcal{Q}_{1_k}$;
/* Cross the elements from $\mathcal{R}_{1_j}$ with $\mathcal{R}_{1_k}$ */
7 **while** $\mathcal{R}_{1_k} \neq \emptyset$ **do**
8 *Select randomly* $r_j \in \mathcal{R}_{1_j}$ and $r_k \in \mathcal{R}_{1_k}$;
9 **if** $\mathcal{U}(0,1) < p_c$ **then**
10 $\mathcal{Q}_{1_j} = \mathcal{Q}_{1_j} \cup \{r_j\}$;
11 $\mathcal{Q}_{1_k} = \mathcal{Q}_{1_k} \cup \{r_k\}$;
12 **else**
13 $\mathcal{Q}_{1_j} = \mathcal{Q}_{1_j} \cup \{r_k\}$;
14 $\mathcal{Q}_{1_k} = \mathcal{Q}_{1_k} \cup \{r_j\}$;
15 $\mathcal{R}_{1_j} = \mathcal{R}_{1_j} \setminus \{r_j\}$;
16 $\mathcal{R}_{1_k} = \mathcal{R}_{1_k} \setminus \{r_k\}$;
/* With $\mathcal{Q}_{1_j}$ and $\mathcal{Q}_{1_k}$ build the offsprings $\mathcal{C}_j, \mathcal{C}_k$ */
17 *Set* $\mathcal{C}_j[i] = 1 \quad \forall i \in \mathcal{Q}_{1_j}$;
18 *Set* $\mathcal{C}_k[i] = 1 \quad \forall i \in \mathcal{Q}_{1_k}$;

Output: Offsprings $\mathcal{C}_j, \mathcal{C}_k$

So, we have the sets (lines 3, 4 from algorithm 22)

$$\mathcal{Q}_{1_1} = \mathcal{I}_1.\mathcal{P}_1 \cap \mathcal{I}_2.\mathcal{P}_1 = \{1,3,5,14\}$$
$$\mathcal{Q}_{1_2} = \mathcal{I}_1.\mathcal{P}_1 \cap \mathcal{I}_2.\mathcal{P}_1 = \{1,3,5,14\}$$

and the sets (lines 5, 6 from algorithm 22)

$$\mathcal{R}_{1_1} = \mathcal{I}_1.\mathcal{P}_1 \setminus \mathcal{Q}_{1_1} = \{0,6,8,10\}$$
$$\mathcal{R}_{1_2} = \mathcal{I}_2.\mathcal{P}_1 \setminus \mathcal{Q}_{1_2} = \{2,7,9,11\}$$

Now we make an uniform crossover with $\mathcal{R}_{1_1}$ and $\mathcal{R}_{1_2}$ (lines 7-16) and depending of the value for the cross probability p_c (line 9), we could have at the end of the crossover the sets

$$\mathcal{Q}_{1_1} = \{1,3,5,14,0,11,8,7\}$$
$$\mathcal{Q}_{1_2} = \{1,3,5,14,9,6,2,10\}$$

Algorithm 23: Mutation

Input : Individual $\mathcal{I}$, mutation probability p_m

1 **foreach** $p_1 \in \mathcal{I}.\mathcal{P}_1$ **do**

2 **if** $\mathcal{U}(0,1) < p_m$ **then**

/* Select $p_0 \in \mathcal{I}.\mathcal{P}_0$ at random to perform a bit-swapping */

3 $BS(\mathcal{I}, p_0, p_1)$;

Output: Individual mutated $\mathcal{I}$

We should remark, that there is not an established order in which the sets $\mathcal{R}_{1_1}$ and $\mathcal{R}_{1_2}$ are crossed.

From the sets $\mathcal{Q}_{1_1}$, $\mathcal{Q}_{1_2}$ we can build two individuals $\mathcal{C}_1, \mathcal{C}_2$ such that

$$\mathcal{C}_i[k] = \begin{cases} 1 & \text{if } k \in \mathcal{Q}_{1_i} \\ 0 & \text{if } k \in \mathcal{N}_n \setminus \mathcal{Q}_{1_i} \end{cases} \qquad i = 1,2 \tag{3.28}$$

$\mathcal{C}_1 =$	1	1	0	1	0	1	0	1	1	0	0	1	0	0	1	0
$\mathcal{C}_2 =$	0	1	1	1	0	1	1	0	0	1	1	0	0	0	1	0

It is important to note that when parents are close to each other, i.e. their differences are not large, then the crossover is not very destructive. In particular, if both parents are the same individual no changes are performed.

2. **Mutation**. Algorithm 23 describes how the mutation of an individual is performed. The mutation operator is used to perturb a single individual. The mutation probability p_m (line 2) is commonly set in such a way that, in average, it only makes one swap between a pair of distinct values in the individual $\mathcal{I}$, *i.e.* $p_m = 1/2^{n-1}$, where 2^{n-1} is the number of decision variables set equal to 1.

Algorithm 24 illustrates how to generate the offspring population by employing the crossover and mutation operators. Since in the mating selection phase the parents are added at random. In the reproduction phase the parents are combined two by two following the order in what they were added to the parent population. In this way it is ensured that each parent is used to generate offsprings just once.

Replacement. The majority of EAs use a Replacement Generational with Elitism (RGE) strategy based on using the offspring population with the best individual either from the parent or the offspring population. This operator is described in algorithm 25.

Most of the components — initialization, evaluation, mating selection and reproduction — are the same than in the basic EA. However, after the

Algorithm 24: Reproduction

Input : Parents P, cross probability p_c, mutation probability p_m

```
/* Generate two offsprings at each iteration                    */
```

1 **for** $i = 0; i < N; i+ = 2$ **do**

2 Select orderly $\mathcal{I}_i, \mathcal{I}_{i+1}$;

```
/* Perform crossover                                            */
```

3 $\mathcal{I}'_i, \mathcal{I}'_{i+1} = Crossover(\mathcal{I}_i, \mathcal{I}_{i+1}, p_c)$;

```
/* Perform mutation to each offspring                           */
```

4 $\mathcal{I}'_i = Mutation(\mathcal{I}'_i, p_m)$;

5 $\mathcal{I}'_{i+1} = Mutation(\mathcal{I}'_{i+1}, p_m)$;

Output: Offspring population P'

Algorithm 25: RGE Survivor Selection Technique

Input : Population P, offspring P'

1 *Set* $\mathcal{I} =$ individual with best cost in P;

2 *Set* $\mathcal{I}' =$ individual with best cost in P';

3 **if** $\mathcal{I} \prec \mathcal{I}'$ **then**

```
/* Replace at random any individual I'_r from P' with I         */
```

4 Select $\mathcal{I}'_r \in P'$;

5 *Set* $\mathcal{I}'_r = \mathcal{I}$;

6 *Set* $P = P'$;

Output: New population P

reproduction phase (crossover and mutation), a method to improve further the offspring is applied. In most cases, local search is applied. In this thesis, the FIQTS algorithm 18 is applied to every member of the offspring. There is just a subtle difference which is the stopping criterion. In this case, FIQTS stops after a certain period of time. The reason for this is that it is better to limit the amount of time to be able to evolve a high number of generations, instead of spending a lot of time on local search.

The simplest version of LMA 7 employs a replacement procedure that does not include diversity management: the RGE survivor selection technique, so this method will be called LMA-RGE.

Diversity Management

The first novelty of our approach is that instead of using a typical replacement operator, such as the generational with elitism, a replacement that takes into account the diversity is used. Particularly, a variant of the RMDDC strategy 8 was used. The novel variant is called Replacement with Elite based Dynamic Diversity Control (REDDC) and the difference is that, among the non-penalized individuals, the one with the lowest cost is selected. Note that in RMDDC, a multi-objective selection was used instead.

Algorithm 26: REDDC Survivor Selection Technique

Input : Population P, offspring P', distance D

1 *Set* $P_c = P \cup P'$; /* Current Population */
2 **foreach** $\mathcal{I} \in P_c$ **do**
3 $\quad \mathcal{I}_{.cost} = C_3(\mathcal{I})$; /* Cost of individual $\mathcal{I}$ */
4 $sort(P_c)$;
/* $\mathcal{I}_0$ is the individual with lowest cost in P_c */
5 $P = \{\mathcal{I}_0\}$; /* New population starts with the best */
6 $P_c = P_c \setminus \{\mathcal{I}_0\}$; /* Update current population */
7 **for** $i = 1; i < N; i++$ **do**
$\quad$ /* Calculate the distance to the closest neighbor (DCN) */
8 $\quad$ **foreach** $\mathcal{I} \in P_c$ **do**
9 $\quad\quad \mathcal{I}_{.DCN} = \min\{d_H(\mathcal{I}, \mathcal{I}') : \mathcal{I}' \in P\}$; /* Hamming distance 2.14 */
10 $\quad\quad$ **if** $\mathcal{I}_{.DCN} < D$ **then**
11 $\quad\quad\quad \mathcal{I}_{.cost} = \infty$; /* Penalize the closest */
12 $\quad sort(P_c)$;
13 $\quad \mathcal{S} = \mathcal{I}_0$; /* Select the best ($\mathcal{I}_0$) */
$\quad$ /* If all are penalized choose the farthest */
14 $\quad$ **if** $\mathcal{S}_{.cost} = \infty$ **then**
15 $\quad\quad \mathcal{S} = \mathcal{I}_k$ *s.t.* $\mathcal{I}_{.DCN} \leq \mathcal{I}_{k \cdot DCN}, \forall \mathcal{I} \in P_c$;
16 $\quad P = P \cup \{\mathcal{S}\}$;
17 $\quad P_c = P_c \setminus \{\mathcal{S}\}$;

Output: New population P

Preliminary results showed that using RMDDC induced a too slow convergence in this problem, so it was discarded and REDDC proposed. Algorithm 26 illustrates how the REDDC strategy works.

Note that in the name of the new method the references to multi-objective were suppressed. The reason is that is the new strategy there is no need to calculate the non-dominated set, meaning that less computation steps are required to perform the replacement. In each step, the non-penalized individual with best cost is always selected (line 13). Since the population is sorted at each iteration (line 12), if the cost of the selected individual is ∞, this means that every individual is penalized. Thus, in such a case the individual which contributes more to the diversity is added (line 15). Note that the sorting strategy might be avoided with the aim of reducing the number of steps. However, this would not result in an important gain because this part of the algorithm is not computationally expensive. As in the RMDDC strategy the parameter D is responsible for exploring the search space towards find promising regions in it.

The LMA 7 that employs the REDDC survivor selection technique 26, is called LMA-REDDC. This is the first evolutionary method proposed to generate BFs with high nonlinearity that incorporates a diversity-based scheme.

Algorithm 27: REDDCC Survivor Selection Technique

Input : Population P, offspring P', distances D and D_C, cluster size C_{size}

1 *Set* $P_c = P \cup P'$; /* Current Population */
2 **foreach** $\mathcal{I} \in P_c$ **do**
3 $\quad \mathcal{I}_{.cost} = C_3(\mathcal{I})$; /* Cost of individual $\mathcal{I}$ */
4 $sort(P_c)$;
/* $\mathcal{I}_0$ is the individual with lowest cost in P_c */
5 $P = \{\mathcal{I}_0\}$; /* New population starts with the best */
6 $P_c = P_c \setminus \{\mathcal{I}_0\}$; /* Update current population */
/* The first cluster with one element is formed */
7 $C_0 = \{\mathcal{I}_0\}$; /* Cluster centered in $\mathcal{I}_0$ */
8 $\mathcal{C} = \{C_0\}$; /* Clusters set */
9 **for** $i = 1; i < N; i++$ **do**
$\quad$ /* Calculate the distance to the closest neighbor (DCN) */
10 $\quad$ **foreach** $\mathcal{I} \in P_c$ **do**
$\quad\quad$ /* Distance given by the Hamming distance 2.14 */
11 $\quad\quad \mathcal{I}_{.DCN} = \min\{d_H(\mathcal{I}, \mathcal{I}') : \mathcal{I}' \in P\}$;
12 $\quad\quad$ **if** $\mathcal{I}_{.DCN} < D \wedge \mathcal{I}' \in C_k \wedge |C_k| = C_{size}$ **then**
13 $\quad\quad\quad \mathcal{I}_{.cost} = \infty$; /* Penalize the closest to P */
14 $\quad\quad$ **if** $\mathcal{I}_{.DCN} < D_C$ **then**
15 $\quad\quad\quad \mathcal{I}_{.cost} = \infty$; /* Penalize the closest to any C_k */
16 $\quad sort(P_c)$;
17 $\quad \mathcal{S} = \mathcal{I}_0$; /* Select the best ($\mathcal{I}_0$) */
$\quad$ /* If all are penalized choose the farthest */
18 $\quad$ **if** $\mathcal{S}_{.cost} = \infty$ **then**
19 $\quad\quad \mathcal{S} = \mathcal{I}_k$ *s.t.* $\mathcal{I}_{.DCN} \leq \mathcal{I}_{k \cdot DCN}, \forall \mathcal{I} \in P_c$;
20 $\quad C_i = \{\mathcal{S}\}$; /* Cluster centered in $\mathcal{S}$ */
21 $\quad \mathcal{C} = \mathcal{C} \cup C_i$;
22 $\quad$ **for** $j = 0; j < i; j++$ **do**
23 $\quad\quad \mathcal{S}_{.DCN} = d_H(\mathcal{S}, \mathcal{I}_j)\ \ \mathcal{I}_j \in P;$
24 $\quad\quad$ **if** $\mathcal{S}_{.DCN} < D$ **then**
25 $\quad\quad\quad C_j = C_j \cup \{\mathcal{S}\}$; /* Update the clusters previously added */
26 $\quad\quad\quad C_i = C_i \cup \{\mathcal{I}_j\}$; /* Update the current cluster */
27 $\quad P = P \cup \{\mathcal{S}\}$; /* Update the new population */
28 $\quad P_c = P_c \setminus \{\mathcal{S}\}$; /* Update the current population */

Output: New population P, clusters $\mathcal{C}$

Diversity Management with Clustering

One of the problems with the previous strategy, *i.e.* with the REDDC method, is that no individual with a distance lower than D is accepted (line 10). Usually, when crossover is applied to distant individuals, an exploration step is performed. While maintaining distant individuals to

explore different regions is important, using close individuals to better intensify is also interesting. For this reason, a novel algorithm that tries to explore and intensify during the whole search is proposed.

The main difference with respect to the previous algorithm is the inclusion of a clustering technique, that as mentioned, induce the maintenance of distant individuals but also allows the acceptance of some close individuals. The REDDC with Clustering (REDDCC) strategy is explained in algorithm 27.

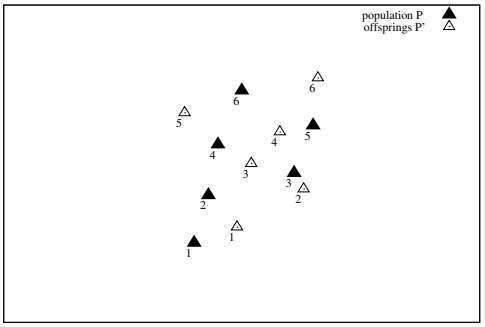
The key of the REDDCC is to alter the replacement strategy by applying a clustering technique that takes into account the stopping criterion. The aim of this novel replacement strategy is to attain a dynamic balance between exploration and exploitation.

In the majority of evolutionary methods we have not control about the diversity of the population, then we introduce the REDDC strategy in order to control the diversity of the population, but at the time the results were not promising and we decided to incorporate a clustering technique. The main difference between the REDDC and the REDDCC strategies is that the the REDDCC strategy allows a certain number of individuals (cluster size C_{size}) to be closer than D (but farther than $D_C < D$), while in the REDDC strategy this is not allowed.

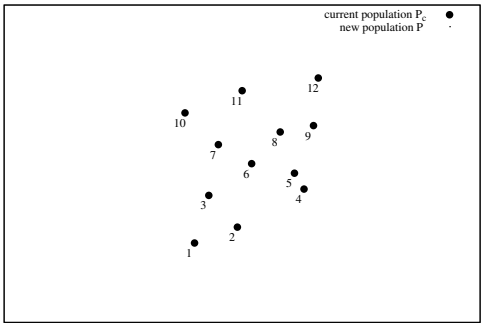
Working Operation of REDDCC

First the population of the previous generation P and the offsprings P' are combined in the current population P_c (line 1). Then, each individual from P_c is evaluated by taking into account its cost (line 3). In order to perform an elitist strategy, the best individual — the one with minimum cost ($\mathcal{I}_0$) — is selected to survive by placing it in the new population (we simply overwrite the population P) and removing it from P_c (lines 5-6). The first cluster with seed $\mathcal{I}_0$ is added to the clusters set (lines 7-8). In the for cycle the $N-1$ individuals remaining are added to the new population (line 9). In order to select the elite individual in P_c the following steps are executed (lines 10-28). First, the contribution to diversity of each individual in P_c is calculated (line 11), this is done by taking into account the minimum distance between each individual and all the individuals in P. If the individual does not contribute enough to the diversity, the clustering technique penalizes it by setting its cost to infinity (lines 12-14). Basically, it is penalized when it is too close to a cluster that is already full or when it is closer than D_C. Then the best individual in the P_c is *Selected* ($\mathcal{S}$) to survive (line 17). In case that all individuals are penalized, the one that contributes more to the diversity is chosen (line 19). A new cluster is created by using the new selected individual as the seed (line 20). Then, the clusters are updated with $\mathcal{S}$ and the individuals in P (lines 21-26). Finally $\mathcal{S}$ is added to P and removed from P_c (lines 27-28).

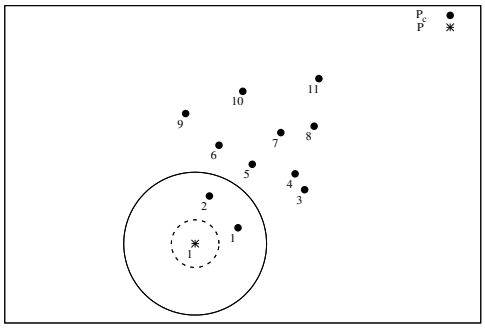
FIGURE 3.3: Illustration of the REDDCC survivor selection technique



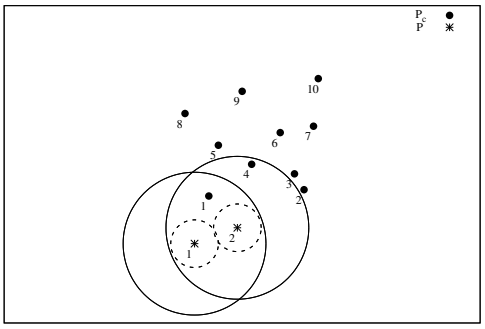
(A) P and P' populations



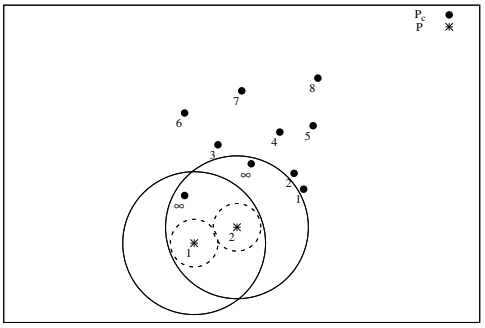
(B) Current population P_c (lines 1-4)



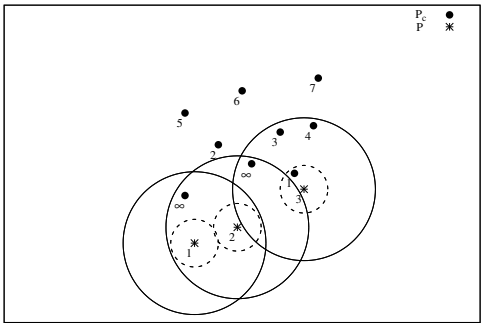
(C) Add the best to P (lines 5-8)



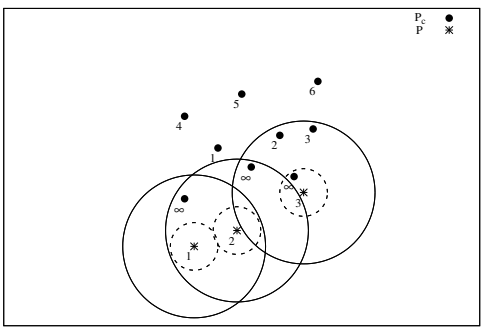
(D) Starts to fill P (line 9)



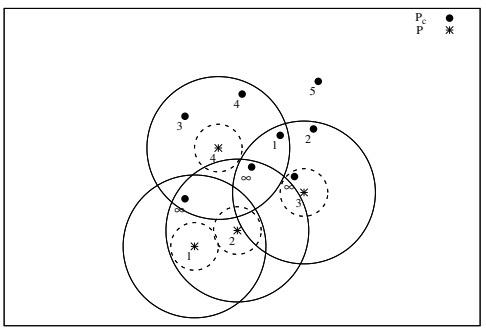
(E) Penalize individuals (line 12)



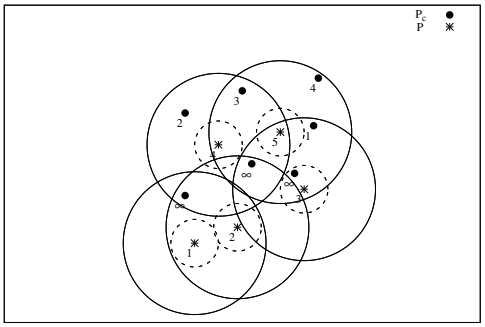
(F) Add the best individual (line 17)



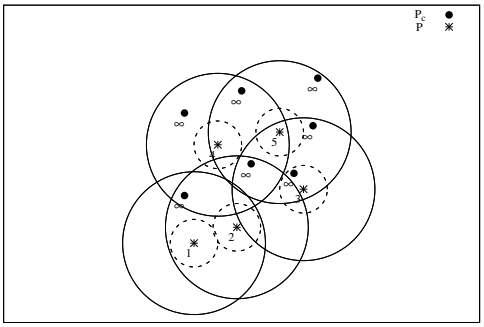
(G) Penalize closest (line 14)



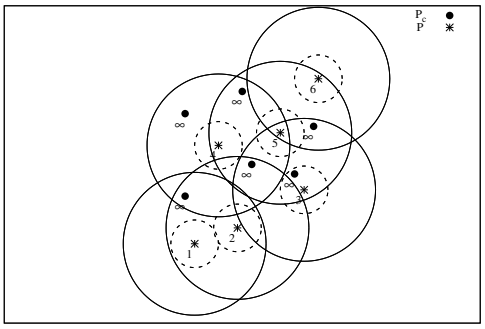
(H) Add the best individual (line 17)



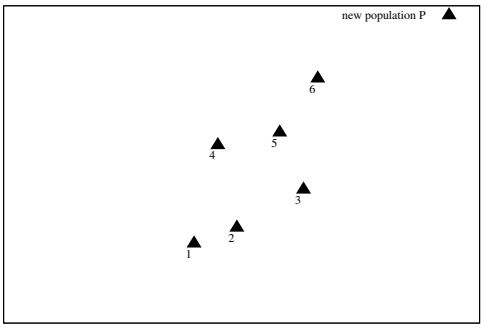
(I) Add the best individual (line 17)



(J) Penalize individuals (line 12)



(K) Add the farthest individual (line 19)



(L) New population

To illustrate the process below, the figure 3.3 shows in an interactive way how the REDDCC strategy works when a $C_{size} = 2$ is used. First consider a population P and its corresponding offspring population P', which are the inputs of the algorithm 27, we can see them as two sets of points as in the figure 3.3a. In the figure 3.3b we can see both populations P, P' combined into the current population P_c, as the points are numbered from 1 to 12 we can consider that they are sorted according to its cost and the point with the label "1" is the best of the population. The lines 1-4 of the algorithm 27 are represented in the figure 3.3b. The selection of the best individual in P_c to survive is represented in the figure 3.3c, it can be seen around this individual two circles that represent the prohibited zones, one of radius D (the solid circle) and another one of radius D_C (the dashed circle). The solid circle can be viewed as the boundary of each cluster, meaning that only C_{size} individuals can be inside of each cluster (line 12). Within the dashed circle there should never be more than one individual (line 14). After adding the best individual to P, the iterative cycle starts in order to fill P with those individuals that meet the conditions of quality and diversity. The figure 3.3d illustrate the result obtained after the first iteration of the for cycle (line 9). We can see that the first two individuals added in the figure 3.3d belong to the same clusters and each one of the formed clusters is already full, this is why the individual with the best cost in P_c and the labeled with "4" are penalized in the figure 3.3e. The figure 3.3f shows the new individual added to P in the second iteration. As the best in P_c violates the distance to the seed of some cluster is penalized in the figure 3.3g. The figure 3.3h shows the

Algorithm 28: Mating Selection with Clusters

Input : Population P, clusters $\mathcal{C}$, p_{cc}

```
/* Perform two binary tournament at each iteration */
```

1 **for** $i = 0; i < N; i+ = 2$ **do**

```
    /* Choose two parents from different clusters */
```

2 **if** $\mathcal{U}(0,1) < p_{cc}$ **then**

3 Select randomly $\mathcal{I}_{j_1}, \mathcal{I}_{j_2} \in \mathcal{C}_j$ and $\mathcal{I}_{k_1}, \mathcal{I}_{k_2} \in \mathcal{C}_k, k \neq j$;

4 $\mathcal{I}'_i = best(\mathcal{I}_{j_1}, \mathcal{I}_{j_2})$;

5 $\mathcal{I}'_{i+1} = best(\mathcal{I}_{k_1}, \mathcal{I}_{k_2})$;

```
    /* Choose two parents in the same cluster */
```

6 **else**

7 Select randomly $\mathcal{I}_{k_1}, \mathcal{I}_{k_2}, \mathcal{I}_{k_3}, \mathcal{I}_{k_4} \in \mathcal{C}_k$;

8 $\mathcal{I}'_i = best(\mathcal{I}_{k_1}, \mathcal{I}_{k_2})$;

9 $\mathcal{I}'_{i+1} = best(\mathcal{I}_{k_3}, \mathcal{I}_{k_4})$;

Output: Parents P'

new individual added to P in the third iteration. The figure 3.3i shows the new individual added to P in the fourth iteration. We can see that the cluster with seed 5* has size 3, as the clusters with seeds 3* and 4* were not full at the moment that the fifth individual was added to P, 5* was accepted to survive and it was until the update of the clusters (lines 20-26) was done when the method realizes that the current cluster exceed the permitted size, since that update part is done later, the number of elements in the clusters can be $C_{size} + 1$. However this kind of situation occurs with very low probability for populations with many individuals. The figure 3.3j shows that all the individuals remaining in P_c are penalized, since all of them are close to some cluster already full. In order to fill P the farthest individual is added as is showed in the figure 3.3k. The figure 3.3l shows the individuals selected to survive according to the REDDCC strategy.

Memetic Algorithm with Clusters

The LMA that incorporates clustering techniques, is called Memetic Algorithm with Clusters (MAC). This is the most novel algorithm developed for this thesis. The main novelty appears in the replacement phase, which is the REDDCC survivor selection technique 27.

Some components of the MAC method are the same than those from the LMA method. The reproduction procedure 24 is exactly the same. When individuals that belong to the same cluster are crossed, an intensification step is performed, whereas when they belong to a different cluster, exploration is induced. Algorithm 28 illustrates how to perform a mating selection in a population with clusters. This new selection procedure is necessary because in the reproduction phase parents are crossed two by two following the order in what they were added to the parent population. In

Algorithm 29: Improvement

Input : Population P, T_{ls}, K_{ls}

```
/* Perform the improvement to the population                    */
```

1 **for** $i = 0; i < N; i++$ **do**
2 **if** $\mathcal{I}_i$ *was generated with parents belonging to different clusters* **then**
3 $\mathcal{I}_i = FIQTS(\mathcal{I}_i, K_{ls} \times T_{ls})$;
4 **else**
5 $\mathcal{I}_i = FIQTS(\mathcal{I}_i, T_{ls})$;

Output: Population improved P

order to better control the behavior of the MAC method, a new parameter, the p_{cc} is added. This parameter refers to the probability that two individuals from different clusters are crossed in order to create a pair of offsprings (line 2). In other cases, individuals that belong to the same cluster are crossed.

In the case of the improvement phase, the only difference is that we grant more time for local search when the offspring is generated by crossing parents from different clusters. The reason is that in such a case the crossover operator is more disruptive, so it is expected that more time is required to attain a new high-quality solution. Algorithm 29 illustrates how the improvement phase works. A new parameter, the K_ls is added. This parameter refers to the local search time granted to those offsprings generated with parents belonging to different clusters (line 3). If the parents of the offspring belong to the same cluster the local search time granted is T_{ls} (line 5).

The MAC coupled with the REDDCC survivor selection technique 27, is called MAC-REDDCC. This is the most novel evolutionary method that incorporates a diversity scheme with a clustering technique. Algorithm 30 illustrates how the MAC-REDDCC method works.

Working Operation of MAC-REDDCC

The MAC-REDDCC method can be viewed as an extension of the LMA-REDDC method. The working operation of the MAC-REDDCC method is different from the LMA-REDDC method, this is because to the incorporation of the clustering paradigm. First, after the initialization of the population, the clusters set is initialized by considering that each individual makes up a different cluster (line 3). All phases are carried out with their corresponding parameters according to their previous descriptions.

At this point, the importance of the parameters D and D_C has not been explained in detail. In this thesis the value D represents the minimum DCN required to avoid penalty when a individual is close to a full cluster and the value D_C represents the minimum DCN in a Cluster (DCNC) required to

Algorithm 30: MAC-REDDCC Method

Input : $D_{C_0}, D_0, C_{size}, N, T_{ls}, K_{ls}, p_{cc}, p_m, p_c, K_E, \quad T_{end}$

1 **Initialize**(P_0) ;
2 **Evaluate**(P_0);
/* Each individual makes up a different cluster */
3 $\mathcal{C} = \{C_k : C_k = \{\mathcal{I}_k\}, \mathcal{I}_k \in P_0\}$;
4 **Improvement**(P_0, T_{ls}, K_{ls}) ;
5 $t = 0, T_{elapsed} = 0$;
6 $T_{exploration} = K_E \times T_{end}$;
7 **while** $T_{elapsed} < T_{end}$ **do**
8 $P'_t =$ **Mating Selection with Clusters**$(P_t, \mathcal{C}, p_{cc})$;
9 $P'_t =$ **Reproduction**(P'_t, p_c, p_m);
10 **Evaluate**(P'_t);
11 $P'_t =$ **Improvement**(P'_t, T_{ls}, K_{ls}) ;
12 **if** $T_{elapsed} \leq T_{exploration}$ **then**
13 $D = D_0 - (D_0 - D_{C_0}) \times T_{elapsed}/T_{exploration}$;
14 $D_C = D_{C_0} - D_{C_0} \times T_{elapsed}/T_{exploration}$;
15 **else** $D = D_{C_0}, D_C = 0$;
16 $P_{t+1}, \mathcal{C} =$ **REDDCC** $(P_t, P'_t, D, D_C, C_{size})$;
17 *Update*$(T_{elapsed})$;
18 $t = t + 1$;

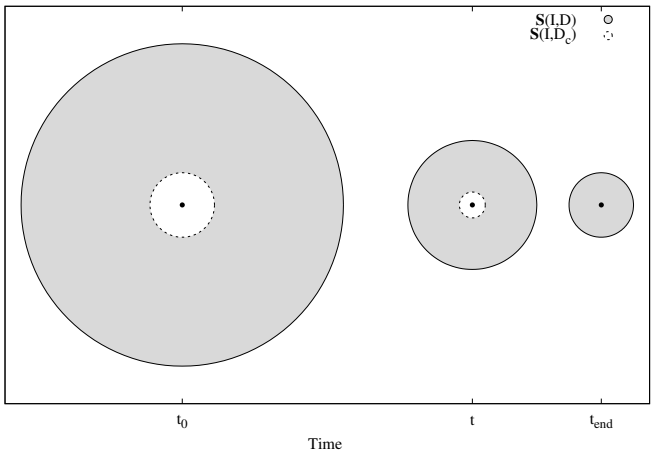
Output: Best solution(s) found

avoid penalty for individuals that could belong to the same cluster. These two parameters should vary during the optimization process. Specifically, these values should decrease as the stopping criterion is approached with the aim of inducing more exploration in the initial stages, whereas more intensification is performed in the last stages. In this thesis the stopping criterion is set by time.

Preliminary results showed that more intensification is performed without loss of diversity by setting D and D_C to very small values after a percentage of the granted execution time (T_{end}) and not necessarily at the end. Taking this into account, a new parameter, K_E is added. This parameter refers to the percentage of granted time to explore the search space during the optimization process (line 6).

In our scheme, an initial D_0 and D_{C_0} values must be selected by the user. Then, a linear reduction of D and D_C is carried out (lines 13-14) in such a way that by the end of the exploration time (line 12), the resulting values are $D = D_{C_0}$ and $D_C = 0$ (line 15). To illustrate it, the figure 3.4 shows an example in 2D of how varies the prohibited zones (displayed as circles) along the execution time.

FIGURE 3.4: Variation of the prohibited zones along the execution time for the MAC.



3.2.4 Hybridization with an Algebraic Technique

Motivated by the related work from Burnett [5] and Izbenko [19] we considered the application of an algebraic procedure to improve our results. Contrary to their work we decide not to use bent functions, instead we employ the work from Tang [52] to build a population of individuals with high nonlinearity to start to evolve our algorithms. The main difference between this hybrid method is in the *initialization* procedure of the population.

Tang [52] propose a method capable to construct balanced BFs of even number of variables with high nonlinearity. Their construction is based on a modification of the Maiorana-McFarland [28] method to construct bent BFs. As a result Tang et al are able to obtain a large amount of balanced BFs with nonlinearity

$$N_n \geq 2^{n-1} - 2^{n/2-1} - 2^{\lceil n/4 \rceil} \tag{3.29}$$

The reason behind this interesting fact about the nonlinearity of the BFs constructed is briefly explained in the following. This method requires a balanced BF h of $n/2-1$ variables such that its nonlinearity is maximal *i.e* $N_{n/2-1}(h)$ is as high as possible. Algorithm 31 illustrates how to construct a balanced BF with high nonlinearity.

In order to explain the algorithm 31 consider that we require to construct a BF of 10 variables with high nonlinearity. The nonlinearity of the constructed BF should be $N_{10} \geq 2^9 - 2^4 - 2^{\lceil 10/4 \rceil} = 488$. We need a BF $h \in \mathcal{B}_4$ such that its nonlinearity is $N_4 = 4$ (highest possible according to the upper bound 2.40). Initially the BF f that will be constructed is set equal to zero in all its truth table positions (line 1), then all the truth table positions are filled in the for cycle (line 4), then $\mathbf{x}$, $\mathbf{y}$ are considered in such a way that their concatenation is $\mathbf{z}$ and $\mathbf{y}'$ is set to be equal to the first $n/2-1$ components of $\mathbf{y}$ in order to use it to evaluate h. The permutation (line 8) is to help us to construct many

Algorithm 31: Algebraic Construction of a Balanced BF

Input : $h \in \mathcal{B}_{n/2-1}$

Set $f(\mathbf{z}) = 0 \quad \forall \mathbf{z} \in \mathbb{F}_2^n$;
Set $\mathbf{0} = (0, \cdots, 0)$;
Set $\mathbf{1} = (1, \cdots, 1)$;
foreach $\mathbf{z} \in \mathbb{F}_2^n$ **do**
    Set $\mathbf{x} = (z_1, \cdots, z_{n/2})$;
    Set $\mathbf{y} = (z_{n/2+1}, \cdots, z_n)$;
    Set $\mathbf{y}' = (y_1, \cdots, y_{n/2-1})$;
    `/* Optional: permute x */`
    $\mathbf{x} = (z_\alpha, \cdots, z_\beta)$, such that $\alpha, \beta \in \{1, \cdots, n/2\}$;
    if $\mathbf{x} = \mathbf{0}$ **then** $f(\mathbf{z}) = (\mathbf{1} \cdot \mathbf{y}) h(\mathbf{y}')$;
    else if $\mathbf{x} = \mathbf{1}$ **then** $f(\mathbf{z}) = (\mathbf{1} \cdot \mathbf{y}) h(\mathbf{y}') \oplus \mathbf{1} \cdot \mathbf{y} \oplus 1$;
    else $f(\mathbf{z}) = \mathbf{x} \cdot \mathbf{y}$;

Output: BF f with high nonlinearity

BFs with the same h, for this example we can construct $5! = 120$ different BFs with nonlinearity $N_{10} = 488$. We should remark that the nonlinearity of the resulting BF f strongly depends on the nonlinearity of h, this is because the inequality 3.29 is a particular case of the following expression

$$N_n(f) \geq 2^{n-1} - \frac{1}{2} \max_{\mathbf{x} \in \mathbb{F}_2^n} |W_n(f)(\mathbf{x})| \tag{3.30}$$

where

$$|W_n(f)(\mathbf{x})| \leq 2^{n/2} + 2 \max_{\mathbf{y}' \in \mathbb{F}_2^{n/2-1}} |W_{n/2-1}(h)(\mathbf{y}')| \tag{3.31}$$

According to this is better to write the inequality 3.29 as

$$N_n(f) \geq 2^{n-1} - 2^{n/2-1} - \max_{\mathbf{y}' \in \mathbb{F}_2^{n/2-1}} |W_{n/2-1}(h)(\mathbf{y}')| \tag{3.32}$$

Where $W_{n/2-1}(h)(\mathbf{y}')$ corresponds to the WHT of h and its maximum absolute value is desired to be $2^{\lceil n/4 \rceil}$. The reader is referred to [52] to see the proof of the fact that the BFs constructed with this method have nonlinearity as is indicated in the inequality 3.29 (or in general 3.32).

Chapter 4

Experimental Validation

In this chapter we expose the experimental validation performed in order to show our main results. Although along this research, several additional experiments were carried out, only the most representative that allow us to proof the hypothesis of the dissertation are exposed in the following.

4.1 Scheme of Experimental Validation

The majority of our experiments were carried out using 8 and 10 variables for the boolean functions, meaning that candidate solutions consisted of 2^8 and 2^{10} boolean decision variables, respectively.

Tests have been run on the cluster "El Insurgente", using bi-processor machines with 32 GB of RAM. Each processor is an Intel Xeon CPU E5-2620 at 2.1 GHz. Since the optimization methods devised in this thesis are stochastic, each experiment was carried out by executing 50 independent runs in every case. In order to compare the results, a set of statistical tests that relied on a guideline similar to that applied in [47] was conducted. Specifically, the following test were applied, assuming a significance level of 5%. First, a *Shapiro-Wilk test* was applied to check whether or not the results followed a Gaussian distribution. If they did, the *Levene test* was used to check for the homogeneity of the variances. If the samples had equal variance, an *ANOVA test* was done; if not, a *Welch test* was performed. For non-Gaussian distributions, the non-parametric *Kruskal-Wallis test* was used to test whether samples were drawn from the same distribution.

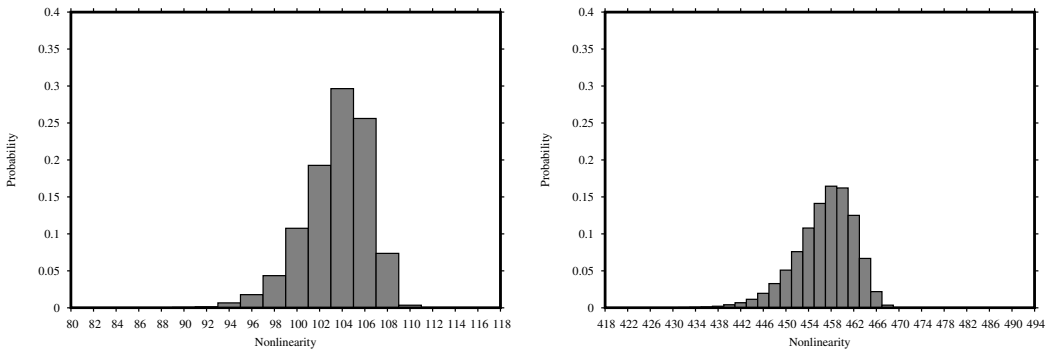
In the following are shown the comparatives between different methods, cost functions and parameter values for the same method. In this thesis, the sentence "*method A is better than method B*" means that the differences between them are statistically significant, and that the mean and median obtained by A are higher than the mean and median achieved by B. For each comparative table, columns with the symbol $\uparrow$ show the number of cases where the method listed in each row is statistically better. The number of cases where it is statistically worse is shown in the column with the symbol $\downarrow$. Finally, the number of cases where the differences are not statistically significant are shown in the column with the symbol $\leftrightarrow$. In

addition, a score can be assigned to each method. This score is equal to the number of cases where the method was statistically better minus the number of cases where the method was statistically worse. The previous considerations have the same meaning if we replace "method" with "cost function" or "parameter value".

4.2 Neighborhoods and Cost Functions

In our preliminary validations, we realized that using the objective function $\mathbf{F_2}$ 3.13 — or cost function C_1 3.14 — attains a really bad performance when is used to achieve solutions with high nonlinearity. In order to compare the four cost functions C_1 3.14, C_2 3.15, C_3 3.17, C_4 3.18 we analyzed the features of the neighborhoods of different n-variable BFs. When BFs are generated at random, figure 4.1 shows the corresponding distribution of the obtained quality for 8 and 10 variables.

FIGURE 4.1: Nonlinearity distribution of 2^{2n-2} BFs generated at random



(A) 8-variables (B) 10-variables

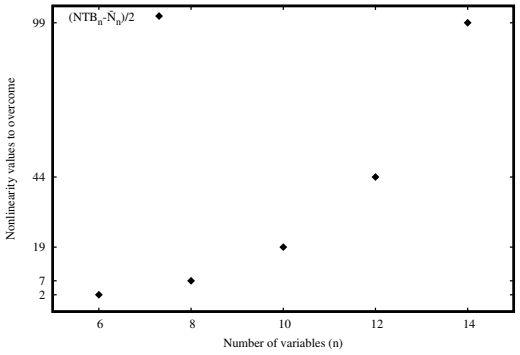
In order to plot the distributions shown in figure 4.1, 2^{2n-2} independent solutions were generated at random. Table 4.1 shows the $\min_{N_n}$, $\max_{N_n}$, mean ($\bar{N}_n$), median ($\tilde{N}_n$) and standard deviation (σ_{N_n}) of the nonlinearity of such solutions. It also shows the nonlinearity theoretical bound (NTB_n) for each n.

TABLE 4.1: Nonlinearity values for 2^{2n-2} solutions generated at random

n	Min	Max	$\bar{N}_n$	$\tilde{N}_n$	σ_{N_n}	NTB_n
6	16	24	21.5	22	1.517	26
8	88	110	103.5	104	2.882	118
10	416	472	456.9	458	5.266	494
12	1826	1954	1926.3	1928	9.774	2014
14	7754	7982	7927.4	7930	18.27	8126

In order to show that the quality of random solutions decrease for BFs with more variables, figure 4.2 shows the average number of nonlinearity values that need to be overcome by a solution generated at random to achieve the NTB.

FIGURE 4.2: Nonlinearity values that need to be overcome to reach the nonlinearity theoretical bound at random generation.



Note that although it is not possible to do an exhaustive search for BFs with more than 5 variables, the 6-variable BF problem is practically solved. In fact, BFs with maximum nonlinearity are generated at random for the 6-variable BF problem if a large number of BFs are generated. The 8-variable BF problem has been widely worked. However, a solution with a nonlinearity equal to the NTB has never been found. In any case, we can see in figure 4.2 that solutions generated at random are not so far in nonlinearity to the NTB. Nowadays, it is relatively easy to achieve the maximum nonlinearity known for the 8-variable BF problem ($N_8 = 116$). However, few methods are able to get the best-known solutions for higher number of variables. The next step for metaheuristics is to achieve the maximum nonlinearity known for the 10-variable BF problem ($N_{10} = 492$) and for the hybrid case, this was achieved in this thesis.

Taking into account the previous results for random generation of BFs, we selected 15 different nonlinearity values $(V(n))$ for $n = 8, 10$ by considering the results in table 4.1. For each one of the 15 nonlinearity values $N_n \in V(n)$ we generated independently 50 different solutions $\mathcal{S}$ to perform our experiments. The chosen nonlinearity values were the following:

- $V(8) = \{88 + 2i : i = 0, \cdots 14\}$
- $V(10) = \{422 + 4i : i = 0, \cdots 14\}$

Among the chosen nonlinearities values, some of them were very low or very high to be generated consistently with random generation. Thus, all the solutions to do this test were generated with the local search method FIQTS 18. In the cases where low nonlinearities are required, we only need to alter the line 13 with the condition $\mathcal{S} \prec \mathcal{S}'$. In order to avoid bias in the validation of the cost functions at exploring the neighborhood, it was checked that the minimum distance between any two solutions at fixed

nonlinearity was large enough. This minimum distance between any two solutions for $n = 8$ is 42.79% of the number of variables and for $n = 10$ is 46.72%, meaning that all solutions are distant.

For each single solution $\mathcal{S}$ with fixed nonlinearity value, its whole neighborhood $N(\mathcal{S})$ is explored in order to calculate the following probabilities:

- $P_I(\mathcal{S}', \mathcal{S})$ A neighbor $\mathcal{S}'$ has *Identical* cost as $\mathcal{S}$
- $P_B(\mathcal{S}', \mathcal{S})$ A neighbor $\mathcal{S}'$ has *Better* cost (lower) than $\mathcal{S}$
- $P_W(\mathcal{S}', \mathcal{S})$ A neighbor $\mathcal{S}'$ has *Worse* cost (higher) than $\mathcal{S}$

Note that a solution has a neighborhood size 3.10 equal to 2^{2n-2} which is a quite large value even for small n. Figure 4.3 shows the plot that describes the neighborhood size for n ranging between 6 and 14.

FIGURE 4.3: Neighborhood size for different values of n.

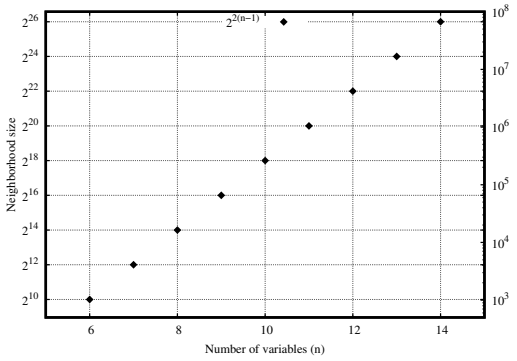
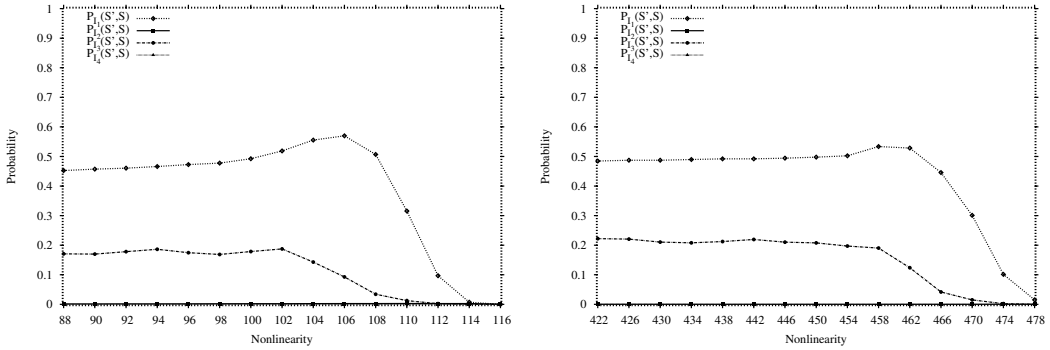


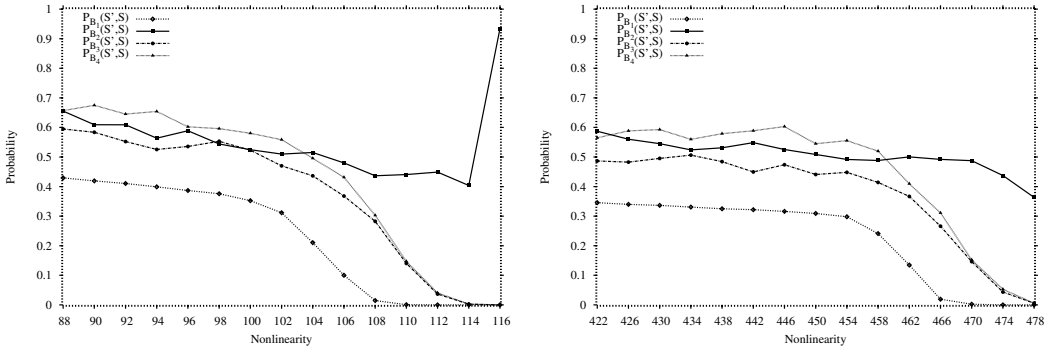
FIGURE 4.4: $P_I(\mathcal{S}', \mathcal{S})$ A neighbor $\mathcal{S}'$ has *identical* cost as $\mathcal{S}$



(A) 8-variables (B) 10-variables

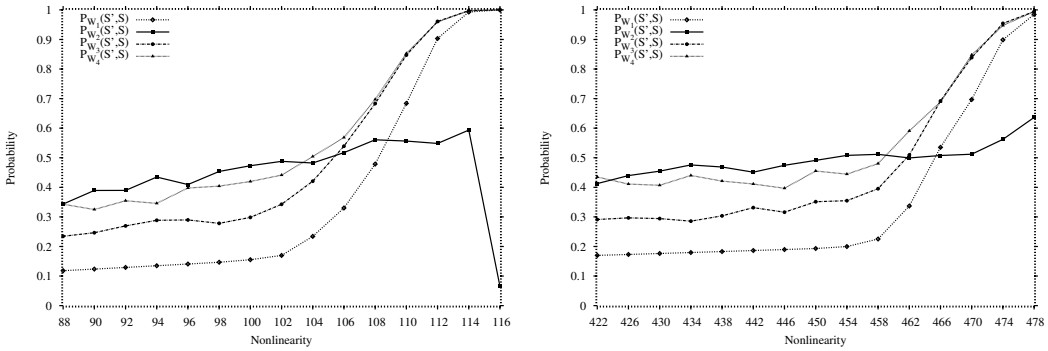
Figure 4.4 shows the average of $P_{I_i}(\mathcal{S}', \mathcal{S})$ probabilities calculated using each cost function C_i for all the solutions $\mathcal{S}$ corresponding to the values $N_n \in V(n)$. As we can see, for the cost function C_1 3.14, a neighbor $\mathcal{S}'$ is equal in quality as the original solution $\mathcal{S}$ with a high probability for the different values of nonlinearity. This means that a high percentage of the

FIGURE 4.5: $P_B(\mathcal{S}', \mathcal{S})$ A neighbor $\mathcal{S}'$ has *better* cost that $\mathcal{S}$



(A) 8-variables (B) 10-variables

FIGURE 4.6: $P_W(\mathcal{S}', \mathcal{S})$ A neighbor $\mathcal{S}'$ has *worse* cost that $\mathcal{S}$



(A) 8-variables (B) 10-variables

neighbors can be mapped into the same cost, which is not effective in order to guide the search. The cost function C_3 3.17 presents a lower probability $P_{l_3}(\mathcal{S}', \mathcal{S})$, although is not very small. In the other extreme the cost functions C_2 3.15 and C_4 3.18 practically do not map neighbors $\mathcal{S}'$ of the solution $\mathcal{S}$ to the same cost. This is a good indicator for these cost functions and this is due to the fact that these cost functions employs more information of the WHT than the cost functions C_1 3.14 and C_3 3.17, which only uses one and two values respectively. However, one might argue that using all the values simultaneously might not be meaningful for the proper guide of the search, so C_3 3.17 might afford a proper balance.

Figure 4.5 shows the average of the $P_{B_i}(\mathcal{S}', \mathcal{S})$ probabilities calculated using each cost function C_i for all the solutions $\mathcal{S}$ corresponding to the values $N_n \in V(n)$. The probabilities are high when the nonlinearity of the individuals is low and the probability decreases as the nonlinearity increases. A contrast for this graphic is in the figure 4.6 which plots the average of the $P_{W_i}(\mathcal{S}', \mathcal{S})$ probabilities calculated using each cost function C_i for the solutions $\mathcal{S}$ corresponding to the values $N_n \in V(n)$. We can see in the figure 4.6a that the cost function has a contradictory behavior, because it is indicating that when a solution is a high-quality one, the cost of its neighbors is worse.

4.3 Trajectory-based Metaheuristics and Cost Functions

The analyses presented in section 4.2 show some undesired features of one the most used cost function nowadays. However, in order to properly compare different cost functions it is important to include them in an optimization scheme. This section includes a comparison for the four cost functions within the local search methods FHC 17 and FIQTS 18. The main aim is to identify which cost function is more suitable so that experiments with MAC-REDDCC 30 —which are more computationally expensive— can be performed with a single cost function.

4.3.1 Full Hill Climbing

In this method the four cost functions to guide the search were tested for $n = 8, 10$. Table 4.2 shows the statistical results obtained for 50 independent runs. FHC was executed until a local optimum is reached.

TABLE 4.2: Comparatives for the FHC method

n	CF	↑	↓	↔	Min	Max	$\bar{N}_n$	$\tilde{N}_n$	σ_{N_n}	$\bar{t}(s)$
8	C_1	0	3	0	108	112	110.16	110	1.503	3.4e-2
	C_2	2	1	0	112	116	114.12	114	0.627	1.6e+0
	C_3	**3**	**0**	**0**	**114**	**116**	**115.92**	**116**	**0.396**	2.4e-1
	C_4	1	2	0	108	112	111.84	112	0.792	3.1e-2
10	C_1	0	3	0	464	472	469.4	470	1.863	7.9e-1
	C_2	2	1	0	482	484	482.72	482	0.97	1.8e+2
	C_3	**3**	**0**	**0**	**480**	**484**	**483.16**	**484**	**1.621**	1.6e+1
	C_4	1	2	0	468	480	475.44	476	1.809	1.9e+0

The best results obtained for each n are shown in bold. We can see that the cost function C_3 3.17 attains the best mean nonlinearity in both cases. If we look at the average time column, we see that the cost function C_3 3.17 employs less execution time to achieve a local optimum than the cost function C_2 3.15, which results are close to the results with the cost function C_3 3.17.

4.3.2 First Improvement Quasi-Tabu Search

As in the FHC method, FIQTS method was executed with the four cost functions. Table 4.3 shows the results obtained with FIQTS for $n = 8, 10$. As in the previous case, the cost function C_3 3.17 attains a better performance. The FIQTS method requires less iterations than the FHC method to reach high-quality solutions, so it is faster to locate local optima.

TABLE 4.3: Comparatives for the FIQTS method

n	CF	↑	↓	↔	Min	Max	$\bar{N}_n$	$\tilde{N}_n$	σ_{N_n}	$\bar{t}(s)$
8	C_1	0	3	0	108	112	110.16	110	1.448	3.8e-2
	C_2	2	1	0	112	116	114	114	0.7	1.1e+0
	C_3	**3**	**0**	**0**	**114**	**116**	**115.96**	**116**	**0.283**	1.4e-1
	C_4	1	2	0	108	116	112.6	112	1.629	3.7e-2
10	C_1	0	3	0	466	472	470.16	470	1.888	6.8e-1
	C_2	2	1	0	482	484	482.68	482	0.957	9.4e+1
	C_3	**3**	**0**	**0**	**480**	**484**	**483.64**	**484**	**1.12**	1.0e+1
	C_4	1	2	0	472	480	478.48	480	2.27	3.4e-1

Comparison Between FHC and FIQTS

According to the results in tables 4.2 and 4.3, the cost function C_3 3.17 is the most adequate cost function for our optimizers. Thus, the cost function C_3 3.17 is chosen to be used from now. Table 4.4 shows a comparison between FHC and FIQTS. Both proposals attain exactly the same score. Thus, the better is chosen as the method with higher mean nonlinearity ($\bar{N}_n$). The previous criterion help us to choose the best method when there is a draw in their scores (there are not statistically significant differences between them).

TABLE 4.4: Comparatives between the best results obtained for FIQTS and FHC

n	Method	↑	↓	↔	Min	Max	$\bar{N}_n$	$\tilde{N}_n$	σ_{N_n}	$\bar{t}(s)$
8	**FIQTS**	**0**	**0**	**1**	**114**	**116**	**115.96**	**116**	**0.283**	1.4e-1
	FHC	0	0	1	114	116	115.92	116	0.396	2.4e-1
10	**FIQTS**	**0**	**0**	**1**	**480**	**484**	**483.64**	**484**	**1.12**	1.0e+1
	FHC	0	0	1	480	484	483.16	484	1.621	1.6e+1

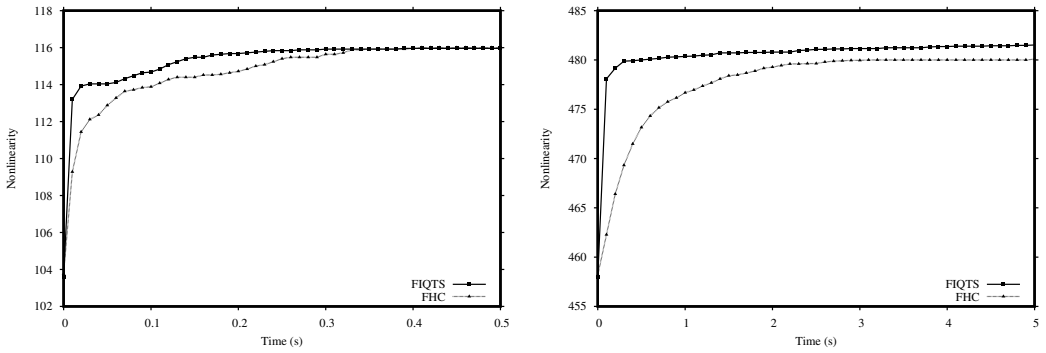
Moreover, it is noticeable that not only the FIQTS method is slightly better in terms of quality, but it also needs less time. In order to decide which local search method is the most suitable to use in the rest of experiments we compare the performance of the FHC and FIQTS methods with the cost function C_3 3.17 employing a fixed time of 0.5 and 5.0 seconds for n equal to 8 and 10 respectively. Table 4.5 shows the results obtained for 50 independent runs for each method.

Additionally, Figure 4.7 shows the evolution of the nonlinearity for the FHC and FIQTS methods with the cost function C_3 3.17. The FIQTS method quickly achieves high nonlinearity, so it is preferable for short-term executions.

TABLE 4.5: Comparison between FIQTS and FHC with the cost function C_3 in executions at fixed time

n	Method	↑	↓	↔	Min	Max	$\bar{N}_n$	$\tilde{N}_n$	σ_{N_n}
8	**FIQTS**	**0**	**0**	**1**	**114**	**116**	**115.96**	**116**	**0.283**
	FHC	0	0	1	114	116	115.92	116	0.396
10	**FIQTS**	**1**	**0**	**0**	**480**	**484**	**481.52**	**482**	**1.182**
	FHC	0	1	0	480	482	480.08	480	0.396

FIGURE 4.7: Nonlinearity evolution for FIQTS and FHC at fixed execution time



(A) 8-variables (B) 10-variables

4.3.3 Iterated Local Search

Given the previously presented results, the ILS method 19 considers the application of the FIQTS local search strategy. Since the 8-variable problem reaches the best-known result with a high-probability, we focus on the 10-variable problem in order to fix the percentage of perturbation parameter p_p. We carried out 50 independent runs independently for each parameter value (0.1%, 1%,2%,3% and 4%) and evolved 100 iterations of the method. We found that for all the different parameter values the results in nonlinearity are the same. Specifically, all of them reach a nonlinearity equal to 484 for $n = 10$. Table 4.6 shows the results obtained for the different parameter values. For $n = 8$ the ILS method found a nonlinearity equal to

TABLE 4.6: Comparison for different parameter values for ILS

p_p	↑	↓	↔	Min	Max	$\bar{N}_n$	$\tilde{N}_n$	σ_{N_n}
0.001	0	0	4	484	484	484	484	0
0.01	0	0	4	484	484	484	484	0
0.02	0	0	4	484	484	484	484	0
0.03	0	0	4	484	484	484	484	0
0.04	0	0	4	484	484	484	484	0

116 in all the independent runs.

4.3.4 Simulated Annealing

In the SA method 20 we use the paradigm of FIQTS in the iterations inside the loop at fixed temperature. For this algorithm there are two parameters to fix: the probability of accept uphill movements at the beginning of the execution P_{0_u} and the probability of accept uphill movements at the end of the execution P_{F_u}. After several tests it could be concluded that the probability to accept worsening individuals should be quite small, because as we can see in figures 4.6, when a solution is close to be a local optimum the probability that a neighbor has worst cost (in many cases just slightly) than the solution it self, increases a lot. Thus, many of the neighbors are considered to be good in order to scape from local optimums and the space is not properly intensified. However, a too low value does not provide improvement either because a too low value converts the SA into a simple local search. The parameters P_{0_u} and P_{F_0} were tested with three different values, each one using 100 outer iterations and the results obtained are shown in table 4.7.

TABLE 4.7: Comparison for the parameters values of SA

P_{0_u}	P_{F_u}	$\uparrow$	$\downarrow$	$\leftrightarrow$	Min	Max	$\bar{N}_n$	$\tilde{N}_n$	σ_{N_n}
0.1	1e-5	4	1	3	480	484	482.24	482	1.492
0.1	1e-7	4	2	2	480	484	482.2	482	1.229
0.1	1e-9	6	1	1	480	484	482.72	482	1.325
0.01	1e-5	0	7	1	480	480	480	480	0
0.01	1e-7	2	4	2	480	484	481.24	482	1.271
0.01	1e-9	3	2	3	480	484	481.76	482	1.379
0.001	1e-5	0	7	1	480	480	480	480	0
0.001	**1e-7**	**8**	**0**	**0**	**484**	**484**	**484**	**484**	**0**
0.001	1e-9	2	5	1	480	484	481.12	480	1.288

We can see that the values reported in order to attain promising values with SA are quite small when compared to usual values.

4.3.5 Comparison Between Trajectory Search Methods

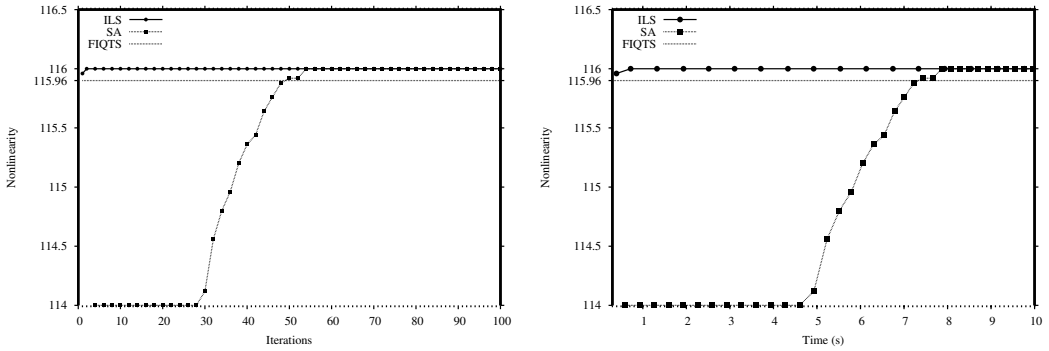
Since the FIQTS method shows the best performance among FHC and FIQTS, it is compared against ILS and SA. Table 4.8 shows the comparison between FIQTS, ILS and SA. It is noticeable that the methods ILS and SA reach the same results, but the SA has a lower execution time. However, this is somewhat misleading because the ILS method reaches these values in a shorter time of execution. To illustrate this, Figures 4.8 and 4.9 shows the evolution of the nonlinearity when compared in terms of time and iterations.

We can see that the ILS method quickly reaches high-quality solutions and that the SA takes more time to reach to such solutions. When compared

TABLE 4.8: Comparatives between the best results obtained for FIQTS, ILS and SA

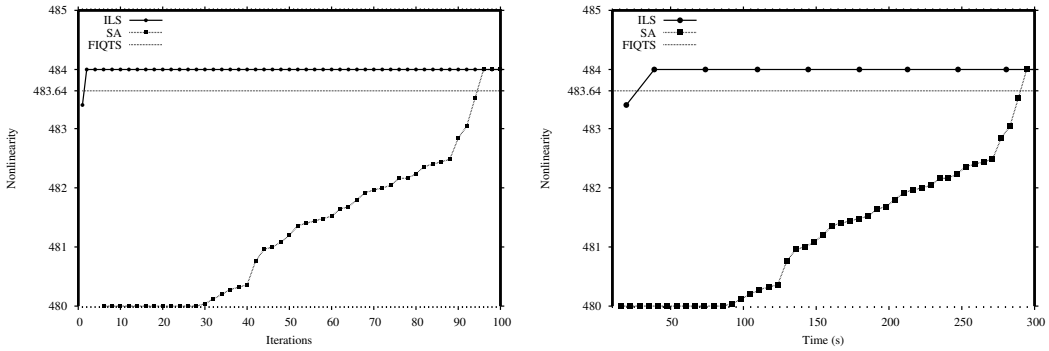
n	Method	↑	↓	↔	Min	Max	$\bar{N}_n$	$\tilde{N}_n$	σ_{N_n}	$\bar{t}(s)$
8	FIQTS	0	2	0	114	116	115.96	116	0.283	1.4e-1
	ILS	1	0	1	116	116	116	116	0	2.9e+1
	SA	1	0	1	116	116	116	116	0	1.3e+1
10	FIQTS	0	2	0	480	484	483.64	484	1.12	1.0e+1
	ILS	1	0	1	484	484	484	484	0	1.6e+3
	SA	1	0	1	484	484	484	484	0	3.0e+2

FIGURE 4.8: Comparison between ILS and SA for $n = 8$



(A) 8-variables iterations (B) 8-variables time

FIGURE 4.9: Comparison between ILS and SA for $n = 10$



(A) 10-variables iterations (B) 10-variables time

to FIQTS, we see that both methods require much more execution time, so these methods are not so suitable to be used in the MAC-REDDCC. After all these experiments we conclude that the best method to be used in the MAC-REDDCC 30 is the FIQTS 18 whit the cost function C_3 3.17.

4.4 Evolutionary-based Metaheuristics

4.4.1 Memetic Algorithm with Clusters

This section focuses on the main algorithmic contribution of this thesis, which is the MAC-REDDCC method 30. One of the inconveniences of the method is that several parameters must be set. Particularly these are the following parameters:

1. **Initial minimum DCNC** D_{C_0}: it avoids that individuals in the same cluster are too close to each other *i.e.* it maintains a proper diversity in each cluster (see algorithm 30 line 14).
2. **Initial minimum DCN** D_0: it is responsible for controlling the degree of diversity maintained in the whole population (see algorithm 30 line 13).
3. **Cluster size** C_{size}: indicates the maximum size allowed for each cluster (see algorithm 27 line 12).
4. **Population size** N: number of individuals in the population and number of offspring generated in each generation (see algorithm 24 line 1).
5. **Local search time** T_{ls}: stopping criterion of the local search procedure executed for every individual belonging to the offsprings (see algorithm 29 line 5).
6. **Local search factor** K_{ls}: if the offspring is generated with parents belonging to different clusters, the local search time applied to this kind of individuals is $T_{ls} \times K_{ls}$ (see algorithm 29 line 3).
7. **Cross clusters probability** p_{cc}: indicates the probability to cross individuals belonging to different cluster (see algorithm 28 line 2).
8. **Mutation probability** p_m: probability of performing swaps to mutate the individual (see algorithm 23 line 2).
9. **Crossover probability** p_c: probability of interchanging each gene (see algorithm 22 line 9).
10. **Exploration factor** K_E: indicates the percentage of exploration explicitly induced to the algorithm taking into account the distances D_{C_0} and D_0 (see algorithm 30 line 6).

Each parameter was fixed by testing 5 different values and for each parameter value, 50 independent runs were executed. For each independent run, the stopping criterion was set to 72 hours of execution. Due to the limitations of time it is not possible to do an exhaustive search in the parameters space by considering the dependences between all of them. In fact if we consider all the possible combinations for 5 different values to each parameter, it would require to do $5^{10} = 9765625$ different experiments. Considering 50 independent runs for each experiment, it would require to

do approximately 5×10^8 independent runs, each one for 72 hours which gives approximately 3.5×10^{10} execution hours, which is approximately 4×10^6 years. Thus, since it is not plausible to do the parameter setting in this way, the optimization of each parameter was performed independently from each other.

Study of the Parameters Influence

As mentioned, instead of dealing with all the dependencies between the parameters we simply consider each one as independent of each other. First, we sorted the parameters considering their importance. This was done in base of some preliminary experiments, but some subjective decisions were required. Once the parameters are ranked according to its importance, we proceed to test different values of the first parameter and once that the first parameter is set to a fixed value, we continue by setting the second parameter and so on taking into account the previous parameters setting. Note that for each parameter an initial value is given, which is used when the corresponding parameter has not been optimized.

Parameters ranking

As the novel proposal of the thesis is the clustering technique in the replacement phase, we decide to initially set the initial minimum DCNC $\mathbf{D_{C_0}}$, deciding that this should be the *first* parameter of study. Very related with $\mathbf{D_{C_0}}$ is the $\mathbf{D_0}$ parameter. Thus, $\mathbf{D_0}$ was established to be the *second* parameter of study. Since the amount of diversity is also related to the cluster size parameter $\mathbf{C_{size}}$, it was set to be the *third* parameter of study. For the same reason the population size parameter $\mathbf{N}$ was set to be the *fourth* parameter of study. Once the population size has been considered, we should consider the execution time that should be spend by the algorithm in the local search. The number of generations that the algorithm perform will depend on the local search time fixed to each individual and on the population size. For this reasons the local search time $\mathbf{T_{ls}}$ was set to be the *fifth* parameter of study. The factor of local search $\mathbf{K_{ls}}$ is strictly related with the local search time, so this parameter was set to be the *sixth* parameter of study. Since the local search factor is used when an offspring is generated with parents belonging to a different cluster, we set the cross cluster probability $\mathbf{p_{cc}}$ to be the *seventh* parameter of study. Then, other parameters related to mating were set. The mutation probability $\mathbf{p_m}$ was set as the *eighth* parameter of study. The last parameter linked with the mating is the crossover probability $\mathbf{p_c}$ which is set to be the *ninth* parameter of study. Finally, we proceed to analyze the exploration factor $\mathbf{K_E}$ parameter, which is set to be the *tenth* and last parameter of study.

Zero parameters setting

Since the problem for 8 variables can be solved easily up to the best results known for nonlinearity, we focus the parameters setting for the 10 variable problem. Taking into account some preliminary results, the zero or initial parameter setting Γ_0 was set according to Table 4.9.

TABLE 4.9: Zero parameters setting Γ_0

D_{C_0}	D_0	C_{size}	N	T_{ls}	K_{ls}	p_{cc}	p_m	p_c	K_E
20	100	10	200	0.01	1.0	0.2	0.002	0.5	1.0

Firstly, the D_{C_0} parameter was set in such way that individuals in the same cluster, are different in at least the 2% of their truth table positions. The truth table has a length of 2^n, as $n = 10$, we have

$$D_{C_0} = \lfloor 0.02 \times 2^{10} \rfloor = 20. \tag{4.1}$$

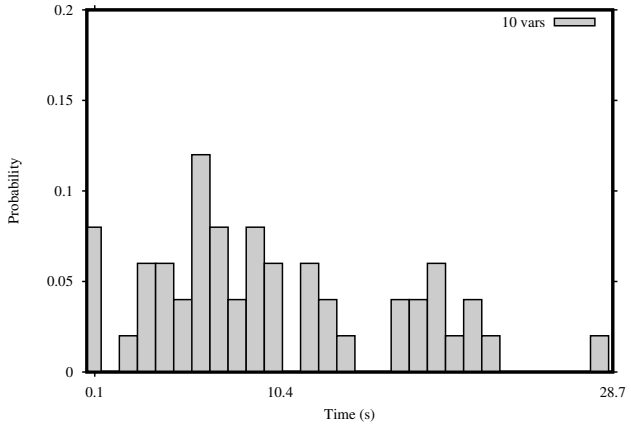
For the D_0 parameter, it was considered that individuals belonging to separated clusters should be different in approximately at least the 10% of their truth table positions. In our case we choose that the individuals in different cluster were differ at least in the 9.8%, in this way we have

$$D_0 = \lfloor 0.098 \times 2^{10} \rfloor = 100. \tag{4.2}$$

Preliminary results showed that even a small population with ten individuals can reach promising results. Based on this, we decide to set the C_{size} parameter equal to 10 and the population size N equal to 200, so that several clusters are maintained in the population. After considering the information given in table 4.3 and figure 4.7, it is noticeable that the FIQTS method quickly improves the nonlinearity of random solutions. Thus, we concluded that a low execution time for the local search refinement could be suitable. The question is how short should be T_{ls}. Figure 4.10 shows the distribution of the time required by FIQTS method to reach a local optimum for 10 variables. We can see that the method works pretty well for short time executions, and is even able to reach local optimums in 0.1 seconds. In order to evolve enough generations we set initial the local search time as the 10% of this value (0.1×0.1). For the local search factor K_{ls} we simply decided to set K_{ls} equal to 1. In this way all the individuals will have the same local search time and the parametrization strategy would change this value if required.

In order to have a proper intensification in each cluster, we decided to set the cross cluster probability p_{cc} equal to 20%, which implies that individuals belonging to the same cluster will be mated with more probability that individuals belonging to different clusters. The mutation probability p_m was set in such a way that one exchange of a 0 position with a 1 position is

FIGURE 4.10: FIQTS method execution time distribution.



carried out on average, so we have

$$p_m = \frac{2}{2^{10}} \approx 0.002. \tag{4.3}$$

The crossover probability p_c was set in such a way that the offspring generated share the same information amount of their parents, *i.e.* equal to 50%. Finally, the exploration factor was set equal to 1, which indicates that the algorithm will be exploring the search space until the end of the execution.

Preliminary results for Γ_0. With Γ_0 we carried out 50 independent runs with the stopping criterion set equal to 72 hours. Table 4.10 shows the results obtained for this experiment.

TABLE 4.10: Results for Γ_0

n	Min	Max	$\bar{N}_n$	$\tilde{N}_n$	σ_{N_n}	S_r
10	484	488	486.96	487	1.087	50%

We can see from the results showed in table 4.10, that using the zero parameters setting, the algorithm achieves the best results known for metaheuristics in the 10-variable problem. The mean nonlinearity is high ($\bar{N}_n = 486.96$), which indicates that in several of the independent executions it was able to find solutions with nonlinearity equal to 488. The seventh column shows the percentage of independent executions that reach this nonlinearity value, which is called the success ratio (S_r).

The mean nonlinearity will be our main indicator to consider the quality of each parametrization Additionally, the success ratio S_r help us to know what percentage of the independent runs achieve a nonlinearity equal to 488. The initial parameter setting Γ_0 has a success ratio $S_r = 50\%$, which means that 25 of the 50 independent runs achieve solutions with nonlinearity equal to

488. Since 488 is the maximum nonlinearity that had been obtained ever in a single execution of a metaheuristic, our aim was to find a parameterization with a success ratio close to 100%.

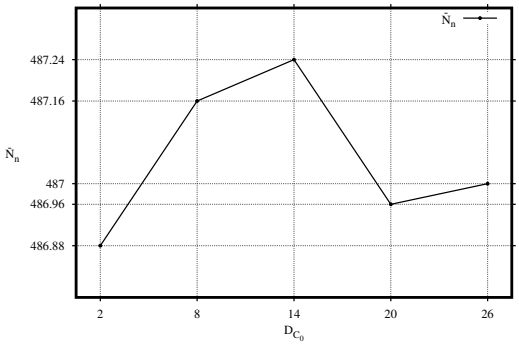
First parameters setting: initial minimum DCNC

The first parameters setting Γ_1 was carried out with the values 2,8,14,20 and 26 for the D_{C_0} parameter. Each of these values was taken with 6 units of distance and considering that the minimum distance that could have two different solutions (2). Table 4.11 shows the summary of the results obtained after 72 execution hours for the independent runs of each parameter value.

TABLE 4.11: Comparison for different D_{C_0} values

D_{C_0}	$\uparrow$	$\downarrow$	$\leftrightarrow$	Min	Max	$\bar{N}_n$	$\tilde{N}_n$	σ_{N_n}
2	0	0	4	484	488	486.88	488	1.288
8	0	0	4	486	488	487.16	488	0.997
14	**0**	**0**	**4**	**486**	**488**	**487.24**	**488**	**0.981**
20	0	0	4	484	488	486.96	487	1.087
26	0	0	4	484	488	487	488	1.088

FIGURE 4.11: Mean nonlinearity for different D_{C_0} values



We see that all the parameters are tied in their scores, which indicates that all the values could be suitable to set the parameter value. However if we look at the mean nonlinearity, we see that the parameter value 14 has a slightly better mean nonlinearity. This is more evident if we look at the figure 4.11 which shows the mean nonlinearity values achieved by the algorithm at this step. With the results obtained we can conclude that $D_{C_0} = 14$ is the most suitable value for the initial minimum DCNC. The success ratio S_r with this parameter value is 62%, which is a better success ratio than the previous. Table 4.12 shows the parameters setting, the mean nonlinearity and the success ratio obtained after the first adjustment.

TABLE 4.12: First parameters setting Γ_1

D_{C_0}	D_0	C_{size}	N	T_{ls}	K_{ls}	p_{cc}	p_m	p_c	K_E	$\bar{N}_n$	S_r
14	100	10	200	0.01	1.0	0.2	0.002	0.5	1.0	487.24	62%

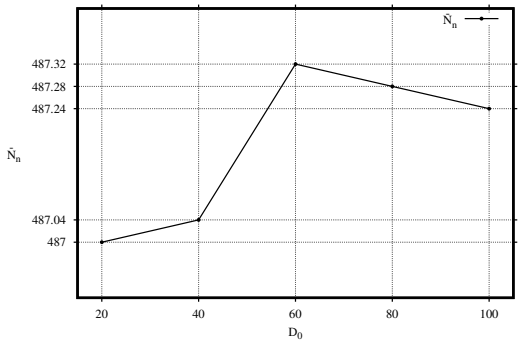
Second parameters setting: initial minimum DCN

The second parameters setting Γ_2 was carried out with the values 20,40,60,80 and 100 for the D_0 parameter. Table 4.13 shows the summary of the results obtained after 72 execution hours for the independent runs of each parameter value. We see that the parameters values are tied in their

TABLE 4.13: Comparison between different D_0 values

D_0	$\uparrow$	$\downarrow$	$\leftrightarrow$	Min	Max	$\bar{N}_n$	$\tilde{N}_n$	σ_{N_n}
20	0	0	4	486	488	487	487	1.01
40	0	0	4	486	488	487.04	488	1.009
60	**0**	**0**	**4**	**486**	**488**	**487.32**	**488**	**0.957**
80	0	0	4	486	488	487.28	488	0.97
100	0	0	4	486	488	487.24	488	0.981

FIGURE 4.12: Mean nonlinearity for different D_0 values



scores again. By taking into account the information from the figure 4.12, we realize that $D_0 = 60$ is the most suitable value for the initial minimum DCN. The corresponding success ratio S_r for D_0 equal to 60 is 66%. Table 4.14 shows the parameter setting, the mean nonlinearity and the success ratio obtained after the second adjustment.

TABLE 4.14: Second parameters setting Γ_2

D_{C_0}	D_0	C_{size}	N	T_{ls}	K_{ls}	p_{cc}	p_m	p_c	K_E	$\bar{N}_n$	S_r
14	**60**	10	200	0.01	1.0	0.2	0.002	0.5	1.0	487.32	66%

Third parameters setting: cluster size

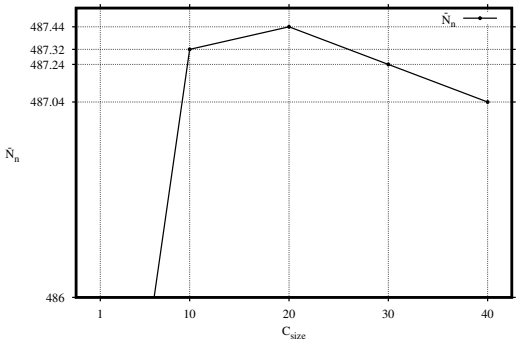
The third parameter setting Γ_3 was carried out with the values 1, 10, 20, 30 and 40 for the C_{size} parameter. Table 4.15 shows the summary of the results obtained after 72 execution hours for the independent runs of each parameter value. From the results we can see that a cluster size equal to 1

TABLE 4.15: Comparison between different C_{size} values

C_{size}	$\uparrow$	$\downarrow$	$\leftrightarrow$	Min	Max	$\bar{N}_n$	$\tilde{N}_n$	σ_{N_n}
1	0	4	0	484	484	484	484	0
10	1	0	3	486	488	487.32	488	0.957
20	**2**	**0**	**2**	**486**	**488**	**487.44**	**488**	**0.907**
30	1	0	3	486	488	487.24	488	0.981
40	1	1	2	486	488	487.04	488	1.009

has a really bad performance when comparing with other cluster size values, meaning that considering clusters is really useful. In this case the results obtained indicate that the most suitable value for C_{size} is 20. Figure 4.13 shows the mean nonlinearity obtained with all the tested values. The

FIGURE 4.13: Mean nonlinearity for different C_{size} values



corresponding success ratio S_r for C_{size} equal to 20 is 72% and table 4.16 shows the corresponding parameter setting, the mean nonlinearity and the success ratio obtained after the third adjustment.

TABLE 4.16: Third parameters setting Γ_3

D_{C_0}	D_0	C_{size}	N	T_{ls}	K_{ls}	p_{cc}	p_m	p_c	K_E	$\bar{N}_n$	S_r
14	**60**	**20**	200	0.01	1.0	0.2	0.002	0.5	1.0	487.44	72%

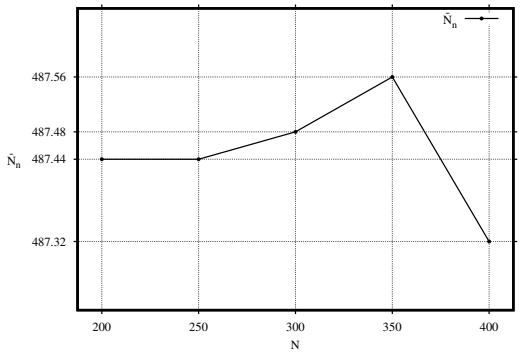
Fourth parameters setting: population size

The values chosen for the fourth parameter setting were: 200, 250, 300, 350 and 400 for the N parameter. Table 4.17 shows the summary of the results obtained after 72 execution hours for the independent runs of each parameter value and figure 4.14 shows the mean nonlinearity obtained for

TABLE 4.17: Comparison between different N values

N	$\uparrow$	$\downarrow$	$\leftrightarrow$	Min	Max	$\bar{N}_n$	$\tilde{N}_n$	σ_{N_n}
200	0	0	4	486	488	487.44	488	0.907
250	0	0	4	486	488	487.44	488	0.907
300	0	0	4	486	488	487.48	488	0.886
350	**0**	**0**	**4**	**486**	**488**	**487.56**	**488**	**0.837**
400	0	0	4	486	488	487.32	488	0.957

FIGURE 4.14: Mean nonlinearity for different N values



the parameter values tested. We see again that the parameter values are tied, but taking into account the information given by the figure we can conclude that the most suitable values is 350. The corresponding success ratio S_r for N equal to 350 is 78%. Table 4.18 shows the parameter setting, the mean nonlinearity and the success ratio obtained after the fourth adjustment.

TABLE 4.18: Fourth parameters setting Γ_4

D_{C_0}	D_0	C_{size}	N	T_{ls}	K_{ls}	p_{cc}	p_m	p_c	K_E	$\bar{N}_n$	S_r
14	**60**	**20**	**350**	0.01	1.0	0.2	0.002	0.5	1.0	487.56	78%

Fifth parameters setting: local search time

The fifth parameters setting Γ_5 was carried out with the values 0.01,0.02,0.04,0.08 and 0.16 for the T_{ls} parameter. Each of these values was generated by multiplying the previous by two. Small values were chosen because preliminary experiments had shown the importance of evolving many generations. Table 4.19 shows the summary of the results obtained after 72 execution hours for the independent runs of each parameter value.

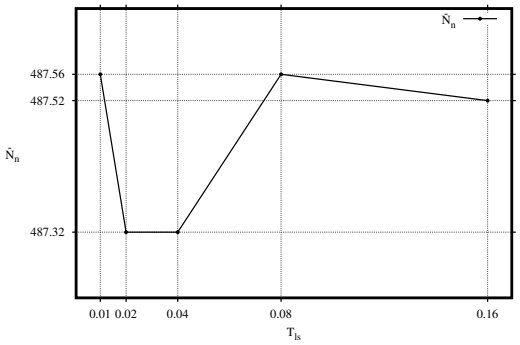
In this case, the first full draw is presented, as we can see in Table 4.19, the parameter values 0.01 and 0.08 have the same score and mean nonlinearity. In order to decide which value is more suitable to use in the algorithm, we focus on the figure 4.15 and according to the slopes of the lines that connect

TABLE 4.19: Comparison between different T_{ls} values

T_{ls}	$\uparrow$	$\downarrow$	$\leftrightarrow$	Min	Max	$\bar{N}_n$	$\tilde{N}_n$	σ_{N_n}
0.01	0	0	4	486	488	487.56	488	0.837
0.02	0	0	4	486	488	487.32	488	0.957
0.04	0	0	4	486	488	487.32	488	0.957
0.08	**0**	**0**	**4**	**486**	**488**	**487.56**	**488**	**0.837**
0.16	0	0	4	486	488	487.52	488	0.863

these points with their neighbors we conclude that the most suitable value is 0.08 for T_{ls}. This is because the next parameter value 0.16 also show a high mean nonlinearity. However, the next value to 0.01 (0.02), has a lower mean nonlinearity. In this way we can conclude that the parameter value T_{ls} equal to 0.08 is more stable.

FIGURE 4.15: Mean nonlinearity for different T_{ls} values



The corresponding success ratio S_r for T_{ls} equal to 0.08 is exactly the same as in the previous parameters setting Γ_4 78%. Table 4.20 shows the parameter setting, the mean nonlinearity and the success ratio obtained after the fifth adjustment.

TABLE 4.20: Fifth parameters setting Γ_5

D_{C_0}	D_0	C_{size}	N	T_{ls}	K_{ls}	p_{cc}	p_m	p_c	K_E	$\bar{N}_n$	S_r
14	**60**	**20**	**350**	**0.08**	1.0	0.2	0.002	0.5	1.0	487.56	78%

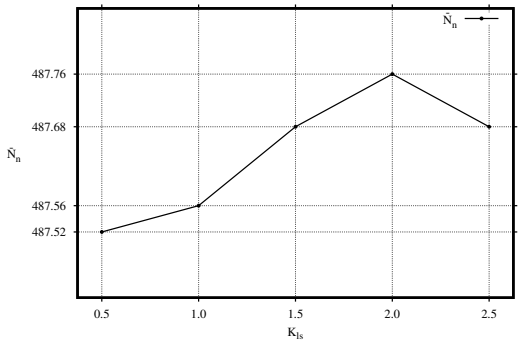
Sixth parameters setting: local search factor

The sixth parameters setting Γ_6 was carried out with the values 0.5,1.0,1.5,2.0 and 2.5 for the K_{ls} parameter. Table 4.21 shows the summary of the results obtained after 72 execution hours for the independent runs of each parameter value. We see that the K_{ls} value with highest mean nonlinearity respect to the other values — see figure 4.16 — is 2.0, which indicates that a local search time equal to 0.16 will be performed to all these offspring generated with parents belonging to different clusters.

TABLE 4.21: Comparison between different K_{ls} values

K_{ls}	$\uparrow$	$\downarrow$	$\leftrightarrow$	Min	Max	$\bar{N}_n$	$\tilde{N}_n$	σ_{N_n}
0.5	0	0	4	486	488	487.52	488	0.863
1.0	0	0	4	486	488	487.56	488	0.837
1.5	0	0	4	486	488	487.68	488	0.741
2.0	**0**	**0**	**4**	**486**	**488**	**487.76**	**488**	**0.657**
2.5	0	0	4	486	488	487.68	488	0.741

FIGURE 4.16: Mean nonlinearity for different K_{ls} values



The corresponding success ratio S_r for K_{ls} equal to 2.0 is 88%. Table 4.22 shows the parameter setting, the mean nonlinearity and the success ratio obtained after the sixth adjustment.

TABLE 4.22: Sixth parameters setting Γ_6

D_{C_0}	D_0	C_{size}	N	T_{ls}	K_{ls}	p_{cc}	p_m	p_c	K_E	$\bar{N}_n$	S_r
14	**60**	**20**	**350**	**0.08**	**2.0**	0.2	0.002	0.5	1.0	487.76	88%

Seventh parameters setting: cross cluster probability

The seventh parameter setting Γ_7 was carried out with the values 0.0,0.1,0.2,0.3 and 0.4. Table 4.23 shows the summary of the results obtained after 72 execution hours for the independent runs of each parameter value. We should notice that p_{cc} has an influence on the degree between exploitation and exploration in the algorithm: a p_{cc} value of 0.0 indicates that the mating will be always with individuals in the same cluster, which reduces the exploration in the search space and increases the exploitation in each cluster. We see that p_{cc} equal to 20% is the most suitable parameter value. Figure 4.17 shows the mean nonlinearity for these p_{cc} values.

The corresponding success ratio S_r for p_{cc} equal to 0.2 is exactly the same as in the previous parameters setting Γ_6 88%. This is because there was no change in the p_{cc} value, i.e. the initial value was a proper one. Table 4.24 shows the parameter setting, the mean nonlinearity and the success ratio obtained after the seventh adjustment.

TABLE 4.23: Comparison between different p_{cc} values

p_{cc}	$\uparrow$	$\downarrow$	$\leftrightarrow$	Min	Max	$\bar{N}_n$	$\tilde{N}_n$	σ_{N_n}
0.0	0	0	4	486	488	487.48	488	0.886
0.1	0	0	4	486	488	487.52	488	0.863
0.2	**1**	**0**	**3**	**486**	**488**	**487.76**	**488**	**0.657**
0.3	0	0	4	486	488	487.48	488	0.886
0.4	0	1	3	486	488	487.16	488	0.997

FIGURE 4.17: Mean nonlinearity for different p_{cc} values

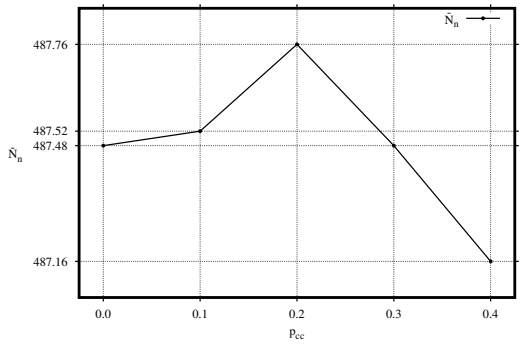


TABLE 4.24: Seventh parameters setting Γ_7

D_{C_0}	D_0	C_{size}	N	T_{ls}	K_{ls}	p_{cc}	p_m	p_c	K_E
14	**60**	**20**	**350**	**0.08**	**2.0**	**0.2**	0.002	0.5	1.0

$\bar{N}_n$	S_r
487.76	88%

Eighth parameters setting: mutation probability

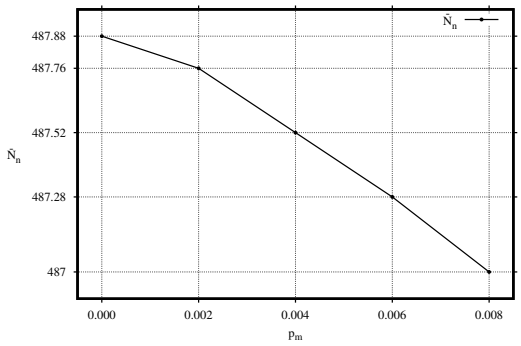
The eighth parameter setting Γ_8 was carried out with the values 0.000,0.002,0.004,0.006 and 0.008 for the mutation probability. These values were chosen so that the mutation method can perform from zero to four swaps between 0's and 1's positions in average. Table 4.25 shows the summary of the results obtained after 72 execution hours for the independent runs of each parameter value. We see that a mutation

TABLE 4.25: Comparison between different p_m values

p_m	$\uparrow$	$\downarrow$	$\leftrightarrow$	Min	Max	$\bar{N}_n$	$\tilde{N}_n$	σ_{N_n}
0.000	**3**	**0**	**1**	**486**	**488**	**487.88**	**488**	**0.48**
0.002	2	0	2	486	488	487.76	488	0.657
0.004	1	1	2	486	488	487.52	488	0.863
0.006	0	2	2	486	488	487.28	488	0.97
0.008	0	3	1	486	488	487	487	1.01

probability equal to zero is the most suitable value for p_m, which indicates that no mutation will be allowed in the algorithm. Figure 4.18 shows the impact of this parameter on the mean nonlinearity. Note that many memetic algorithms do not use mutation, because in some sense, the local search operator acts as a mutation.

FIGURE 4.18: Mean nonlinearity for different p_m values



The corresponding success ratio S_r for p_m equal to 0 is 94%. Table 4.26 shows the parameter setting, the mean nonlinearity and the success ratio obtained after the eighth adjustment.

TABLE 4.26: Eighth parameters setting Γ_8

D_{C_0}	D_0	C_{size}	N	T_{ls}	K_{ls}	p_{cc}	p_m	p_c	K_E	$\bar{N}_n$	S_r
14	**60**	**20**	**350**	**0.08**	**2.0**	**0.2**	**0**	0.5	1.0	487.88	94%

Ninth parameters setting: cross probability

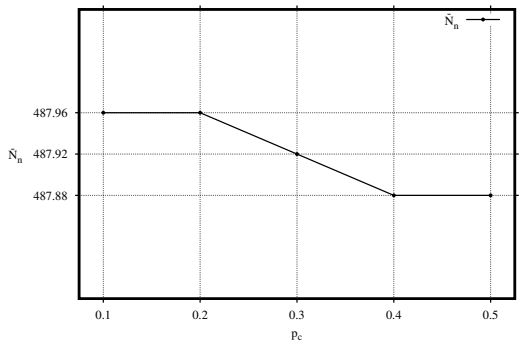
The ninth parameter setting Γ_9 was carried out with the values 0.1,0.2,0.3,0.4 and 0.5 for the cross probability. Table 4.27 shows the summary of the results obtained after 72 execution hours for the independent runs of each parameter value and figure 4.19 shows the mean nonlinearity obtained for these values.

TABLE 4.27: Comparison between different p_c values

p_c	$\uparrow$	$\downarrow$	$\leftrightarrow$	Min	Max	$\bar{N}_n$	$\tilde{N}_n$	σ_{N_n}
0.1	0	0	4	486	488	487.96	488	0.283
0.2	**0**	**0**	**4**	**486**	**488**	**487.96**	**488**	**0.283**
0.3	0	0	4	486	488	487.92	488	0.396
0.4	0	0	4	486	488	487.88	488	0.48
0.5	0	0	4	486	488	487.88	488	0.48

In this case we have another full draw between two parameter values, but now there are not other values between them that help us to decide which is better. We should consider that a p_c value equal to 0.0 makes no sense, since the offspring generated would simply be clones of their parents. Keeping this in mind, we decided that the most suitable value for p_c is 0.2, which allows the algorithm to have a little bit more of exploration.

FIGURE 4.19: Mean nonlinearity for different p_c values



The corresponding success ratio S_r for p_c equal to 0.2 is 98%. Table 4.28 shows the parameter setting, the mean nonlinearity and the success ratio obtained after the ninth adjustment.

TABLE 4.28: Ninth parameters setting Γ_9

D_{C_0}	D_0	C_{size}	N	T_{ls}	K_{ls}	p_{cc}	p_m	p_c	K_E	$\bar{N}_n$	S_r
14	**60**	**20**	**350**	**0.08**	**2.0**	**0.2**	**0**	**0.2**	1.0	487.96	98%

We should notice that p_c is also responsible in some sense for the balance between exploitation and exploration in the algorithm. A p_c value equal to 0.5 indicates that the offspring generated will have the same amount of information from both parents, if the parents belong to different clusters is highly probably that the offspring will be generated in a different region of the search space, far from both parents.

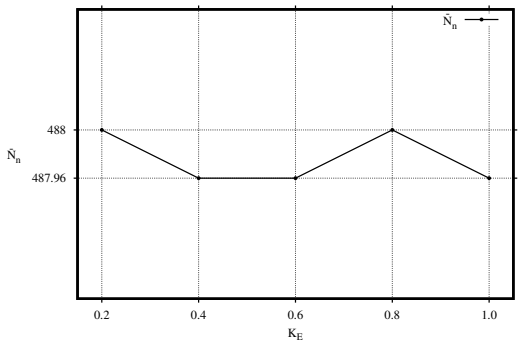
Tenth parameters setting: exploration factor

The tenth parameter setting Γ_{10} was carried out with the values 0.2,0.4,0.6,0.8 and 1.0. Table 4.29 shows the summary of the results obtained after 72 execution hours for the independent runs of each parameter value and figure 4.20 shows the mean nonlinearity obtained for these values.

TABLE 4.29: Comparison between different K_E values

K_E	$\uparrow$	$\downarrow$	$\leftrightarrow$	Min	Max	$\bar{N}_n$	$\tilde{N}_n$	σ_{N_n}
0.2	**0**	**0**	**4**	**488**	**488**	**488**	**488**	**0**
0.4	0	0	4	486	488	487.96	488	0.283
0.6	0	0	4	486	488	487.96	488	0.283
0.8	0	0	4	488	488	488	488	0
1.0	0	0	4	486	488	487.96	488	0.283

FIGURE 4.20: Mean nonlinearity for different K_E values



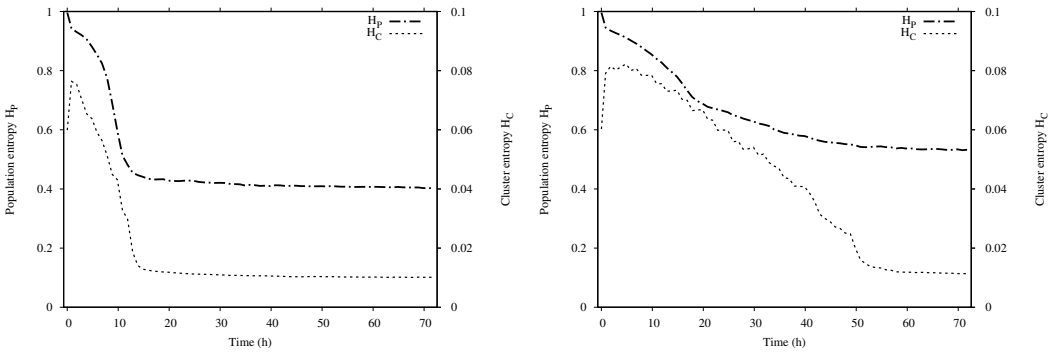
In this case we have another full draw between the K_E values 0.2 and 0.8. We see in the figure 4.20 that they appear to have the same stability, and the choice of the most suitable value for K_E turns complicated.

TABLE 4.30: Comparison between different K_E values for the amount of individuals found with nonlinearity equal to 488

K_E	Min	Max	Mean	Median
0.2	**1**	**9**	**4.52**	**4**
0.8	1	9	4.06	4

At this time, when it is achieved the 100% of success ratio for solutions with nonlinearity of 488, we can figure out how many different individuals with nonlinearity equal to 488 are found in average for each parameter value. Table 4.30 shows the results obtained for K_E equal to 0.2 and 0.8 concerning to the amount of individuals with nonlinearity equal to 488 that are found. We can see that the results are almost equal again, but the mean of solutions found is slightly better for $K_E = 0.2$. we might use $K_E = 0.8$ by considering that a small exploration factor could cause low diversity. However, this does not occurs. This is clear if we look at the figures 4.21. Figures 4.21 shows the

FIGURE 4.21: Population entropy and cluster entropy for K_E equal to 0.2 and 0.8



(A) Entropies for $K_E = 0.2$

(B) Entropies for $K_E = 0.8$

entropies obtained with both parameter values. The entropy is calculated for the whole population and for the clusters. Note that an entropy value close to 0 indicates that the individuals are almost equal. From the figure 4.21a we conclude that if we choose to set K_E equal to 0.2, the diversity of the population during the execution will not be reduced drastically and in the clusters will never be zero because it is no allowed that in the same cluster there are identical individuals. Finally, we decided to use the 0.2 value.

Table 4.31 shows the parameter setting, the mean nonlinearity and the success ratio obtained after the tenth adjustment.

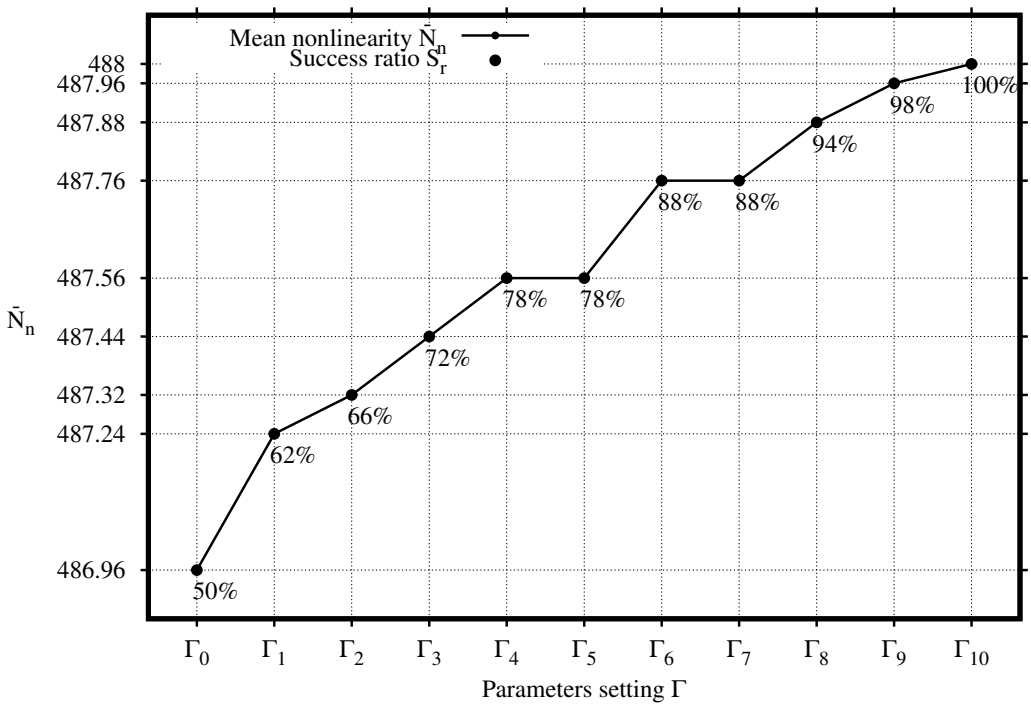
TABLE 4.31: Tenth parameters setting Γ_{10}

D_{C_0}	D_0	C_{size}	N	T_{ls}	K_{ls}	p_{cc}	p_m	p_c	K_E	$\bar{N}_n$	S_r
14	**60**	**20**	**350**	**0.08**	**2.0**	**0.2**	**0**	**0.2**	**0.2**	**488**	**100%**

Progress of the parameters setting

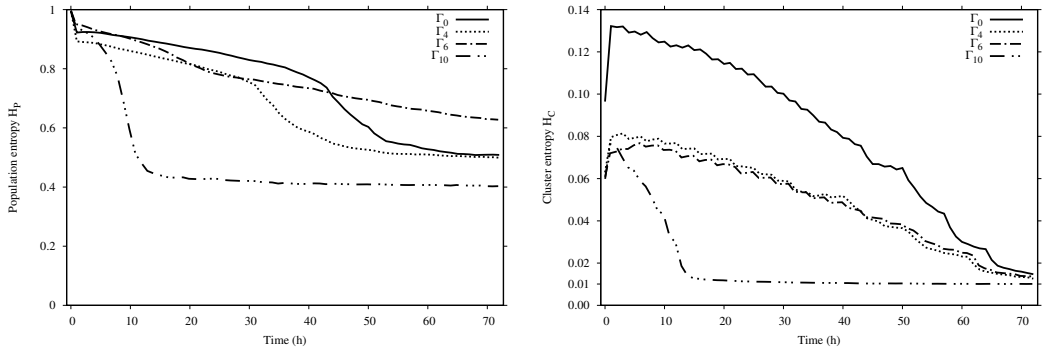
Once all the parameters have been set, we can see that the mean nonlinearity and the success ratio improves for the parameter settings process. Figure 4.22 shows the mean nonlinearity for different parameter settings.

FIGURE 4.22: Mean nonlinearity for different parameter settings



It is interesting to see the behavior of the diversity of the population and the diversity in the clusters for different parameter settings. Figures 4.23 shows the curves that describes the population entropy and cluster entropy for the configuration of parameters zero, four, six and ten.

FIGURE 4.23: Population entropy and cluster entropy for different parameter settings



(A) Population entropy (B) Cluster entropy

From the figure 4.23a we can see that the population never converges to a single region in the search space. This is due to the incorporation of the clustering technique, which has the population divided in many regions of the search space. Conversely, from the figure 4.23b we can see that the individuals in each cluster converges to a different regions in the search space. The entropy reduction behaves as it was expected, it is reduced as the evolution progresses. We can see that the diversity behavior for the last parameter setting K_E (Γ_{10}) is very different to the others. This is due to the exploration factor that has been set equal to 0.2, which is approximately 14.4 hours. This is the reason behind the particular decrease of the entropy. Again, the clusters never admit clone individuals. This is why the entropy at the end of the execution is never 0 for the clusters, because all the individuals need to be a little different.

Comparison Among Population-based Metaheuristics

MAC-REDDCC is an extension of the LMA-REDDC which in turns is an extension of LMA-RGE. All these algorithms were executed using the set of parameters obtained Γ_{10}. Specifically, each of the three methods were executed 50 independent times both with 8 and 10 variables in order to properly validate our proposal when compared to the simplest variants LMA-REDDC and LMA-RGE.

The parameterization of each method is shown in table 4.32. MAC-REDDCC incorporates a clustering technique for the replacement phase, LMA-REDDC incorporates an elitist diversity control for the replacement phase and LMA-RGE simply incorporates a replacement generational with elitism based on the offspring population plus the elite individual.

TABLE 4.32: Parameters involved in MAC-REDDCC, LMA-REDDC and LMA-RGE

Method	D_{C_0}	D_0	C_{size}	N	T_{ls}	K_{ls}	p_{cc}	p_m	p_c	K_E
MAC-REDDCC	**14**	**60**	**20**	**350**	**0.08**	**2.0**	**0.2**	**0**	**0.2**	**0.2**
LMA-REDDC	×	**60**	×	**350**	**0.08**	×	×	**0**	**0.2**	**0.2**
LMA-RGE	×	×	×	**350**	**0.08**	×	×	**0**	**0.2**	×

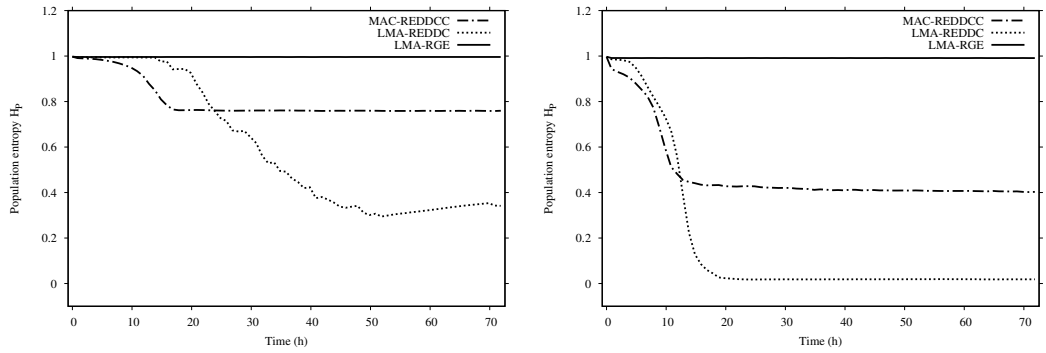
The results obtained for LMA-REDDC and LMA-RGE are shown in the table 4.33 next to the results obtained with MAC-REDDCC.

TABLE 4.33: Comparatives between the best results obtained for evolutionary algorithms

n	Method	↑	↓	↔	Min	Max	$\bar{N}_n$	$\tilde{N}_n$	σ_{N_n}
8	MAC-REDDCC	0	0	2	116	116	116	116	0
	LMA-REDDC	0	0	2	116	116	116	116	0
	LMA-RGE	0	0	2	116	116	116	116	0
10	**MAC-REDDCC**	**2**	**0**	**0**	**488**	**488**	**488**	**488**	**0**
	LMA-REDDC	1	1	0	486	488	487.44	488	0.907
	LMA-RGE	0	2	0	482	484	483.92	484	0.396

We see that the results obtained for LMA-RGE in the 10-variable case are poor in comparison with the ones obtained with LMA-REDDC and MAC-REDDCC. Thus, we can see here the importance of using a scheme based on diversity in order to find proper solutions for the problem of generating BFs with high nonlinearity. When compared to LMA-REDDC, we can see that the novel proposal attains slightly better results, so the changes performed in this thesis were required to attain good values with a success ratio close to 100%. Thus, the modifications developed in this thesis, increase the stability of the proposal.

FIGURE 4.24: Population entropy for MAC-REDDCC, LMA-REDDC and LMA-RGE



(A) Population entropy $n = 8$ (B) Population entropy $n = 10$

Figure 4.24 shows the population entropy for the different evolutionary methods. We can see that the LMA-RGE never converges. For the

LMA-REDDC method, we see that is the method that looses more diversity along the execution time. The MAC-REDDCC method has almost the same behave in both figures 4.24a and 4.24b, with the difference that for $n = 8$ the solutions found are the best known and there exist a lot of such functions. In the 10-variable case we see that the method losses more diversity that in the 8-variable case, this is because is harder to achieve high nonlinearity for 10 variables than 8 variables, so the algorithm should perform more exploitation of the search space in order to found solutions with high nonlinearity.

4.5 Hybridization with Algebraic Techniques

With the aim of improving the results obtained with the previously presented algorithms, we decided to use an initial population of individuals that already have high nonlinearity. This initial population is constructed by employing the algorithm 31. We decided to make experiments with the hybrid method only for the 10-variable BF problem. The reason behind this is that the algorithm 31 allows us to construct only 56 BFs $h \in \mathcal{B}_3$ for $n = 8$, meaning that we can only build BFs in 56 different regions of the search space, and for each region we can build only $4! = 24$ BFs. Moreover, achieving BFs with nonlinearity equal to 116 is relatively easy with our methods. In the case of constructing BFs with nonlinearity equal to 488 for the 10 variable case, we can note that there are 10920 BFs $h \in \mathcal{B}_4$, meaning that we can build BFs in 10920 different regions of the search space, and in each region we can build $5! = 120$ BFs, so, we can construct 1310400 different BFs with nonlinearity equal to 488.

In order to make 50 independent runs and each one with a population size equal to 350, we build 17500 ($50 \times 350 = 17500$) individuals with nonlinearity equal to 488 with the algorithm 31. In order to explore a lot of regions of the search space, it was ensured that the individuals belonging to each population were not close to each other. Each population was employed along with the three evolutionary methods: MAC-REDDCC, LMA-REDDC and LMA-RGE. The parameter values are the indicated in table 4.32. The attained results are shown in table 4.34.

TABLE 4.34: Comparatives between the best results obtained for the hybrid methods

n	Method	$\uparrow$	$\downarrow$	$\leftrightarrow$	Min	Max	$\bar{N}_n$	$\tilde{N}_n$	σ_{N_n}
10	MAC-REDDCC	0	0	2	492	492	492	492	0
	LMA-REDDC	0	0	2	492	492	492	492	0
	LMA-RGE	0	0	2	492	492	492	492	0

We can see that the information in table 4.34 does not help us to discern which hybrid method is better in comparison with the others because all the

methods achieve in all the executions BFs with nonlinearity equal to 492. Note that this is currently the best-known solution. In order to discern which method is preferable we compare the amount of individuals in the final population that achieves a nonlinearity equal to 492. This comparative is shown in table 4.35.

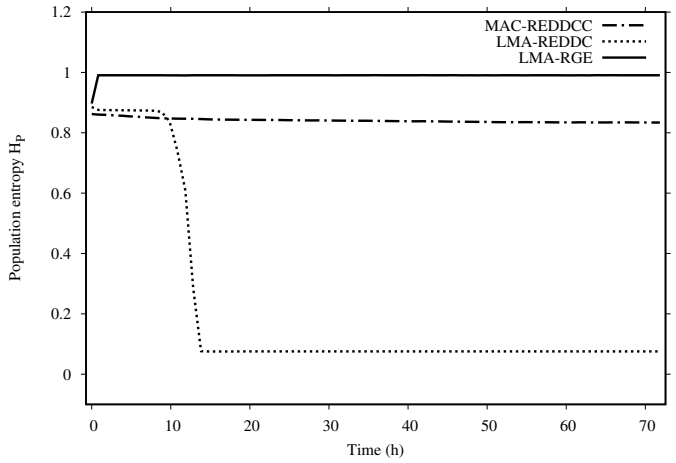
TABLE 4.35: Comparison between hybrid methods for the amount of individuals found with nonlinearity equal to 492

Method	Min	Max	Mean	Median
MAC-REDDCC	350	350	350	350
LMA-REDDC	350	350	350	350
LMA-RGE	1	1	1	1

From the information in the table 4.35, we can see that in MAC-REDDCC and LMA-REDDC every individual in the final population has nonlinearity equal to 492. However, the LMA-RGE is able to find only one solution with such nonlinearity. We consider that using the MAC-REDDCC and LMA-REDDC is preferable, because maintaining a diverse population means that several regions are explored and there is a larger probability to find better solutions (if they exist). However, we are not able to discern between the REDDCC and the REDDC strategies.

Figure 4.25 shows the entropy of the population along the execution time for the three methods tested. This last comparative will be between the

FIGURE 4.25: Population entropy for MAC-REDDCC, LMA-REDDC and LMA-RGE



methods MAC-REDDCC and LMA-REDDC. As we can see in figure 4.25, MAC-REDDCC never decrease its diversity but is able to reach very high quality solutions, so in this sense its behavior seems more adequate to try to reach a solution with a higher nonlinearity (if it exists).

Chapter 5

Conclusions and Future Work

This chapter is devoted to describe the conclusions of the work developed in this thesis and to describe some lines of future work that are considered to be promising.

5.1 Conclusions

The problem of generating Cryptographic Boolean Functions (CBFs) with high nonlinearity is an extremely complicated problem. This problem has been addressed with many strategies in the last 30 years. Many heuristic methods have been proposed to generate CBFs with high nonlinearity and the results obtained have been improving among the years. However, the results obtained so far are not as extraordinary as those obtained with algebraic techniques. According to our knowledge, heuristic methods have never been able to generate 10-variable CBFs with nonlinearity equal to 492, but the algebraic constructions do.

In this thesis, we work with the hypothesis that a diversity-based metaheuristic can provides a better way of generating CBFs with high nonlinearity, improving further the best current results obtained by other metaheuristic methods and reducing the gap between algebraic and metaheuristic approaches. Based on this, a set of trajectory-based and population-based metaheuristic methods are proposed.

Among all of methods proposed, the most novel is a population-based metaheuristic that incorporates explicit diversity management with a clustering technique that allows intensifying and exploring throughout the optimization process, since it forces some members of the population to be distant but some are allowed to be close. This method is a Memetic Algorithm with Clusters coupled with a Replacement with Elite based Dynamic Diversity Control with Clustering (MAC-REDDCC) survivor selection technique. According to our knowledge, we are the first ones that proposes the use of a diversity scheme to generate CBFs with high nonlinearity.

MAC-REDDCC works with a set of novelty components proposed in this thesis. The most important component is the REDDCC strategy, which is responsible for the explicit diversity management. The cost function employed to guide the search is another important component. The cost function devised takes into account more information from the Walsh Hadamard Transform (WHT), what makes it more suitable to guide the search. The local search method employed to improve the members of the population is another important component. This novel local search method is inspired by the Tabu Search algorithm and is called First Improvement Quasi-Tabu Search (FIQTS).

The results obtained with the MAC-REDDCC method improves the best current results reported in literature by other pure metaheuristic methods, even match the results reported by hybrid metaheuristics with algebraic techniques with success rate close to 100%.

As another researchers, we propose an hybridization between the MAC-REDDCC method and an algebraic technique. The results obtained with the hybridization are the best results known for 10-variable CBFs, since CBFs with nonlinearity equal to 492 are generated. Being MAC-REDDCC the first metaheuristic method that is able to generate CBFs with that nonlinearity.

5.2 Future Work

As a future work we should say that the methods implemented here have the drawback that are not scalable. After performing some initial experiments with 12 variables, we concluded that we need to do some changes — probably in the cost functions and in the local search procedures — to be able to deal with 12 or more variables. Particularly, in order to develop a better local search procedure it would be interesting to use a neighborhood coupled with the Walsh Hadamard transform. In this way, the improvement process could be faster. In other words, extending the smart hill climbing algorithm 9 to operate with other cost functions seems promising. Additionally, it seems interesting to employ another properties such as the autocorrelation or the algebraic degree inside the cost function. Some authors have obtained improvements with these kinds of transformations, so integrating them with the advances developed in this thesis seems promising.

Bibliography

[1] Emile Aarts, Emile HL Aarts, and Jan Karel Lenstra. *Local search in combinatorial optimization*. Princeton University Press, 2003.

[2] Thomas Back. *Evolutionary algorithms in theory and practice: evolution strategies, evolutionary programming, genetic algorithms*. Oxford university press, 1996, pp. 109–113.

[3] Richard Bellman. *Dynamic programming*. Courier Corporation, 2013.

[4] Walid Ben-Ameur. "Computing the initial temperature of simulated annealing". In: *Computational Optimization and Applications* 29.3 (2004), pp. 369–385.

[5] Linda Burnett et al. "Simpler methods for generating better Boolean functions with good cryptographic properties". In: *Australasian Journal of Combinatorics* 29 (2004), pp. 231–248.

[6] Anne Canteaut. "Lecture notes on Cryptographic Boolean Functions". In: *Inria, Paris, France* (2016).

[7] Claude Carlet. "Boolean functions for cryptography and error correcting codes". In: *Boolean models and methods in mathematics, computer science, and engineering* 2 (2010), pp. 257–397.

[8] Xianshun Chen et al. "A multi-facet survey on memetic computation". In: *IEEE Transactions on Evolutionary Computation* 15.5 (2011), pp. 591–607.

[9] Andrew Chipperfield et al. "Genetic algorithm toolbox for use with MATLAB". In: (1994), pp. 8–9.

[10] Benny Chor et al. "The bit extraction problem or t-resilient functions". In: | *26th Annual Symposium on Foundations of Computer Science*. IEEE. 1985, pp. 396–407.

[11] John A Clark and Jeremy L Jacob. "Two-stage optimisation in the design of Boolean functions". In: *Australasian Conference on Information Security and Privacy*. Springer. 2000, pp. 242–254.

[12] John A Clark, Jeremy L Jacob, and Susan Stepney. "Searching for cost functions." In: *IEEE Congress on Evolutionary Computation*. 2004, pp. 1517–1524.

[13] John A Clark et al. "Almost Boolean functions: The design of Boolean functions by spectral inversion". In: *Evolutionary Computation, 2003. CEC'03. The 2003 Congress on*. Vol. 3. IEEE. 2003, pp. 2173–2180.

[14] Thomas W Cusick and Pantelimon Stanica. *Cryptographic Boolean functions and applications*. Academic Press, 2017.

[15] Hans Dobbertin. "Construction of bent functions and balanced Boolean functions with high nonlinearity". In: *International Workshop on Fast Software Encryption*. Springer. 1994, pp. 61–74.

[16] Adham M Elhosary et al. "State of the art in Boolean functions cryptographic assessment". In: *International Journal Of Computer Networks And Communications Security* 1.3 (2013), pp. 88–94.

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

[18] Xiang-dong Hou. "On the norm and covering radius of the first-order Reed-Muller codes". In: *IEEE Transactions on Information Theory* 43.3 (1997), pp. 1025–1027.

[19] Yuriy Izbenko, Vladislav Kovtun, and Alexandr Kuznetsov. "The design of boolean functions by modified hill climbing method". In: *Information Technology: New Generations, 2009. ITNG'09. Sixth International Conference on*. IEEE. 2009, pp. 356–361.

[20] Jonathan Katz and Yehuda Lindell. *Introduction to Modern Cryptography (Chapman 8 Hall/Crc Cryptography and Network Security Series)*. 2007.

[21] Selçuk Kavut and Melek D Yücel. "Improved cost function in the design of Boolean functions satisfying multiple criteria". In: *International Conference on Cryptology in India*. Springer. 2003, pp. 121–134.

[22] Scott Kirkpatrick, C Daniel Gelatt, and Mario P Vecchi. "Optimization by simulated annealing". In: *science* 220.4598 (1983), pp. 671–680.

[23] John R Koza. *Genetic programming II, automatic discovery of reusable subprograms*. MIT Press, Cambridge, MA, 1992.

[24] Florence Jessie MacWilliams and Neil James Alexander Sloane. *The theory of error-correcting codes*. Elsevier, 1977.

[25] Subhamoy Maitra and Palash Sarkar. "Cryptographically significant Boolean functions with five valued Walsh spectra". In: *Theoretical Computer Science* 276.1-2 (2002), pp. 133–146.

[26] Luca Mariot and Alberto Leporati. "Heuristic search by particle swarm optimization of boolean functions for cryptographic applications". In: *Proceedings of the Companion Publication of the 2015 Annual Conference on Genetic and Evolutionary Computation*. ACM. 2015, pp. 1425–1426.

[27] Nicholas McDonald. "Past, present, and future methods of cryptography and data encryption". In: *URL: http://www. eng. utah. edu/~ nmcdonal/Tutorials/EncryptionResearchReview. pdf (2017-09-17)* (2009).

[28] Robert L McFarland. "A family of difference sets in non-cyclic groups". In: *Journal of Combinatorial Theory, Series A* 15.1 (1973), pp. 1–10.

[29] James McLaughlin and John A Clark. "Evolving balanced Boolean functions with optimal resistance to algebraic and fast algebraic attacks, maximal algebraic degree, and very high nonlinearity." In: *IACR Cryptology ePrint Archive* 2013 (2013), p. 11.

[30] William Millan, Andrew Clark, and Ed Dawson. "An effective genetic algorithm for finding highly nonlinear Boolean functions". In: *International Conference on Information and Communications Security*. Springer. 1997, pp. 149–158.

[31] William Millan, Andrew Clark, and Ed Dawson. "Boolean function design using hill climbing methods". In: *Australasian Conference on Information Security and Privacy*. Springer. 1999, pp. 1–11.

[32] William Millan, Andrew Clark, and Ed Dawson. "Heuristic design of cryptographically strong balanced Boolean functions". In: *International Conference on the Theory and Applications of Cryptographic Techniques*. Springer. 1998, pp. 489–499.

[33] William Millan, Andrew Clark, and Ed Dawson. "Smart hill climbing finds better boolean functions". In: *Workshop on Selected Areas in Cryptology 1997, Workshop Record*. Citeseer. 1997, pp. 50–63.

[34] Pablo Moscato et al. "On evolution, search, optimization, genetic algorithms and martial arts: Towards memetic algorithms". In: *Caltech concurrent computation program, C3P Report* 826 (1989), p. 1989.

[35] Quang Huy Nguyen, Yew Soon Ong, and Meng Hiot Lim. "Non-genetic transmission of memes by diffusion". In: *Proceedings of the 10th annual conference on Genetic and evolutionary computation*. ACM. 2008, pp. 1017–1024.

[36] Michael G Norman and Pablo Moscato. "A competitive and cooperative approach to complex combinatorial search". In: *Proceedings of the 20th Informatics and Operations Research Meeting*. Citeseer. 1991, pp. 3–15.

[37] Stjepan Picek, Roberto Santana, and Domagoj Jakobovic. "Maximal nonlinearity in balanced boolean functions with even number of inputs, revisited". In: *Evolutionary Computation (CEC), 2016 IEEE Congress on*. IEEE. 2016, pp. 3222–3229.

[38] Stjepan Picek, Dominik Sisejkovic, and Domagoj Jakobovic. "Immunological algorithms paradigm for construction of Boolean functions with good cryptographic properties". In: *Engineering Applications of Artificial Intelligence* 62 (2017), pp. 320–330.

[39] Stjepan Picek et al. "Cryptographic Boolean functions: One output, many design criteria". In: *Applied Soft Computing* 40 (2016), pp. 635–653.

[40] Stjepan Picek et al. "Evolutionary methods for the construction of cryptographic boolean functions". In: *European Conference on Genetic Programming*. Springer. 2015, pp. 192–204.

[41] Stjepan Picek et al. “Fighting the symmetries: The structure of cryptographic boolean function spaces”. In: *Proceedings of the 2015 Annual Conference on Genetic and Evolutionary Computation*. ACM. 2015, pp. 457–464.

[42] Ingo Rechenberg. “Evolutionsstrategie–Optimierung technisher Systeme nach Prinzipien der biologischen Evolution”. In: (1973).

[43] Stuart J Russell and Peter Norvig. *Artificial intelligence: a modern approach*. Malaysia; Pearson Education Limited, 2016.

[44] Palash Sarkar and Subhamoy Maitra. “Nonlinearity bounds and constructions of resilient Boolean functions”. In: *Annual International Cryptology Conference*. Springer. 2000, pp. 515–532.

[45] Carlos Segura et al. “A novel diversity-based replacement strategy for evolutionary algorithms”. In: *IEEE transactions on cybernetics* 46.12 (2016), pp. 3233–3246.

[46] Carlos Segura et al. “Improving diversity in evolutionary algorithms: New best solutions for frequency assignment”. In: *IEEE Trans. Evol. Comput* 21.4 (2017), pp. 539–553.

[47] Carlos Segura et al. “The Importance of Proper Diversity Management in Evolutionary Algorithms for Combinatorial Optimization”. In: *NEO 2015*. Springer, 2017, pp. 121–148.

[48] Claude E Shannon. “Communication theory of secrecy systems”. In: *Bell system technical journal* 28.4 (1949), pp. 656–715.

[49] Kenneth Sörensen, Marc Sevaux, and Fred Glover. “A history of metaheuristics”. In: *Handbook of heuristics* (2018), pp. 1–18.

[50] T Stützle. “Local search algorithms for combinatorial problems”. In: *Darmstadt University of Technology PhD Thesis* 20 (1998).

[51] El-Ghazali Talbi. *Metaheuristics: from design to implementation*. Vol. 74. John Wiley & Sons, 2009.

[52] Deng Tang, Weiguo Zhang, and Xiaohu Tang. “Construction of balanced Boolean functions with high nonlinearity and good autocorrelation properties”. In: *Designs, codes and cryptography* 67.1 (2013), pp. 77–91.

[53] Chuan-Kun Wu, Dengguo Feng, et al. *Boolean functions and their applications in cryptography*. Springer, 2016.

[54] G-Z Xiao and James L Massey. “A spectral characterization of correlation-immune combining functions”. In: *IEEE Transactions on information theory* 34.3 (1988), pp. 569–571.