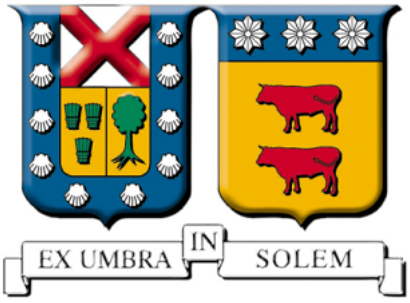
Universidad Técnica Federico Santa María
Departamento de Informática
Valparaíso - Chile



Opposite Learning Strategies for Improving the Search Process of Ant-based Algorithms

NICOLÁS EMILIO ROJAS MORALES
nicolasrojas@acm.org

A Thesis submitted in fulfillment
of the requirements for the degree of
Doctor in Informatics Engineering

Supervisor MARÍA CRISTINA RIFF ROJAS

2018

Acknowledgements

Agradezco a Francisca y a nuestro hijo Gabriel, por todo su inmenso apoyo y amor. A mi padre Miguel, a mi madre Emilia y a mi hermana Francisca por ser el centro y raíz de todo. A Veronica, Hernán, Panchi, Felipe, Gonzalo, Paula, Omar, Gabriela, Vicky y a la familia presente.

A María-Cristina Riff por su perseverancia y toda su energía entregada en este proyecto. A Elizabeth Montero por su contención, amistad y ayuda eterna. A Bertrand Neveu por todas sus enseñanzas y lecciones en este período. A Thomas Stützle, Carlos A. Coello Coello, Gabriela Ochoa, Leslie Perez, Ignacio Araya, Miriam Pescador y Raquel Hernandez por sus palabras y consejos.

Finalmente, a mis amigos y amigas de la vida, del LabIAA y de la música.

Abstract

This thesis proposes an Opposite-Inspired Learning strategy for ant-based algorithms that were designed and are able to solve combinatorial problems. We are interested in improving the search process of a target ant-based algorithm, in terms of the quality of the solutions it can obtain. Our strategy is focused on learning about such an undesirable characteristic that can bias some intermediate decisions performed during the construction process. Usually, these intermediate decisions give priority to locally interesting components and in some cases, only poor quality solutions can be reached. Inspired in the concept of anti-pheromone, the idea is to produce a repel effect to (temporarily) avoid components that were related to an undesirable characteristic. The claim of our work is the following: if we consume a certain amount of resources in identifying and learning about some undesired characteristic, the search process could be further focused making decisions using this knowledge so that we can obtain complete instantiations of a better quality. The objective of our strategy is to change some intermediate decisions allowing the selected ant-based algorithm to consider components that could not be evaluated or are initially discarded.

To evaluate our strategy we selected ant-based algorithms designed for solving two different combinatorial problems: the Multidimensional Knapsack Problem (a Constraint Optimization Problem) and Constraint Satisfaction Problems (CSPs). We selected Ant Knapsack as the baseline algorithm for solving the Multidimensional Knapsack Problem and Focused Ant Solver, for solving Constraint Satisfaction Problems. The inclusion of our strategy requires the study of both problems and algorithms. Results in benchmark instances show that both algorithms were allowed to reach better quality solutions. In the case of Ant Knapsack, the use of opposite information allows the algorithm to reach better quality solutions. About Focused Ant Solver, the use of opposite information allowed the algorithm to solve more problems from the transition phase. Boxplots, statistical analysis and fitness-distance plots are presented to assess how the search process of these ant-based algorithms are modified. Moreover, the search process of the two baseline algorithms were able to obtain different solutions, in terms of quality and structure.

In addition to the evaluation of our strategy in Ant Knapsack and Focused Ant Solver, a cooperative scheme named COISA is presented as a preliminary work in this Thesis. The objective of this scheme is to evaluate the collaboration of different sub-colonies of ants using opposite information.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Since ant-based algorithms have been proposed for solving Combinatorial Optimization Problems in 1992 [Dor92], there have been several applications in different types of problems (i.e, Discrete [DDCG99], Continuous [SD08], Multi-Objective Optimization Problems [LIS12]), new strategies and variants, parameters analysis [SLIP$^+$12], among others.

In an ant-based algorithm, each ant (or agent) constructs candidate solutions for a problem of interest. The construction process of partial instantiations is based on decisions taken using two components: local information in the form of pheromone and heuristic information. Pheromone is used as a communication method among the ants in the colony, to guide the search process to interesting regions of the search space. On the other hand, heuristic knowledge measures the effect of each possible assignment using information on the problem being solved.

When the problems are complex to solve, some of the decisions made during the construction process can lead the algorithm to complete instantiations with less quality than expected. Our work is focused on ant-based algorithms that have shown to be able to solve complex combinatorial problems, but in some stages of the construction of partial instantiations they make some decisions that could be improved. In other words, the construction process can be biased by the attraction of some characteristic of the candidate components. We named this characteristic as *undesirable* ($u\mathcal{D}$) and our objective is to provide useful information to an ant-based algorithm, to guide the search process in a better way, when decisions can lead to poor quality complete instantiations. We propose an Opposition-Inspired Learning strategy to identify an $u\mathcal{D}$-characteristic from complete instantiations. The idea is to reduce the attraction of this particular $u\mathcal{D}$-characteristic, giving a chance to other components that the original algorithm would not consider.

Opposite Learning (OL) was the main motivation of this work. Chapter 2 presents the principal concepts and definitions related to OL. Many metaheuristic approaches related to Opposite Learning have been proposed in the last 10 years. These approaches can be classified in two main areas: Opposition-Based Learning (OBL) and Opposition-Inspired Learning (OIL). OBL metaheuristic approaches were mostly proposed for solving continuous problems. On the other hand, OIL approaches are designed for solving discrete problems. A revision and a classification of existing OBL and OIL metaheuristic approaches is presented in Chapter 2.

Our strategy is proposed for ant-based algorithms solving combinatorial problem. Chapter 3 presents a review of the most important and pioneer ant-based approaches. The first ant-based algorithm was proposed in 1992 and the motivation behind the design of this algorithm is related to the organization, communication methods and stimuli response of real ants. Section 3.1 presents a brief revision of how the behavior of real ants motivates the creation and design of ant-based algorithms.

The pioneer ant-based approaches are Ant System, Ant Colony System, $\mathcal{MAX} - \mathcal{MIN}$ Ant System, among others. Mainly, these approaches were originally proposed for solving the Travelling Salesman Problem. Section 3.3 introduce these algorithms and also, some applications in other combinatorial problems are detailed in Section 3.4.3. As our strategy was mainly inspired in the OL literature, Section 3.3.5 introduces the concept and presents details of the first application of *anti-pheromone*.

Our Opposite-Inspired Learning strategy is presented and detailed in Chapter 4. In order to evaluate alternative possibilities of identifying an $u\mathcal{D}$-characteristic, we proposed three different *Methods*. These methods will perform a particularly defined search process and some components should be defined. Section 4.3 presents details of these three *Methods*. To evaluate our strategy we consider ant-based algorithms proposed for solving two different combinatorial problems: the Multidimensional Knapsack Problem (a Constraint Satisfaction and Optimization Problem) and for binary Constraint Satisfaction Problems (CSPs).

Chapter 5 presents the application in an ant-based algorithm for solving the Multidimensional Knapsack Problem (MKP). Section 5.1 formally defines MKP and also, presents a brief revision of existing efficiency metrics, heuristics, metaheuristic approaches and benchmark instances. We

selected Ant Knapsack as the baseline algorithm to evaluate our strategy and Section 5.2 introduces the main components of this technique. Experiments in large-size benchmark instances were performed and results show that the inclusion of opposite information allows Ant Knapsack to be competitive with state-of-the-art algorithms.

Chapter 6 presents the details of the inclusion of our strategy in an ant-based algorithm for solving CSPs. Definitions and a review of existing approaches for solving these problems are presented in Section 6.1. We selected Focused Ant Solver as the baseline algorithm for CSPs. Section 6.2 introduces this algorithm and Section 6.3 present details of the implementation of our strategy. Experiments using randomly-generated binary CSPs were considered to evaluate the inclusion of the strategy in FAS. The generated instances are part to the transition phase, where the hardest problems are. Experiments and results are presented in Section 6.4. Results show that the inclusion of opposite information allows Focused Ant Solver to solve more problems.

For both baseline algorithms, statistical analysis, boxplots and fitness-distance plots are presented in order to confirm that using opposite information allows the original algorithms to reach different solutions, in terms of quality and structure. In addition, fine tuning processes were performed using Evolutionary Calibrator (EVOCA), a well-known tuner algorithm based on Evolutionary Algorithms.

COISA is a collaborative scheme that considers the cooperation of sub-colonies of ants and it is proposed in Chapter 7. Despite that the key idea and main components of COISA are the same as in our OIL strategy, COISA has some implementation and design differences. Section 7.1 introduces the details of *COISA* and explains how the sub-colonies cooperate in the construction of a pheromone matrix. The *First Step* of this scheme is executed in parallel, considering some threads focused on the construction of solutions and other focused in the management of pheromone information. Main details of the parallelism are presented in Section 7.2 and preliminary results in benchmark instances are presented in Section 7.4.

Finally, Chapter 8 present some conclusions and possible paths for future work.

Chapter 2

Opposite Learning

A metaheuristic can be defined as an iterative generation process which guides a subordinate heuristic by intelligently combining different concepts for exploring and exploiting the search space [OL96]. In general, metaheuristics identify promising regions of the search space and exploit them to obtain the best quality solutions from those regions. However, metaheuristics based approaches present typical difficulties: getting trapped in local optima, having convergence problems, among others. For these reasons, the design of strategies for improving the search of metaheuristics has been studied for several years [Tal02, Tal09].

Opposite Learning (OL) has been introduced in optimization and metaheuristics as a search strategy for obtaining complementary candidates from a set of solutions [Tiz05]. The key idea is to support the search process of a target algorithm considering alternative candidate solutions. A relationship between two candidate solutions is defined as "opposite" and the highest quality solution among them is kept to continue the search process. The objective of mapping candidate solutions is to increase the coverage of the search space, accuracy and convergence of the search process [Mal08].

Many OL metaheuristic approaches have been reported in the metaheuristics community [RRM17, MRD17, XWW+14]. OL has been mainly applied with different objectives like: initialization process of metaheuristics, generation of solutions, parameter control, local search procedures, perturbation of solutions, among other goals. Existing Opposite Learning approaches can be classified in two main areas: Opposition-Based Learning (OBL) and Opposition-Inspired Learning (OIL). OBL is more related to solve continuous optimization problems and OIL to combinatorial optimization problems. This chapter presents the main concepts and ideas of both, OBL and OIL.

First, eight types of Opposition-Based Learning are defined and explained, and also several OBL metaheuristic approaches are presented. Then, Opposition-Inspired Learning is presented followed by a description of some metaheuristic approaches inspired in opposite ideas. All the approaches here reviewed were classified and described considering four important characteristics:

- The objective of including OL: we analyze the principal reason of including an Opposite Learning component into an algorithm. This shows some possible deficiencies of metaheuristics that can be tackled with OL.
- The role of OL: this feature shows the task performed by a component based on OL.
- The type of OL: this attribute shows which type of OBL or OIL is applied in each approach.
- The type of problem tackled: continuous or discrete optimization problem.

Finally, a discussion and a taxonomy are presented to classify all these Opposite Learning metaheuristics. In the following section Opposition-Based Learning is presented.

2.1 Opposition-Based Learning

There are several definitions of what *opposition* is, considering a long list of situations and contexts where this concept can be applied.[1] Focused on metaheuristics and optimization, *opposition* has been presented as a relationship between a pair of candidate solutions. Both in combinatorial and continuous optimization, each candidate solution has its own particularly defined opposite candidate. In general, there are two ways of searching using opposite candidate solutions: defining a function for mapping every solution of the search space with its own opposite solution or searching for solutions with opposite quality.

Let us suppose that we are interested in solving a problem whose variable domains are defined in defined $\mathcal{C}$ space, where $\mathcal{C}$ can be a set of numerical (e.g, discrete, continuous) or categorical concepts. As Opposition-Based Learning (OBL) is more related to continuous problems, we will consider in this chapter that $\mathcal{C}$ space is a real number space $\mathbb{R}$.

Let us suppose that $X = [x_1, x_2, \ldots, x_M]$ is a point in a M-dimensional space where $x_i \in [a_i, b_i], \forall i \in [1, M]$. Next, we will present different existing types of OBL to obtain opposite candidate solutions $\breve{X}$ that consider variables in a M-dimensional space.

[1] Def: Completely different; of a contrary kind. (Oxford Dictionary - `http:\\en.oxforddictionaries.com`)

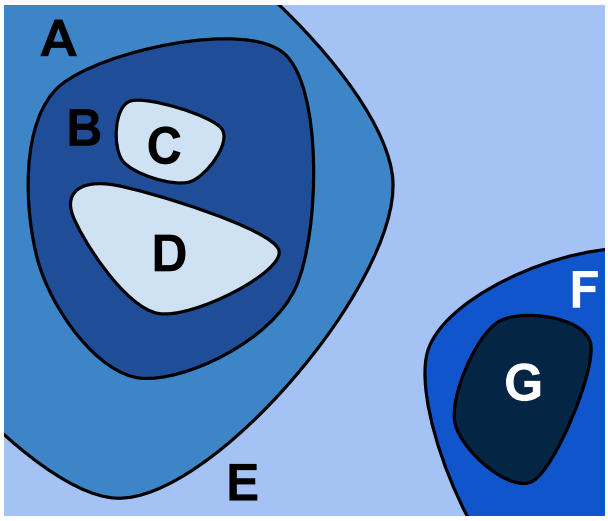


Figure 2.1: Search space example

Definition 2.1.1. Type-I Opposition: Given $\Phi : \mathbb{R} \to \mathbb{R}$ an opposition mapping function that maintains the characteristics of the input space. Thus, a Type-I Opposite candidate solution is defined by $\breve{X} = \Phi(X)$

Type-I Opposition was proposed in the first article related to OBL [Tiz05]. Here, Φ is a function that maps every solution X in the search space to its opposite solution $\breve{X}$. The most commonly used Φ function considers the boundaries of the domain of each dimension of X. Considering $x_i \in [a_i, b_i]$ the i^{th} dimension of X, its opposite $\breve{x_i}$ can be defined as:

$$\breve{x_i} = a_i + b_i - x_i \tag{2.1}$$

Figure 2.1 shows a search space example of a problem to be solved. There are seven regions named from A to G, where the darker the region the better the quality solution. Suppose that a candidate solution X is in region D, where lowest quality solutions are placed. Using a particularly defined mapping function Φ, we can obtain its opposite candidate solution $\breve{X} = \Phi(X)$. This solution can be in other region of the search space, for example in Figure 2.1, in region B. The key idea is to generate alternative candidate solutions using Φ and continue the search process with the highest quality solutions. It is important to emphasize that Φ is defined as a one-to-one mapping function. Moreover, the relationship between two candidate solutions is symmetric: $\Phi(X) = \breve{X}$ and $\Phi(\breve{X}) = X$.

From Type-I Opposition, two extensions were proposed according to a function that mea-

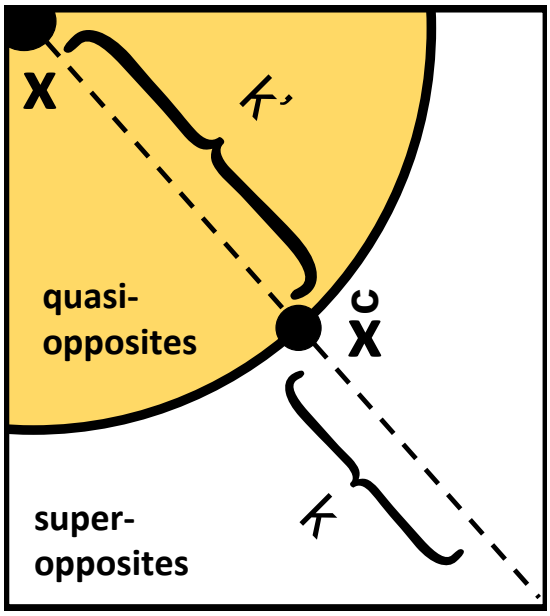


Figure 2.2: Quasi and Super Opposition example

sures the distance between X and $\breve{X}$: *Type-I Quasi-Opposition* [RTS07c] and *Type-I Super-Opposition* [TVR08]:

Definition 2.1.2. Type-I Quasi-Opposition: Given $X \in \mathbb{R}$ and d a distance function. We define that points $\breve{X}_q$ are Type-I Quasi-opposite of X when $d(\breve{X}_q, X) < d(\breve{X}, X)$.

Definition 2.1.3. Type-I Super-Opposition: Given $X \in \mathbb{R}$ and d a distance function. We define points $\breve{X}_s$ are Type-I Super-opposite of X when $d(\breve{X}_s, X) > d(\breve{X}, X)$.

A distance relation is defined between these opposite types. Figure 2.2 shows an example in a two dimensional space. Let us suppose that the distance between a candidate solution X and its Type-I opposite $\breve{X}$ is $d(X, \breve{X}) = k'$. Quasi-opposite points $\breve{X}_q$ are defined closer to the original candidate solution X than a Type-I Opposition point $\breve{X}$. Here, a Quasi-opposite point $\breve{X}_q$ can be defined considering $d(\breve{X}_q, X) < k'$. In the figure, $\breve{X}_q$ can be inside the region colored in yellow. The hypothesis is that quasi-opposite points have a higher chance to be closer to an optimal solution than opposite points [TVR08].

On the other hand, compared to a Type-I Opposition point $\breve{X}$, a Super-opposite candidate solutions $\breve{X}_s$ is defined farther from X. In the figure, $\breve{X}_s$ can be defined in the white color region. Moreover, a Super-opposite point $\breve{X}_s$ can be defined as $(k' + k) \geq d(\breve{X}_s, X) > d(\breve{X}, X)$.

There is other existing type of opposition named *Type-II Opposition*, that defines a relationship

between a pair of candidate solutions considering an evaluation function for mapping them.

Definition 2.1.4. Type-II Opposition: Given $X \in \mathbb{R}$ and $\Upsilon : \mathbb{R} \to \mathbb{R}$ by a Type-II Opposition mapping that requires a defined performance function f. Then, the set of Type-II opposites of $\mathbb{R}$ is completely defined by Υ.

As was proposed in [VT07], let us suppose that $\Upsilon(X) = \min(f(X), f(\breve{X}))$ for a minimization problem. An opposite candidate $\breve{X}$ is calculated and evaluated by Υ considering a fixed value Δ: $\breve{X} = X \pm \Delta$. Υ allows to compute an opposite candidate and compare its quality with X. The candidate solution that optimizes the objective of the problem being solved will be considered to continue the search process of the algorithm.

However, a different application of Type-II Opposition was proposed in [TVR08]. Here, an opposite candidate $\breve{X}$ is computed for a continuous problem considering the maximum and minimum values of the performance function f. $\breve{X}$ is computed such that its quality value $\breve{y}$ is:

$$\breve{y} = y_{max} + y_{min} - y \tag{2.2}$$

where $y = f(X)$ and $y \in [y_{min}, y_{max}]$. The *Type-II Opposition* strategy has been used for exploring the search space, maintaining highest quality solutions.

Generalized OBL (GOBL) [WWRK09] is a type of OBL originally presented as *Space Transformation Search* (STS) [WWL+09]. Similarly to the idea behind Quasi-Opposition, the aim of GOBL is to obtain candidate solutions closer to an hypothetical global optimum.

Definition 2.1.5. Generalized OBL: Given $X = [x_1, \ldots, x_M]$ a solution in a M-dimensional space. An opposite candidate solution $\breve{X}_g = [\breve{x}_1, \ldots, \breve{x}_M]$ is defined by a weight parameter k that controls the distance between each opposite dimension and its original. The dimension j of X is mapped using:

$$\breve{x}_j = k * (a_j + b_j) - x_j \tag{2.3}$$

where k is a random number $\in [0, 1]$ and a_j and b_j are the minimum and maximum values for j^{th} dimension of X.

If $k = 0$ the opposite candidate is defined as $\breve{x}_j = -x_j$. In this case, $\breve{x}_j$ is a reflection of x_j through the origin (0). Also, if $k = 1$ the opposite candidate is equal to a Type-I Opposition candidate (Equation 2.1). When $k \in (0, 1)$, different values for $\breve{x}_j$ will be obtained. The idea of this type of OBL is to map candidate solutions controlling the distance between opposite and

original candidate solutions.

Center-Based Sampling (CBS) [RW09] is a type of OBL similar to GOBL. Here, the objective is to obtain opposite candidate solutions closer to the center of the domain of each variable.

Definition 2.1.6. Center-Based Sampling: Given $X = [x_1, \ldots, x_M]$ a solution in a M-dimensional space. An opposite candidate solution $\breve{X}_c = [\breve{x}_{c_1}, \ldots, \breve{x}_{c_M}]$ is defined by a random point between X and its opposite point $\breve{X}$. Each dimension j of X is mapped using:

$$\breve{x}_{c_j} = rand_j * (a_j + b_j - 2 * x_j) + x_j \tag{2.4}$$

where $rand_j$ is a uniformly distributed random number $\in [0, 1]$, $j \in [1, M]$ and a_j and b_j are the minimum and maximum values of the j^{th} component of X.

In literature there exist other types of OBL: *Quasi-Reflection OBL* (QR_{OBL}) [ESD09] and *OBL using the Current Optimum* (CO_{OBL}) [XWHW11]. Table 2.1 summarizes all these types of OBL. For each type, we present the opposite candidate $\breve{X}$ of a solution X and the required inputs for obtaining this opposite candidate solution.

OBL Type	**Opposite of X**	**Possible opposite solution obtained $\breve{X}$**	**Input**
Type-I Opposition	$\breve{X}$	Solution between two defined boundaries	-
Type-I Quasi-Opposition	$\breve{X}_q$	Solution closer to X than $\breve{X}$	$\breve{X}$, Distance Metric
Type-I Super-Opposition	$\breve{X}_s$	Solution farther to X than $\breve{X}$	$\breve{X}$, Distance Metric
Type-II Opposition	$\breve{X}_{II}$	Solution with opposite performance	f Evaluation Function
Generalized OBL	$\breve{X}_g$	Solution can be closer to X than $\breve{X}$	k parameter
Quasi-Reflection OBL	$\breve{X}_{qr}$	Reflection from the center of the domain of the $\breve{X}_q$	$\breve{X}_q$
Center-Based Sampling	$\breve{X}_c$	Solution close to the center of the domain	-
OBL using Current Optimum	$\breve{X}_{COOBL}$	Solution related to the current optimum (X_{co}) or same as GOBL	X_{co}, $rand(0,1)$

Table 2.1: OBL types summary

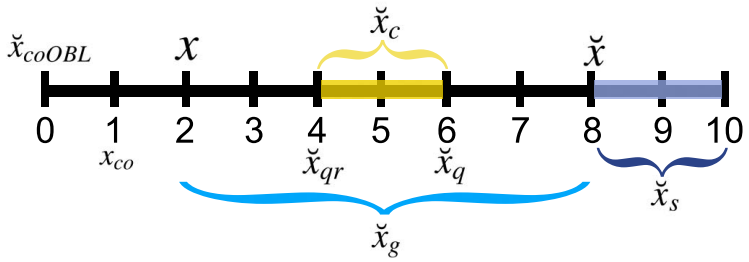


Figure 2.3: OBL types example

Figure 2.3 shows a minimization problem example that considers one dimension solutions ($M = 1$) and all the OBL types described. Let us suppose that $\mathcal{C}$ space is $\mathcal{R}$ and the candidate solution $x \in [0, 10]$, specifically $x = 2$. Also, let us suppose that the current best solution found is $x_{co} = 1$. Notice that the center of the domain is $\frac{x_{min} + x_{max}}{2} = 5$. Let us suppose that the Type-I opposite will be mapped using the $\Phi(x)$ function

$$\breve{x} = \Phi(x) = 10 + 0 - x \tag{2.5}$$

where 0 and 10 are the domain limits of x. In this case, $\breve{x} = \Phi(2)$ will be equal to 8. Then, the Type-I Quasi opposite can be $\breve{x}_q = 6$ and the Type-I Super opposite should be $\breve{x}_s \in (8, 10]$. Notice that for $\breve{x}_q = 6$ the constraint $d(2, 6) < d(2, 8)$ is satisfied. For $\breve{x}_s$, the constraint $d(\breve{x}_s, 2) > d(8, 2)$ should be satisfied.

Then, $\breve{x}_{qr}$ is a reflection of $\breve{x}_q$ across the center of the domain, $\breve{x}_{qr} = 4$. About CBS, $\breve{x}_c$ can be between 4 and 6, near to the center of the domain of x. GOBL defines the opposite $\breve{x}_g$ closer to the original candidate solution than $\breve{x}$. For this reason, $\breve{x}_g$ can take values from 2 to 8, depending of the random value of k (Equation 2.3). Considering that the current optimum is $x_{co} = 1$, CO_{OBL} is defined as $x_{CO_{OBL}} = 0$. Notice that Type-II opposite $\breve{x}_{II}$ cannot be represented without the definition of an evaluation function.

In the following we will present several applications of these OBL types metaheuristics. A brief review will be detailed for each approach, considering the four features already presented.

2.1.1 OBL Metaheuristics

Since 2005, several metaheuristic approaches related with Opposition-Based Learning have been proposed. Mainly, any of these OBL types were applied in metaheuristic like Differential Evolution, Particle Swarm Optimization, Evolutionary Algorithms and Harmony Search. In this section a

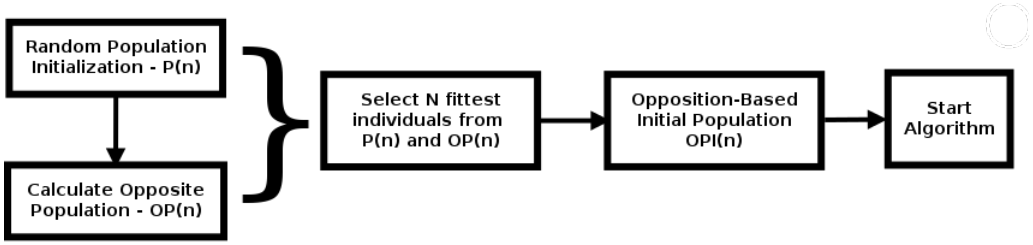


Figure 2.4: Opposite initialization procedure included in ODE

literature review of OBL metaheuristic approaches is presented.

2.1.1.1 Differential Evolution

Differential Evolution (DE) [SP97] is a population-based metaheuristic originally proposed for solving continuous problems. In a basic DE approach, a population of candidate solutions evolves considering mutation, crossover and selection operators. There are several DE algorithms that apply any of the described OBL types in their components. In general, the main objective of including OBL in Differential Evolution (DE) is to accelerate the convergence of the search process by improving the chance of getting better candidate solutions, by evaluating the original and their opposite candidate solutions simultaneously.

Opposite Differential Evolution (ODE) [RTS06, RTS07a] is a DE algorithm that includes two important opposite procedures: *opposite initialization* and *opposite generation jumping*. These schemes were proposed in [RTS06] and Type-I Opposition is considered in both procedures. Figure 2.4 shows details of the *opposite initialization* procedure. During *opposite initialization*, a set of random candidate solutions is created and then their opposites are computed. Initial population is then conformed by the best solutions selected from these two sets. Figure 2.5 shows details of the *opposite generation jumping* procedure. Similar to the *opposite initialization*, the *opposite generation jumping* also produces an opposite population, but in this case, during the execution of the algorithm. For each individual, its opposite is calculated with Type-I Opposition mapping rule (as in Equation 2.1). Considering an opposite and a normal population, the fittest individuals are kept to continue the search process. Here, the evolutionary process can be forced to jump to a fitter generation, with better quality individuals. The application of this procedure will be controlled by a parameter named Jumping Rate (JR).

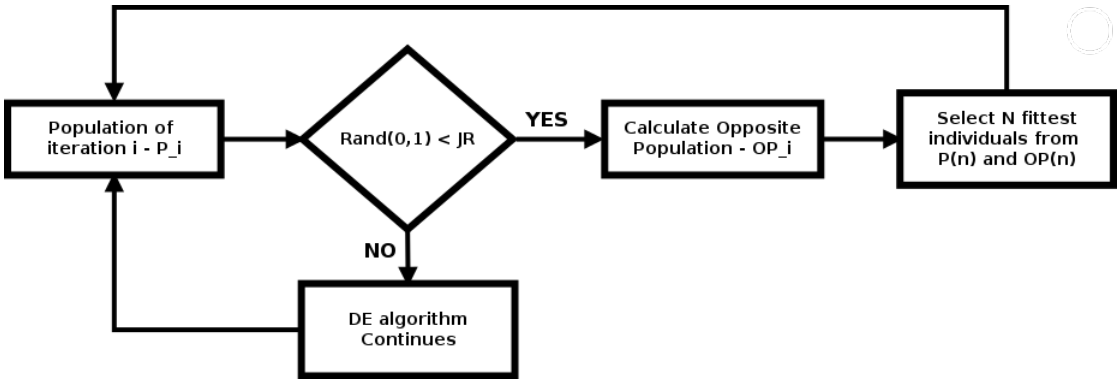


Figure 2.5: Generation Jumping method of ODE

To evaluate these procedures, experiments were performed using a test-suite of nine well-known benchmark functions (De Jong, Axis parallel hyper-ellipsoid, Rotated hyper-ellipsoid, Rosenbrock, Griewank, Sum of different power, One dimensional multimodal, Ackley and Rastrigin functions). Details of these benchmark functions can be found in [WRDM96]. First, they compare the inclusion of *opposite initialization* with a random initialization procedure. Results showed that the *opposite initialization* obtains better solutions than a randomly generated set. Then, both initialization processes were included in DE algorithm. Results showed that the convergence of ODE was accelerated in an average of 3%, in terms of the number of evaluation function calls. However, the algorithm improves the quality of the solutions found in an average of 30%. In the last scenario, *opposite generation jumping* and *opposite initialization* were included in DE. Results showed an overall improvement of 40% of average number of function calls [RTS06].

A performance evaluation of DE and ODE in large scale optimization problems is presented in [RW08]. For this, a set of benchmark functions proposed in a special session of CEC-2008 were used [TYS+07]. These functions are Shifted Sphere, Shifted Schwefels, Shifted Rosenbrocks, Shifted Rastrigins, Shifted Griewanks, Shifted Ackleys, and Fast Fractal DoubleDip functions. Results showed that ODE outperforms DE in these large scale optimization functions.

Finally, it is important to mention that these two opposite procedures have being severally applied in OBL metaheuristic approaches. When *opposite initialization* or *generation jumping* is mentioned in the following, we refer to these procedures.

Opposition-based Differential Evolution using the Current Optimum (COODE) is a variant of ODE. The structure and components of COODE are the same as ODE. The main difference is that opposite candidates are computed considering the *OBL using the Current Optimum* type instead of using Type-I Opposition. Experiments with 33 benchmark functions showed that COODE is

more efficient than ODE in terms of the number of evaluation function calls.

Quasi-Oppositional DE (QODE) [RTS07c] is a variant of ODE that considers the Type-I Quasi-Opposition instead of Type-I Opposition in their procedures. Results confirm that QODE outperforms ODE and DE in 15 benchmark functions. In [Bas16], an application of QODE is presented for the Optimal Reactive Power Dispatch problem. The objective of the reactive power dispatch problem is to minimize the active power loss and to improve voltage profile and voltage stability satisfying some constraints. The approach was tested on IEEE 30-bus, 57-bus and 118-bus test systems. Results showed that QODE outperforms other evolutionary techniques reported in literature such as DE, OGSA [SMG14], QOTLBO [MS15], SOA [DCZZ09], among others.

Generalized Opposition Orientation Neighborhood DE (GoOnDE) [Wan15] is an Enhanced Differential Evolution approach that includes two strategies based on Generalized-OBL (GOBL) and Orientation Neighborhood Mining (ONM). The idea is to increase the population diversity and the chance of finding the optimal solution using GOBL. For this purpose, a Generalized Opposite Population is generated, maintaining the fittest individuals from both populations. The algorithm was tested with seven well-known continuous benchmark functions and results showed that this algorithm outperforms another DE algorithms that include OBL or ONM.

There are some works only focused on the *opposite generation jumping* procedure. *Opposition DE with Time Varying Jumping Rate (ODE-TVJR)* is a variant of ODE that considers a variable jumping rate [RTS07b]. The generation jumping rate parameter changes linearly during the evolution, based on the number of function evaluations. Moreover, strategies for linearly increasing or decreasing the rate were proposed. Several functions and problems were used to evaluate this strategy (Rosenbrock, Rastrigin, Griewank, Perm, Michalewicz, Schwefel, Kowalik functions and Multi-Gaussian, Neumaier 2, Odd Square, Price's Transistor Modeling and Schaffer 2). Results showed that the linearly decreasing jumping rate performs better than constant settings and the linearly increasing strategy.

ODE-Protective Generation Jumping (*ODE-PGJ*) [ER11] is an ODE variant that adaptively uses the *opposite generation jumping* procedure. A parameter rate is defined to decide when the *opposite generation jumping* will be applied. The *opposite generation jumping* procedure will be used until the individuals selected from the opposite population is lower than a fixed percentage. To evaluate this strategy, twelve specific benchmark functions where ODE performs poorly were used (Rosenbrock, Rastrigin, Griewank, Perm, Michalewicz, Schwefel and Kowalik functions). Results showed that this strategy outperforms ODE in 10 out of 12 functions.

Shuffled Extended Opposition-Based DE (OB-SEOBDE) [AAR12] is a Shuffled Differential Evolution (SDE) approach that applies OBL to accelerate the convergence. Here, the population is divided into several partitions and each one is improved by a DE approach. Four different approaches were tested, combining *opposite initialization* and *opposite generation jumping*. These approaches were evaluated on 25 benchmark functions designed for a special session on real-parameter optimization [SHL$^+$05]. Encouraging results were found using these approaches. Their main disadvantage was the large number of parameter values to define. There is another variant of *OB-SEOBDE* extended for the Bidirectional SDE algorithm [Aha16].

In literature there are also DE algorithms proposed for solving Multi-objective optimization problems (MOP) that include OBL concepts. *Multiobjective Differential Evolution Based on Opposite Operation (MODEA-OO)* [DW09] is an ODE variation for MOPs. Here, in addition to using *opposite initialization*, the algorithm calculates opposite points of the offspring in each generation, according to the number of the non-dominated individuals. The algorithm was tested in *ZDT1* to *ZDT3*, *ZDT6* and *F1* functions [ZDT00].

Opposition-based Multiple Objective Differential Evolution (OMODE) [CT15] is a similar approach proposed for solving the Time Cost Utilization Work Shift Tradeoff (TCUT) problem. The aim of the TCUT problem is to minimize simultaneously the project duration, project cost and total evening and night shift working hours. Here, opposite initialization and generation jumping are included in OMODE.

Multiobjective Differential Evolution Algorithm (MODEA-OA) [LZY12] proposed another DE algorithm that includes an Opposition-Based parameter control mechanism. Here, OBL is used to accelerate the convergence of the search process. For this purpose, parameter configurations are characterized in three conditions: keep, flip and re-generate. If a configuration obtains good results, then it is in the keep condition, so it will be maintained. But, if a configuration does not obtain good results, it can be flipped using OBL. If that configuration was flipped in another iteration, the algorithm re-generates a new one. MODEA-OA was tested with problem functions with bi, tri and five-objectives. Results showed that MODEA-OA outperforms several MODEA algorithms in literature.

Table 2.2 shows a summary of these OBL-DE approaches. In general, Type-I Opposition is the most used type of OBL and most of these algorithms include OBL with the objective of accelerating the convergence of a DE approach. The most recurrent role when including OBL in DE the initialization of the algorithm and for the application of a generation jumping procedure.

Algorithm	Objective	Role	Type	Problem
ODE [RTS06, RW08], ODE-TVJR [RTS07b], ODE-PGJ [ER11], OB-SEOBDE [AAR12, Aha16], OMODE [CT15]	Accelerate Convergence	Initialization, Generation Jumping	Type-I Opposition	Continuous
COODE [XWHW11]	Accelerate Convergence	Initialization, Generate Solutions	OBL using the Current Optimum	Continuous
MODEA-OO [DW09]	Accelerate Convergence	Initialization, Generate Solutions	Type-I Opposition	Continuous
QODE [RTS07c, Bas16]	Accelerate Convergence	Initialization, Generation Jumping	Type-I Quasi-Opposition	Continuous
GoOnDE [Wan15]	Increase Exploration	Generate Solutions	GOBL	Continuous
MODEA-OA [LZY12]	Accelerate Convergence	Parameter Control	Type-I Opposition	Continuous

Table 2.2: Summary of Differential Evolution approaches that applies OBL

It is important to notice that all these applications are used for solving continuous optimization problems.

2.1.1.2 Particle Swarm Optimization

Particle Swarm Optimization (PSO) [KE95] is a stochastic swarm intelligence algorithm inspired in the social behavior lying in the bird flocking. In PSO, a population of individuals (referred as particles) are grouped into a swarm. Each particle represents a candidate solution and has its own position and velocity. Different rules are followed by each particle for moving with a certain acceleration. In the literature there are some PSO approaches that implement OBL ideas. Here, *opposite initialization* and *opposite generation jumping* were also implemented for PSO algorithms. For this, an opposite swarm from the regular swarm is computed using a numerical opposite position and opposite velocity. Fittest particles, belonging to both swarms, will be selected to be part of the next generation. This is used in the beginning of the execution process for the *opposite initialization* and during the execution for the *generation jumping procedure*.

Opposition-based Particle Swarm Optimization (OPSO) [HH07] considers the idea of opposition in three components: *opposite initialization*, *opposite generation jumping* and local improvement of the best individual in swarm. For the generation jumping, the procedure is repeated iteratively using a jumping rate and a dynamic constriction factor for enhancing the convergence speed. Experiments with well-known optimization functions (Rosenbrock, Rastrigin, Shubert, Beale, Ackley, Schaffer functions) showed that opposition-based ideas improve the performance of PSO.

Opposition based Particle Swarm Optimization (O-PSO) [JJB09] was proposed as the Opposition based initialization in PSO. In this case, the algorithm considers an opposite population only during the initialization. Fittest particles between both populations are considered for the execution of the algorithm. Experiments with Sphere, Rosenbrock, Ackley and MultiModal functions

showed that the opposite initialization procedure allows to improve the search process of standard PSO.

Opposition based Particle Swarm Optimization with Cauchy Mutation (OPSOCM) [WLL+07] is an extension of OPSO that includes a dynamic Cauchy mutation. The objective of this mutation is to avoid getting trapped in local optima. This dynamic Cauchy mutation is applied to the global best particle. To evaluate this approach, unimodal and multimodal functions were used. Results showed that OPSOCM presents a faster convergence and can find better quality solutions than PSO.

Another variant named *Generalized Opposition based Particle Swarm Optimization (GOPSO)* [WWR+11] includes Generalized OBL in addition to use Cauchy mutation. Generalized OBL considers a random number k as a weight to control the distance between a candidate solution and its opposite. Here, GOBL is used in the *opposite initialization* and *opposite generation jumping* procedures. To evaluate this approach, authors compared GOPSO with eight PSO variants from literature using unimodal, simple multimodal, unrotated and rotated multimodal benchmark functions [LQSB06]. Also, another version of GOPSO with dynamic population size is presented (DP-GOPSO). Large scale problems were used to compare GOPSO with their dynamic variant. Results showed that GOPSO was improved considering a dynamic population size. In [DKZ16], a complementary strategy to GOPSO was proposed, including an adaptive mutation selection strategy for performing a local search procedure to the global best particle. Also, an adaptive non linear inertia weight is included, with the objective of balance exploitation and exploration of the algorithm.

Other technique that includes Generalized OBL is named *Opposition based Personal Best PSO (OpbestPSO)* [SDB14]. In *PSO* algorithms, the movements of particles are guided by their personal best known positions and the best known position of the entire swarm. In this case, GOBL is used to obtain an opposite personal best position and for the initialization of the algorithm. The idea is that particles can learn about their opposite personal best positions before updating their velocities. To evaluate this strategy, uni and multimodal, quartic noisy and discontinuous step functions were used. Results showed that *OpbestPSO* outperforms *PSO* and *OPSO* algorithms.

Another similar Opposition based PSO approach is called *OPSO-Exp*, focused on exploration and exploitation [MS15]. As *OpbestPSO*, *OPSO-Exp* also uses GOBL for initialization and to obtain opposite particle personal best positions. In addition, *OPSO-Exp* includes a component for controlling exploitation and exploration levels considering the global best solution. To evaluate this strategy, 28 benchmark functions presented in CEC-2013 conference were used [LQSHD13].

Results showed that OPSO-Exp outperforms OPSOCM, OpbestPSO and PSO algorithms.

Enhanced Opposition-based PSO (EOPSO) [TZ09] is another PSO-OBL approach. Here, another type of OBL called Enhanced-OBL is proposed. An Enhance Opposite Particle is computed from a candidate solution using the Type-I Opposition definition and then, mapping $\breve{X}$ closer to the center of the domain. We considered this method as an hybrid between CBS and Type-I Opposition. The strategy is applied to obtain an enhance opposite population probabilistically controlled by a parameter named Opposition Probability. Experiments with well-known optimization functions showed that EOPSO outperforms OPSO and PSO, according to the quality of the solutions found.

Opposition-based PSO with Velocity Clamping (OVCPSO) is presented in [SBM+09]. Similarly to EOPSO, this algorithm considers opposite particles and an opposite population, but in this case, defined as Type-I Opposition. OVCPSO includes a velocity clamping component to manage swarms exploration and exploitation levels. Uni and multimodal functions were used to evaluate and compare OVCPSO with different variants of PSO that include velocity clamping components. Results showed that OVCPSO obtains better performance than the other two algorithms. However, OVCPSO was not able to find the optimum solutions for some test functions.

Particle Swarm Optimization with Opposition-based Disturbance (PSOOD) [CC10] tries to increase the population diversity and exploration in PSO. For this reason, the approach includes the Opposition-based Disturbance component, that changes the position of particles when the local best is updated. For testing this approach, multimodal functions were used and results showed that PSOOD outperforms PSO, OPSO and EOPSO algorithms.

Another PSO variant with a chaotic OBL population initialization procedure and a stochastic search technique is named *CSPSO* [GLH12]. Here, the random initialization is replaced by an opposition-based initialization with a sinusoidal iterator. Also, it considers a stochastic search strategy to increase the exploration of CSPSO search process. Experiments with 10 well-known benchmark functions with dimensions ranging from 30 to 300 showed that CSPSO performs better than seven other PSO variants from literature.

Multi-start Opposition-based Particle Swarm Optimization algorithm with Adaptive Velocity (MSOPSOAV) [Kau13] is a multi-start PSO variant for bound constrained optimization problems (BCOP). This algorithm considers an adaptive velocity strategy and includes OBL in the swarm initialization process. Also, to tackle the premature convergence, the search process is restarted re-initializing particles according to the Super-Opposition definition. To evaluate the performance of MSOPSOAV, 100 global optimization problems from literature were used. Results showed that

Algorithm	Objective	Role	Type	Problem
OPSO [HH07]	Accelerate Convergence, Handle Noisy Problems	Initialization, Generation Jumping, Local Improvement	Type-I Opposition	Continuous
OPSOCM [WLL+07], OVCPSO [SBM+09]	Accelerate Convergence	Generate Solutions	Type-I Opposition	Continuous
GOPSO [WWR+11, DKZ16]	Accelerate Convergence	Initialization, Generation Jumping	GOBL	Continuous
OPSO-Exp [MS15], OpbestPSO [SDB14]	Accelerate Convergence	Initialization, Generate Solutions	GOBL	Continuous
EOPSO [TZ09]	Accelerate Convergence	Generate Solutions	Type-I Opposition + CBS	Continuous
O-PSO [JJB09], CSPSO [GLH12]	Accelerate Convergence	Initialization	Type-I Opposition	Continuous
PSOOD [CC10]	Increase Exploration	Generate Solutions	Type-I Opposition	Continuous
MSOPSOAV [Kau13]	Accelerate Convergence, Increase Exploration	Initialization, Restart	Type-I Super Opposition	Continuous

Table 2.3: Summary of Particle Swarm Optimization approaches that applies OBL

OBL initialization and super-opposition based restart strategy prevent the premature convergence and stagnation of PSO.

Table 2.3 shows a summary of these PSO approaches that includes ideas from OBL. In general, the most recurrent objective of including OBL is to accelerate the convergence speed of PSO algorithms. For this purpose, OBL was applied mostly in initialization processes and for generating opposite particles during the search process. Again, all these approaches were implemented for solving continuous optimization problems.

2.1.1.3 Evolutionary Algorithms

Evolutionary Algorithms (EA) [BFM97] are population-based algorithms inspired in Darwin's principles of evolution. A population of individuals evolves considering transformation operators and a natural selection procedure (based in the fitness of individuals). Most used EAs are Genetic Algorithms, Memetic Algorithms, Genetic Programming, among others.

The first OBL proposed approach was incorporated to a Genetic Algorithm named *Opposition GA (OGA)* [Tiz05]. The idea here was to generate anti-chromosomes for best and worst members of the population. To obtain these anti-chromosomes, binary genes are inverted by a *total mutation* procedure. Results in unimodal functions showed how interesting the use of opposite knowledge can result during the search process of an algorithm. Finally, *total sub-mutation* was proposed for inverting only specific parts of the chromosome.

Oppositional Biogeography-based Optimization (OBBO) [ESD09] is a technique that includes a Quasi-Reflection OBL in the standard Biogeography-based Optimization (BBO). [2] Here, *opposite*

[2]Biogeography-based Optimization [Sim08] is an EA inspired by biogeography and the migration of species in

initialization and *opposite generation jumping* were implemented. The objective of the inclusion of opposite information is to accelerate the convergence process of the standard BBO algorithm. Results in sixteen benchmark functions showed that OBBO outperforms BBO in terms of the successful runs and of the number of evaluation function calls made.

In the field of Multi-Objective Optimization Evolutionary Algorithms (MOEA), one OBL approach was published. A MOEA/D variant that includes OBL for the initialization and for generating new chromosomes during the evolutionary process is named *MOEA/D-OBL* [MLQ+14]. Center-Based Sampling is used for the initialization of the population. An opposite population is calculated and one of these solutions replaces a solution from the original population. To generate new chromosomes, Generalized-OBL is implemented in a strategy called opposition-based local search. A solution is mapped considering the maximum and minimum values of its neighbor solutions. MOEA/D-OBL was evaluated using four representative test problems including ZDT, DTLZ, UF and WFG problems. Results showed that MOEA/D-OBL outperforms the well-known MOEA/D.

Opposition-Based Memetic Algorithm (OBMA) [ZHD17] is an MA approach for solving the Maximum Diversity Problem (MDP). In MDP, considering a set of N elements and a distance d_{ij} between each pair of elements, the idea is to select a subset S of elements with the objective of maximizing the total distance between all of them. OBMA uses a binary representation to define which elements will be selected. For its initialization, it simultaneously considers opposite candidates for each randomly generated candidate solution. Then, both solutions are improved using a Tabu Search procedure. To obtain an opposite candidate $\breve{X}$, the complement in each dimension i of the candidate solution X is calculated. The solution with best quality value will be included in the initial population. Also, OBMA includes a component named Opposition-based Double Trajectory Search, that searches simultaneously in the neighborhoods of an offspring solution and its opposite. Results in 80 large benchmark instances, considering from 2000 to 5000 items, show the competitiveness of OBMA obtaining best-known solutions in most instances. Also, new boundaries were obtained in 22 instances.

There are other EA-OBL approaches in literature: a Hybrid Parallel EA based on STS (HPEA) [DYWC10], a GA for the Multi-Unmanned Combat Aerial Vehicle problem (OGA-M) [WLWK15] and a Balanced Cartesian Genetic Programming approach with a new mutation operator inspired in OBL (BCGP) [YS15].

evolution. Each solution represents an island with an immigration and emigration rates based in its fitness. Each variable represents a species that lives in that island. Species migrate among islands based on the immigration and emigration rates to find a better habitat.

Algorithm	Objective	Role	Type	Problem
OGA [Tiz05]	Accelerate Convergence	Generate Solutions	Type-I Opposition	Continuous
OGA-M [WLWK15]	Accelerate Convergence	Generate Solutions	Type-I Opposition	Discrete
OBBO [ESD09]	Accelerate Convergence	Initialization, Generate Solutions	OBL using CO	Continuous
MOEA/D-OBL [MLQ+14]	Accelerate Convergence	Initialization, Generate Solutions	CBS - GOBL	Continuous
HPEA [DYWC10]	Accelerate Convergence	Generate Solutions	GOBL	Continuous
BCGP [YS15]	Increase Exploration	Mutation	Type-I Quasi Opposition	Continuous

Table 2.4: Summary of Evolutionary Algorithms approaches that apply OBL

Table 2.4 shows a summary of these EA-OBL approaches. The most recurrent objective of including OBL in an EA algorithm is to accelerate its convergence. For this reason, OBL was applied mostly for generating opposite solutions during the search process. GOBL, CBS, Type-I Opposition and Type-I Quasi-Opposition were the types implemented in EA approaches. Moreover, GOBL and Type-I Opposition were the more recurrent options. Finally, OGA-M is the only approach that was implemented for solving a discrete optimization problem. The other four approaches were proposed to solve continuous optimization problems.

2.1.1.4 Harmony Search

Harmony Search (HS) [Gee00] is a population-based metaheuristic inspired in the improvisation process of jazz music. Each harmony represents a candidate solution and different components (random selection, memory consideration and pitch adjustment) are considered to construct and perturb harmonies. In literature, there are some HS approaches that implement OBL ideas.

Opposition Based HS (OHS) [CGM12] is an approach for solving the Combined Economic and Emission Dispatch (CEED). Here, *opposite initialization* and *opposite generation jumping* are implemented. For the initialization, opposite candidates are obtained from the randomly generated candidates. Here, an opposite candidate is computed using a function that uses the boundaries of the domain variable. Results showed that OHS successfully solved several CEED problems and its performance is comparable to state-of-the-art algorithms for this problem. There are other applications of OHS for solving engineering optimization problems [BMG14, UKM+14].

Dynamic Regional HS (DRHS) [QF11] is a HS approach that includes OL with the objective of avoiding stagnation. It considers an opposition-based initialization of the half of Harmony Memory (HM). Here, a function is used to map continuous variables to their opposite value. Moreover, for searching solutions in other sub-regions of the space, DRHS splits the HM into multiple groups. In each group, when a solution is constructed, its opposite is also calculated. Results in uni and multimodal test problems showed that DRHS outperforms HS.

An improved approach of the Global-best HS named *IGHS* is presented in [XAL+14]. Here, an

Algorithm	Objective	Role	Type	Problem
DRHS [QF11], IGHS [XAL+14]	Avoid Premature Convergence	Initialization, Generate Solutions	Type-I Opposition	Continuous
OHS [CGM12, BMG14, UKM+14]	Accelerate Convergence	Initialization, Generation Jumping	Type-I Opposition	Continuous
ACHS [NZW+14]	Increase Exploration	Perturb Solutions	Type-I Opposition	Continuous
QOHSA [SM16, SSM15]	Accelerate Convergence	Initialization, Generation Jumping	Type-I Quasi-Opposition	Continuous

Table 2.5: Summary of Harmony Search approaches that applies OBL

initialization opposition-based procedure is proposed, using a function to map continuous variables to their opposite value. Also, modifications of the original Global-best HS (GHS) are proposed: a new improvisation scheme based on differential evolution, on-line modification of the HMCR and PAR parameters, a modified *Randomization* component and, two perturbation schemes to avoid premature convergence. Results in twenty-eight benchmark continuous functions showed that IGHS outperforms the original HS and GHS and also, is competitive with population based state-of-the-art algorithms like Artificial Bee Colony (ABC) and Teaching-Learning-Based Optimization algorithm (TLBO).

Harmony Search with Arithmetic Crossover (ACHS) [NZW+14] was proposed for solving the Economic Dispatch problem. Here, Type-I Opposition concept is employed to enhance diversity of solutions modifying the Pitch Adjusting component. Results in unimodal functions showed that ACHS outperforms several HS variants and some well-known algorithms from literature.

In electric engineering, an HS variant that uses Type-I Quasi-Opposition was presented *(QOHSA)* [SM16]. Here, Quasi-Opposite concept is used to initialize the Harmony Memory and for an *opposite generation jumping* procedure. QOHSA was proposed for the study of load frequency control (LFC) of power systems. In [SSM15], another application of QOHSA in electric engineering is presented, specifically for automatic generation control of power systems.

Table 2.5 shows a summary of these Harmony Search approaches that includes OBL. In this case, the acceleration of convergence of Harmony Search algorithm was the main reason for including OBL. There is not a clear tendency related to the role of OBL in Harmony Search. On the other hand, the Type-I Opposition was the most used type of OBL. As in PSO and DE approaches, all these HS approaches were implemented for solving continuous optimization problems.

2.1.1.5 Other Metaheuristics

In our knowledge, *Opposition-based Simulated Annealing (OSA)* [VT07] is the only Simulated Annealing based in OBL that has been reported. OSA algorithm proposes to generate opposite neighbors during the search process. At each iteration, an opposite neighbor can be calculated according to a probabilistic criterion. To obtain a neighbor, a fixed number k is summed to a con-

tinuous candidate solution. Then, to obtain an opposite neighbor the same number k is subtracted. In this case, an hybrid between Type-I and Type-II Opposition types was considered. Moreover, this work presents theoretical foundations of OBL, specifically related to opposite mapping and opposite evaluation functions. Here, the opposition mapping is focused on obtaining near candidate solutions considering implicit boundaries of the neighborhood. To evaluate OSA, continuous unimodal functions were used and results showed that SA is improved using the opposite neighbors idea.

Novel Artificial Bee Colony (NABC) is an ABC approach that includes an Opposite Learning initialization mixed with the Tent Chaotic Map (TCM) strategy [WWAS17]. Here, NABC is used for optimizing the input weights and biases of a Extreme Learning Machine (ELM). ELM technique is a single-hidden layer feed forward Neural Network used for classification and, NABC-ELM is here proposed. For the initialization of NABC, SN candidate solutions are generated using the TCM strategy and their opposite are calculated considering the domain of each variable. Also, a local search procedure based in TCM is used for scout bees with the objective of escaping from local optima. Datasets obtained from UCI machine learning repository are used to evaluate NABC-ELM. Results showed that NABC-ELM outperforms other ELM approaches optimized with other metaheuristics (PSO, Gbest-guided ABC, Inspired-ABC, ABC, among others).

Other ABC approach is the Grey Artificial Bee Colony (GABC), proposed for solving global optimization of numerical function problems [XLM$^+$17]. Here, Opposite Learning and TCM is used for its initialization. Also, GABC includes a Grey Regional Degrees scheme to guide the search process. For this, an individual is selected as a current employed bee to guide the search process. To evaluate this approach, fifty-seven continuous benchmark functions were used considering multimodality, unimodality, nonseparability, among others. Results show that GABC is competitive to other well-known Artificial Bee Colony and Differential Evolution variants.

2.2 Opposition-Inspired Learning

As mentioned early, the concept of opposite can be applied in several situations. If we analyze the definitions of the different types of OBL, an opposite solution is a candidate solution mapped from an original one, that will be in the same domain of the original solution. However, there are some situations where the concept of opposition cannot be directly related to mapping solutions. Opposition-Inspired Learning (OIL) [RRM17] was proposed considering that mapping solutions is not intuitive because of some algorithm-specific properties. A clear example of this situation is

how any of the already explained OBL types can be applied to Ant Colony Optimization (ACO) algorithms. Since decisions are stochastically made during the construction process, obtaining an opposite complete instantiation is not straightforward as applying a mapping function. In these situations, the term *opposite* has been particularly defined for each proposed strategy in literature. For example, the decisions made by an algorithm can be modified to their opposite instead of applying any of these OBL definitions. In this section we present a brief review of the existing approaches that do not directly apply any definition of OBL but were inspired by opposition-based concepts. First, OIL ants based algorithms are presented and then, some evolutionary algorithm approaches are described.

2.2.1 Ant-based algorithms

Ant-based algorithms are population-based algorithms inspired in the behavior and communication of real ants. Some early proposed approaches can be related to the idea of including opposite information into ant-based algorithms, specifically into the pheromone management. Schoonderwoerd introduced the concept of *anti-pheromone* proposing the idea that ants would decrease pheromone rather than reinforce it [SBHR97].

In [MR02], three approaches of anti-pheromone structure were proposed for an Ant Colony System (ACS) algorithm for solving the Traveling Salesman Problem (TSP). The objective of studying these structures was to improve the exploration of the search space. The approaches were: (1) *Subtracting Anti-Pheromone* (SAP), subtracting pheromone from the elements that worsen the solutions, (2) *Preferential Anti-Pheromone* (PAP), using two pheromones matrices: one for good solutions and one for bad solutions; and (3) *Explorer Ants*, that are interested in regions with lower levels of pheromone.

Another related algorithm is *Best-Worst Ant System (BWAS)* [CdVHM00]. In this case, the best so far and current worst solutions are considered respectively to perform positive and negative pheromone updates. Also, they include a mutation operator for introducing diversity into the pheromone matrix. Eight symmetric TSP instances were used to evaluate and compare BWAS with Ant Solver (AS) and ACS. Results showed that BWAS outperforms both algorithms. Moreover, a study of the application of BWAS for QAP is presented in [CdVH02]

In 2007, Malisia proposed the first Ant Colony System approach with an extension of OBL for solving the TSP [MT07, Mal07]. Here, the idea of opposition is related with increasing the coverage

of the solution space. In this work, authors propose to include OBL during their construction phase: changing the decisions or their main components (pheromone or heuristic).

Three extensions were presented for modifying the decisions made during the construction process: Synchronous Opposition (SO), Free Opposition (FO) and Free Quasi-Opposition (FQO). These algorithms considered that each ant must be paired: a *leading ant* and an *opposite ant*. The *leading ant* constructs solutions like in ACS. At each step, the leading ant selects each city based on its rank, while the *opposite ant* selects the city with the opposite rank of the leading ant. These algorithms differ in the decision made when both ants are in the same city.

Other two extensions are related to the pheromone management: Opposite Pheromone per Node (OPN) and Opposite Pheromone per Edge (OPE). When the ant k is constructing a solution, in both approaches, altered pheromone values can be considered. These altered values try to make farther cities more desirable in order to increase the exploration of the algorithm. Both approaches differ in the way the pheromone matrix is altered. In OPN, considering a pheromone matrix τ, an opposite pheromone matrix $\breve{\tau}$ is computed by the ant k as:

$$\breve{\tau} = \tau_o + \frac{1}{L_{bs}} - \tau \tag{2.6}$$

where τ_o is the initial pheromone value and L_{bs} is the length of the best-so-far path. This matrix is computed to decide which city will be visited next by the ant k. The decision of modifying pheromone values is controlled by a parameter $\breve{\lambda}$.

On the other hand, OPE can consider opposite pheromone values only on some edges of the current decision of ant k:

$$\tau_{ij} = \begin{cases} \breve{\tau_{ij}} = \tau_o + \frac{1}{L_{bs}} - \tau_{ij} & \text{if } \lambda \leq \breve{\lambda} \\ \tau_{ij}, & \text{otherwise} \end{cases} \tag{2.7}$$

where $\lambda \in [0, 1]$ is a uniform random number.

These five extensions were evaluated using four instances of TSP and compared to the original ACS (*eil51, eil76, kroA100* and *d198*). Results showed that OPN outperforms all the extensions and ACS in 3 out of the 4 instances.

In [Mal08], OPN was revisited and a pheromone update procedure called Opposite Pheromone Update (OPU) is presented. After pheromone is updated, for each component of a given solution, outgoing edges are ranked. Pheromone is deposited or removed from opposite edges, depending

Algorithm	Objective	Role	Problem
GA_{PSP} [BB16]	Accelerate Convergence	Perturb Solutions	Discrete
ONSGA-II [TDR16]	Increase Exploration, Accelerate Convergence	Generate Solutions	Continuous
OPU [Mal08], OPN,OPE,FO,SO,FQO [MT07, Mal07], ACON [YZNL17] BWAS [CdVHM00, CdVH02], SAP, PAP, Explorer Ants [MR02]	Increase Exploration	Generate Solutions	Discrete

Table 2.6: Summary of Opposition Inspired Learning metaheuristics approaches

on what kind of ant algorithm is used and on the problem being solved. Again, the objective of OPU is to improve the exploration of ACS. Nine symmetric TSP instances were used to evaluate and compare OPN with ACS. Also, OPU was compared to AS using instances of the Grid World Problem. Results showed that these approaches outperform ACS and AS, respectively.

ACON is an ant-based algorithm with a transition rule and a pheromone management that considers information from low quality solutions [YZNL17]. Here, an alternative pheromone matrix is used to obtain information about the worst quality complete instantiations of each interation. This information is used in a transition rule to decrease the attraction to edges that are related with this unpromising candidates. *ACON* also considers a typical pheromone matrix where best ant deposits pheromone in each iteration. On the other hand, heuristic knowledge represents the effect of a candidate assignment in terms of its produced conflicts. Experiments were made using random binary CSP instances and N Queens instances considering different number of queens ranging from 4 to 700. Here, *ACON* solved all the N Queen instances, but only solved CSP instances with the lowest κ values (near to $0.70 - 0.80$).

2.2.2 Evolutionary Algorithms

There are some Evolutionary Algorithms that include an OIL component. GA_{PSP} is a Genetic Algorithm proposed for the Protein Structure Prediction problem (PSP) [BB16]. Here, a different concept of opposition is proposed considering an opposite-based mechanism that inverts sequences of proteins for obtaining new candidate solutions. It is important to remark that the opposition here inverts amino acid sequences considering their specific properties. The hypothesis points out that the algorithm can repair inverted sequences than the original ones more easily. Results show that GA_{PSP} outperforms evolutionary and swarm algorithms of the state-of-the-art.

The notions of *Opposite Convergence* and *Opposite Diversity* were proposed in an EA algorithm

for solving MOPs named *ONSGA-II* [TDR16]. About convergence, a solution that is far from the true Pareto-front (PF) is defined as *opposite* to any solution that is closer to the true Pareto-front. About diversity, authors defined that an isolated solution on the true PF is opposite to a crowded solution. The idea is to deterministically generate opposite candidate solutions during the search process using both concepts. For this, it is necessary to define extreme points of the true PF. These extreme points are the pivots between opposites and original solutions. In each generation, 25% of the best individuals from the current population are used to generate opposite solutions. This strategy was included into *NSGA-II* and tested with 2-objective well-known problems.

Table 2.6 presents a summary of these OIL metaheuristic approaches. Unlike the applications of OBL in metaheuristics, most of these approaches were implemented for solving discrete optimization problems. As these approaches do not apply any type of OBL, the columnn type was not considered.

2.3 Classification

In this section we propose a classification of metaheuristics that are related with Opposition-based Learning. As we mentioned earlier, we considered two main areas: Opposition-Based Learning and Opposition-Inspired Learning. Figure 2.6 presents our proposed classification. Considering OBL metaheuristic approaches, the most intuitive and useful characteristic to classify them is which type of OBL is applied: Type-I Opposition, Type-I Quasi-Opposition, Type-I Super-Opposition, Type-II Opposition, Generalized-OBL, Center-Based Sampling, Quasi-Reflection OBL or OBL using Current Optimum.

On the other hand, about OIL metaheuristic approaches, we first classify them considering when the opposite information is applied: *On-line* or *Off-line*. In the case of On-line approaches, the learning takes place when the algorithm is solving a problem and Off-line approaches consider a previous learning step. On-line approaches can consider the opposite information *Deterministically* or *Stochastically*. The idea is to analyze if the inclusion of the opposite information is considered at every iteration or when an stochastic condition is satisfied. Finally, we analyzed the effect of using the opposite information. We observed that the inclusion of opposite information can allow algorithms to *Focus* or *Disperse* the search process, visiting more regions of the search space. For example, the GA_{PSP} algorithm can be classified as: Inspired in OBL, On-line, Deterministically applied and to Disperse the search process.

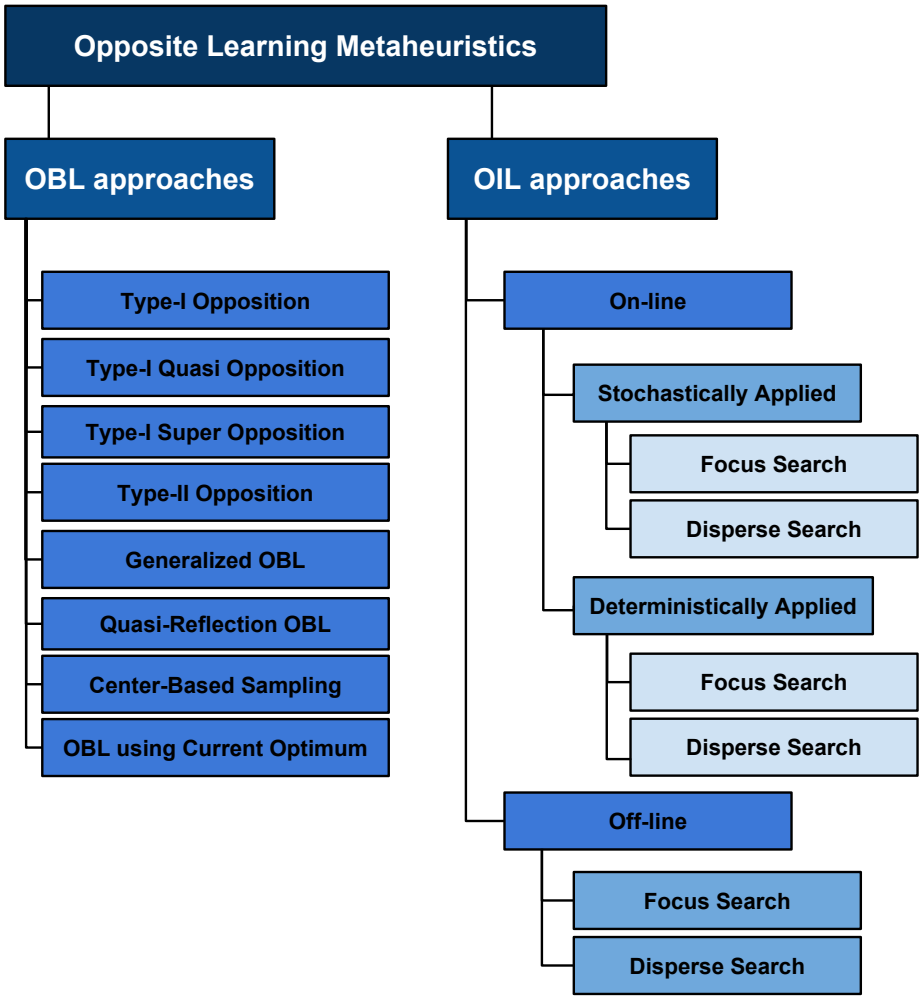


Figure 2.6: Opposition-Based Metaheuristics classification

2.4 Discussion

In this chapter several Opposite Learning metaheuristic approaches were reviewed. Most of these algorithms apply directly any of the detailed types of Opposition-Based Learning. The other approaches are algorithms that were inspired in the opposition-learning concepts. Differential Evolution and Particle Swarm Optimization are the most preferred metaheuristics selected for applying directly any of the definitions of OBL. On the other hand, Ant Colony Optimization was the most selected metaheuristic in OIL.

The most recurrent objective of including OBL was to accelerate the convergence of these approaches. For this reason, the most frequent uses were the initialization, generation jumping and generating solutions. As the idea of including OBL is for improving a metaheuristic performance, these results show two important remarks: (1) the speed of convergence is an important attribute that can be improved using OBL and, (2) focusing the attention on the initialization and the search process, the convergence speed can be improved including an OBL component.

Analyzing the types of Opposition-Based Learning, the Type-I Opposition was the most used, implemented in twenty four out of the thirty seven applications. This situation shows that maybe there is not a solid argument to consider other opposite candidate solutions from other areas of variables domains. On the other hand, the need for proposing other seven OBL types shows that these concepts can be highly modifiable and adaptable to different situations.

About the type of problem solved by each approach, most of these approaches were implemented for solving continuous optimization problems. On the other hand, OIL approaches were implemented mostly to tackle discrete optimization problems. This shows that the study of opposite candidates for integer and discrete variables can be considered as a future trend of research [MRD17]. Also, it reflects that opposite-learning concepts can be applied in several other scenarios. In general, the key idea is to detect which component or decision of an algorithm could be improved and analyze how opposition information can be used.

Table 2.7 shows our proposed classification for all the approaches reviewed in this chapter. It is important to notice that Off-line (OIL approaches, (10)) and approaches that include opposite information for Discard Solutions (Objective, (5)) are not listed in the table. These classes were included into the classification because the approaches proposed in this thesis consider an Off-line learning step and for discard components of candidate solutions.

Finally, a paper entitled “A survey and classification of Opposition-Based Metaheuristics” was published with the contents presented in this chapter [RRM17].

Approach	OBL directly applied								Inspired on OBL						Objective					Role							
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	1	2	3	4	5	1	2	3	4	5	6	7	8
ODE [RTS06, RW08], ODE-TVJR [RTS07b], ODE-PGJ [ER11], OBSEOBDE [AAR12, Aha16], OMODE [CT15]	X														X					X	X						
OSA [VT07]	X			X											X							X					
COODE [XWHW11]								X							X					X	X						
OBBO [ESD09]						X									X					X	X						
MODEA-OO [DW09]	X														X					X		X					
QODE [RTS07c, Bas16]		X													X					X	X						
GoOnDE [Wan15]					X											X						X					
MODEA-OA [LZY12]	X														X								X				
OPSO [HH07]	X														X		X			X	X			X			
OPSOCM [WLL+07], OVCPSO [SBM+09]	X														X							X					
GOPSO [WWR+11, DKZ16]					X										X					X	X						
OPSO-Exp [MS15], OpbestPSO [SDB14]					X										X					X		X					
EOPSO [TZ09]	X						X								X							X					
O-PSO [JJB09], CSPSO [GLH12]	X														X					X							
PSOOD [CC10]	X															X						X					
MSOPSOAV [Kau13]			X												X	X				X					X		
OGA [Tiz05], OGA-M [WLWK15]	X														X							X					
MOEA/D-OBL [MLQ+14]					X		X								X					X		X					
HPEA [DYWC10]					X										X							X					
BCGP [YS15]		X														X										X	
DRHS [QF11]	X																	X		X		X					
OHS [CGM12, BMG14, UKM+14]	X														X					X	X						
ACHS [NZW+14]	X															X											X
QOHSA [SM16, SSM15]		X													X					X	X						
GA_{PSP} [BB16]									X			X		X	X												X
ONSGA-II [TDR16]									X			X		X	X	X						X					
OPU [Mal08]									X		X		X			X						X					
OPN,OPE [MT07, Mal07]									X		X			X		X						X					
FO,SO,FQO [MT07, Mal07], Explorer Ants, PAP [MR02]									X			X		X		X						X					
BWAS [CdVHM00], SAP [MR02]									X			X	X			X						X					

Table 2.7: Classification of Opposite Learning metaheuristic approaches

OBL directly applied: (1) Type-I Opposition (2) Type-I Quasi Opposition (3) Type-I Super Opposition (4) Type-II Opposition (5) GOBL (6) Quasi-Reflection OBL (7) CBS (8) OBL using CO - **Inspired on OBL:** (9) On-line (10) Off-line (11) Stochastically applied (12) Deterministically applied (13) Focus search (14) Disperse search - **Objective:** (1) Accelerate Convergence (2) Increase Exploration (3) Handle Noisy Problems (4) Avoid Premature Convergence (5) Discard Solutions - **Role:** (1) Initialization (2) Generation Jumping (3) Generate Solutions (4) Parameter Control (5) Local Improvement (6) Restart (7) Mutation (8) Perturb Solutions.

Chapter 3

Ant Colony Optimization

Ant Colony Optimization (ACO) [Dor92] is a metaheuristic that represents the analogy with the organization and communication of colonies of real ants. Real ants are social insects highly structurally organized, that respond to different stimuli of the environment. They deposit a chemical substance named pheromone that perturbs the environment producing a stimulus. Pheromone is used to collaborate and communicate to the colony, for example, a path from their nest to a food source. In this chapter a description of the behavior of real ants is first presented in Section 3.1.

This social and collaborative behavior is represented with artificial ants searching solutions for a problem. A problem can be represented as a graph of a finite set of solution components. Ants perform randomized walks on this completely connected graph constructing candidate solutions. Pheromone is deposited on paths that are most promising to be visited by the colony. Pheromone is used as a communication method among the ants in the colony, to guide the search process towards interesting regions of the search space. To decide the path to be visited, pheromone information and heuristic knowledge are considered. Heuristic knowledge measures candidate vertices considering specific information about the problem being tackled. Section 3.2 presents the details of how the behavior of real ants has been represented in a metaheuristic.

In this chapter, most important variants of Ant Colony Optimization are detailed: Ant System, Ant Colony System, $\mathcal{MAX} - \mathcal{MIN}$ Ant System and Ranked Ant System. These ant-based algorithms were the first proposed in literature and during the last decades, most ants approaches and applications are based on these algorithms. The structure of these algorithms differ mainly in how pheromone is managed, which ant will be allowed to deposit pheromone, among other characteristics. Here, the main components of each algorithm will be presented. As most of these

ant-based algorithms were designed for solving the well-known Traveling Salesman Problem (TSP), we present their components as they were proposed for this problem. However, these components can be defined and re-designed for solving other combinatorial problems. Section 3.4.3 present how ant-based algorithms have been applied to other combinatorial problems like Quadratic Assignment Problem (QAP), Multidimensional Knapsack Problem (MKP), Constraint Satisfaction Problems (CSP), among others. Finally, as the concept of *anti-pheromone* that was already described in Section 2.2.2 was important for the motivation of our work, we present more details of these ideas in this chapter.

3.1 Real Ants going from Food to Nest

An ant colony is a highly structured group of ants with self organizing principles that involve cooperation and communication among them. As there exist blind ant species, their communication is produced via a mechanism named *stigmergy* related to mediate animal-animal interactions. To communicate among them, modifications in the environment are generated to produce a stimulus for the other individuals of the group [TB99]. More specifically, a chemical substance called *pheromone* is deposited by ants to mark paths on the ground. The objective is to indirectly communicate to the other ants in the colony something in particular (e.g., presence of food nearby). An isolated ant walks almost randomly because no previously deposited pheromone amounts are on the ground. Then, this new trail can be detected by another ant and it will bias its walk.

The concentration of this chemical substance will modify the probability of response of the individuals to these stimuli. The pheromone evaporates after a while and paths with lower amounts of pheromone will be less interesting to be visited by ants. The evaporation allows the colony to slowly "forget" paths that were not interesting to be visited. When pheromone is more concentrated in a particular path, ants are attracted to walk through that way.

The pheromone trail-laying has been investigated in controlled experiments by several researchers. Figure 3.1 shows the *Binary Bridge* experiment [DAGP90]. Here, the nest is separated from a food source by a bridge with two equally long branches (1) and (2). Moreover, at the beginning, branches are free from any pheromone. From this experiment, a stochastic model that represents the path selection process was proposed [DAGP90, GADP89]:

$$P_{(1)}(t+1) = \frac{(c + n_{(1)}(t))^{\alpha}}{(c + n_{(1)}(t))^{\alpha} + (c + n_{(2)}(t))^{\alpha}} = 1 - P_{(2)}(t+1) \tag{3.1}$$

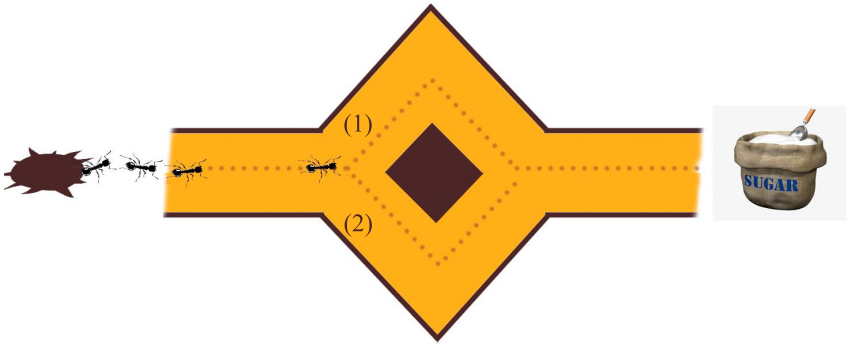


Figure 3.1: Binary bridge experiment

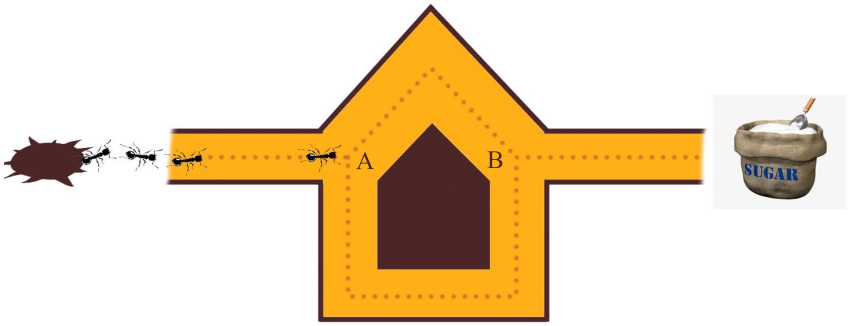


Figure 3.2: Altered binary bridge experiment

where $P_{(1)}$ and $P_{(2)}$ are the probabilities of the next ant to choose path (1) and (2) at time step $t+1$ respectively, $n_{(1)}$ and $n_{(2)}$ denote the number of ants on path (1) and (2) respectively, c quantifies the degree of attraction of an unexplored branch, α is the bias to using pheromone deposits in the decision process. This model assumes that the amount of pheromone on a branch is proportional to the number of ants that selected this branch in the past. Finally, evaporation was not considered in the model because it produces too small changes in the pheromone amounts. However, the evaporation component in artificial ants was important to avoid stagnation, so authors decided to finally maintain it in the design of the algorithm.

Figures 3.2 and 3.3 illustrate an altered version of the binary bridge experiment [GADP89]. Here, one of the branches of the bridge was longer than the other. In this experiment the behavior of a particular species of ants named *Iridomyrmex humilis* was studied. These ants were capable to select the shortest route while walking from a food source to their nest and vice versa. In both figures, each ant has to decide which path to continue in A and B points. Figure 3.2 shows an initial situation where no pheromone has been deposited. In this case, ants do not have a preference and paths are selected with the same probability for any of the branches. Ants start to

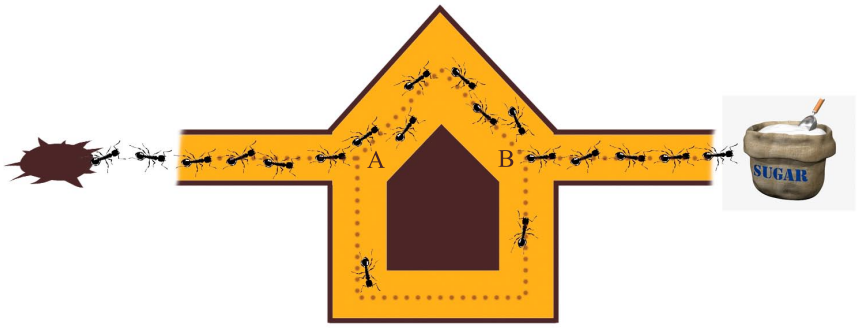


Figure 3.3: Shortest path marked with pheromone

produce a *autocatalytic behavior* [Dor92] (also known as positive feedback) depositing pheromone preferentially on the shortest branch. Figure 3.3 shows an advanced situation where pheromone has biased the decision of the colony to follow the shortest path. Also, some ants choose the larger branch performing a "path exploration".

The social behavior of real ants, its organization structure and the optimization ability to find the shortest possible path inspired the idea of designing artificial ants to solve combinatorial problems. The stigmergy mechanism will be represented by artificial pheromone to communicate to the other ants which components are more promising to be considered. In the following section we present the details of Ant Colony Optimization metaheuristic.

3.2 Artificial ants in a Metaheuristic

Ant Colony Optimization (ACO) is a metaheuristic algorithm designed and proposed for solving Combinatorial Optimization problems. To this day, there exist several variants and applications of this algorithm in continuous or discrete optimization, constraint satisfaction, real-world application problems. As it is mentioned in [DS04], an ACO algorithm can be applied to any combinatorial optimization problem for which a constructive heuristic can be defined. This section describes common components that are part of most existing ant-based algorithms. The objective of this section is to present details of how a combinatorial problem can be represented by a graph, how artificial ants construct solutions and some characteristics of pheromone management. Further details of each variant will be presented in following sections of this chapter.

3.2.1 A combinatorial problem into a Construction Graph

To define and explain ACO, we will consider a combinatorial problem P that we want to solve. This problem will be represented by a graph structure where ants will walk through and construct candidate solutions. First, we will define a Combinatorial Problem P that will be solved by an ant-based algorithm.

Definition 3.2.1. Combinatorial Problem Let P a Combinatorial Problem defined as a 4-tuple $P = (X, D, \Omega, F)$ where:

- X is the set of variables $X = \{x_1, x_2, \ldots, x_n\}$,
- D is the set of variable domains $D = \{D_1, D_2, \ldots, D_n\}$,
- Ω is the set of constraints among variables,
- F is an objective function F to be optimized.

The set of all possible complete instantiations S for P can be defined as:

$$S = \{I_C = \{(x_1, v^1), \ldots, (x_n, v^n)\} | v^i \in D_i\}. \tag{3.2}$$

Also, a partial instantiation I_P can be defined as:

$$I_P = \{(x_1, v^1), \ldots, (x_k, v^k)\}. \tag{3.3}$$

that considers the assignment of a subset of size $k < n$ variables in X. To solve P, a complete instantiation $I_C^* \in S$ that satisfies all constraints in Ω has to be found. If P is a maximization problem and $F(I_C^*) \geq F(I_C)$, $\forall\ I_C \in S$, the complete instantiation I_C^* is an optimal solution for P.[1]

In general, a combinatorial problem P can be represented as a *construction graph* G_C where:

- $C = \{c_1, c_2, \ldots, c_{N_c}\}$ is the set of components, where N_c is the number of total components. These components can be defined as the variables in P, the set of possible assignations (variable, value), the union of variables and values of P, among others.
- States of the problem can be defined in terms of partial instantiations of finite length, in this case, over the elements of C: $I_P = \{c_1, c_2, \ldots, c_k\}$. Here, $k < n$ and the size of a partial instantiation I_P is $|I_P|$.

[1] If P is a minimization problem, $F(I_C^*) \geq F(I_C)\ \forall\ I_C \in S$, should be satisfied

- A set of all possible states is denoted by $\mathcal{X}$ and the set of feasible states $\tilde{\mathcal{X}}$, where $\tilde{\mathcal{X}} \subseteq \mathcal{X}$, is defined by the partial and complete instantiations that satisfy all constraints in Ω.
- A cost $Q(I_C)$ is associated to each complete instantiation I_C. In general, $Q(I_C) \equiv F(I_C)$ and this function can also measure the cost of a partial instantiation I_P.

Finally, $G_C = (C, L)$ is a completely connected graph whose vertices are the solution components C, and L are the set of edges that connect vertices. Artificial ants will perform randomized walks in this *construction graph* to construct complete instantiations for P.

3.2.2 Artificial Ants

Let us consider A an ant-based algorithm that considers m artificial ants and suppose that A has been designed for solving a combinatorial problem P. These artificial ants will perform walks in G_C cooperating and communicating among themselves. As real ants do, artificial ants will deposit (artificial) pheromone on paths that are being visited. The role of pheromone is to communicate to the other ants in the colony about vertices (or edges) that can be promising to solve P.

In G_C, components $c_i \in C$ and connections $l_{ij} \in L$ can have associated an amount of pheromone. This pheromone can be deposited on components (τ_i) or on the edge between components (τ_{ij}). These pheromone trails information is collected in a long-term memory (τ). For example, if pheromone is deposited on each component a vector can be considered and each cell represents how desirable a component can be. On the other hand, if pheromone is deposited on edges a pheromone matrix can be considered. Here, the cell τ_{ij} represents how desirable it can be to visit vertex j from vertex i. Usually, the initial value of each cell in the pheromone matrix τ will be defined at the beginning of the execution of A. As real pheromone, artificial pheromone intensity decays over time because of evaporation. How pheromone decreases is controlled by an *evaporation rule*.

On the other hand, heuristic information (η) is used to represent a priori specific information about the problem P. In general, η is a defined function that measures or estimates the cost of including a candidate component. Each artificial ant k of algorithm A has the following characteristics:

- It will perform stochastic walks in G_C for searching for an optimal solution I_C^*
- It has a memory M^k where the partial (and then, complete) instantiation will be stored. In some cases, M^k can be used during the construction process to calculate the pheromone and

Algorithm 1 Ant-based algorithm structure

```
1: InitializePheromoneTrails()
2: while e^k criterion not reached do
3:    for k = 1 → m do
4:       M^k ← ∅
5:       M^k ← ConstructCompleteInstantiation()
6:       LocalSearch(M^k) %%Optional;
7:    end for
8:    UpdatePheromone(I_C^LBest);
9: end while
10: return I_C^GBest
```

heuristic knowledge of each candidate component.

- A starting state x_{st}^k and termination condition(s) e^k should be defined.

Constructing complete instantiations: Algorithm 1 shows the structure of an ant-based algorithm. At every iteration of A, each ant k incrementally constructs a complete instantiation I_C^k (line 5), making stochastic *intermediate* decisions to include components into a partial instantiation I_P^k. At the beginning, the memory M^k of the ant k is empty (line 4), and components will incrementally be added to M^k. Considering a current state where last component c_i has been added, to decide which c_j component will be added next, a probabilistic decision rule named *state transition rule* is used. Here, pheromone information and heuristic knowledge of all candidate components are considered. This rule is particularly defined in each ant-based algorithm. In some ant-based algorithms, a local search procedure can be applied when an complete instantiation is constructed (line 6).

Pheromone management: Artificial pheromone is updated by a pheromone updating rule. In some cases, pheromone is updated applying a *global pheromone updating rule* where a fraction of the pheromone evaporates on all edges. Moreover, a heuristic H determines which ants will deposit an amount of pheromone on edges which belong to their tour. During evaporation, pheromone is decremented considering a scheduling rule. In general, this scheduling rule is defined by:

$$\tau_{ij}^{new} = (1 - \rho) * \tau_{ij}^{old} \tag{3.4}$$

where ρ is a parameter that controls the trail persistence. Usually, the *global pheromone updating rule* is applied when all ants have constructed a complete instantiation (line 8). There is also a *local pheromone updating rule* where pheromone is deposited during the construction process

of candidate solutions. Which pheromone updating rule and the heuristic H should be defined in each ant-based algorithm.

3.3 Ant-based algorithms

This section presents a review of the most important ant-based algorithms. These algorithms were the first approaches that considered artificial ants for solving combinatorial problems. Mostly, they were proposed for solving the well-known Traveling Salesman Problem and the construction process is quite similar between them. However, main components related to pheromone management and heuristic knowledge were specifically proposed. Also, the *anti-pheromone* concept and some applications are presented at the end of this section.

3.3.1 Ant System

Ant System (AS) [CDM92, Dor92, DMC96] was first introduced for solving the well-known Traveling Salesman Problem (TSP).[2] Constraints in TSP ensure that cities should be visited only once. In Ant System, artificial ants construct feasible complete instantiations checking that only *allowed* vertices can be visited. For this, a *"tabu"* list is used to avoid loops during the construction process. Pheromone values are initialized by a small positive constant c.

A TSP instance can be represented as a construction graph where its vertices of are the cities and a distance is associated to its edges. The distance considered in each problem instance is previously defined. For example, the length of the path between cities i and j considering the Euclidean distance in a 2 dimensional space is $d_{ij} = [(x_i - x_j)^2 + (y_i - y_j)^2]$. In Ant System for TSP, the heuristic knowledge considers the distance between these two cities (Eq. (3.5)). This definition will produce that cities that are closer will obtain a higher value, and vice versa.

$$\eta_{ij} = \frac{1}{d_{ij}}. \tag{3.5}$$

At each iteration, each ant k incrementally constructs a complete instantiation. If ant k is currently in vertex i, a transition rule is applied to decide which vertex will be visited next. In Ant System, the state transition rule is named *random-proportional rule* and is defined as:

[2] Let $C = \{a, \ldots, z\}$ be a set of cities, $L = \{(r, s) : r, s \in C\}$ be the edge set. d_{rs} is the distance between cities r and s. The TSP is the problem of finding a minimal length closed tour that visits each city once.

$$p_{ij}^k = \frac{[\tau_{ij}]^\alpha [\eta_{ij}]^\beta}{\sum_{u \in J^k(i)} [\tau_{iu}]^\alpha [\eta_{iu}]^\beta} \tag{3.6}$$

where $J^k(i)$ is the list of feasible vertices that can be visited, j is a unvisited allowed vertex, τ_{ij} is the current pheromone in edge between vertices i and j, η_{ij} is the heuristic knowledge related to this pair of vertices, α and β control the relative importance of pheromone information versus heuristic knowledge. Here, the construction process will favor attractive edges evaluated by the heuristic knowledge and also, will favor actions that were found in the past and which proved to be effective. Balance between exploration and exploitation can be managed with the values assigned to α and β. If $\alpha = 0$, the algorithm will perform as a stochastic greedy algorithm with multiple starting points. On the other hand, if $\beta = 0$ the algorithm will be guided only by the pheromone information without considering problem-dependent information.

Pheromone updating applies a *global pheromone updating rule* after all ants have constructed a complete instantiation considering the following formula:

$$\tau_{ij}^{new} = (1 - \rho) * \tau_{ij}^{old} + \Delta\tau_{ij} \tag{3.7}$$

where τ_{ij}^{new} is the updated pheromone value for edge (i,j), τ_{ij}^{old} is the current pheromone value for edge (i,j), ρ is the decreasing rate used for pheromone evaporation (as it was already explained in Section 3.2) and $\Delta\tau_{ij}$ is the quantity of pheromone that will be deposited by all the ants in this iteration. This amount is defined by:

$$\Delta\tau_{ij} = \sum_{k=1}^{m} \Delta\tau_{ij}^k \tag{3.8}$$

where $\Delta\tau_{ij}^k$ is the quantity deposited by ant k considering:

$$\Delta\tau_{ij}^k = \begin{cases} \frac{Q}{L_k} & \text{if } k\text{-th ant uses edge (i,j) in its tour in this iteration} \\ 0 & \text{otherwise} \end{cases} \tag{3.9}$$

where L_k is the total distance of the tour constructed by ant k, Q is a parameter. For Ant System, the heuristic H defines that all ants will deposit pheromone at the end of each cycle.

In [DMC96], three variants of Ant System were studied: *Ant-cycle*, *Ant-density* and *Ant-quantity*. These algorithms differ in the way how pheromone is deposited. In *Ant-cycle* pheromone

is deposited as in Equation 3.9. In the other two variants pheromone is deposited during the construction process.

Ant-density: An amount of Q will be deposited on edge (i,j) every time an ant goes from i to j:

$$\Delta\tau_{ij} = \begin{cases} Q & \text{if } k\text{-th ant goes from i to j} \\ 0 & \text{otherwise} \end{cases} \tag{3.10}$$

Notice that Q is a parameter of the algorithm. This variant increases pheromone without considering the distance associated to edges.

Ant-quantity: An amount of Q/d_{ij} will be deposited on edge (i,j) every time an ant goes from i to j:

$$\Delta\tau_{ij} = \begin{cases} \frac{Q}{d_{ij}} & \text{if } k\text{-th ant goes from i to j} \\ 0 & \text{otherwise} \end{cases} \tag{3.11}$$

This variant will consider to deposit higher amounts of pheromone on short distance edges producing that they will be more desirable during the construction process.

Experiments were performed with TSP instances in [DMC96] and *Ant-cycle* obtained better results than the other two variants of Ant System. Finally, an elitist strategy was also proposed to increase the importance of ants that found best quality tours in each iteration. For this, a fixed number of elitist ants will deposit an extra amount of pheromone. Equation 3.7 is modified as:

$$\tau_{ij}^{new} = (1-\rho) * \tau_{ij}^{old} + \Delta\tau_{ij} + n_e * \Delta\tau_{ij}^{e} \tag{3.12}$$

where n_e is the number of elitist ants and $\Delta\tau_{ij}^{e}$ is defined as:

$$\Delta\tau_{ij}^{e} = \begin{cases} \frac{Q}{L*} & \text{if edge}(i,j) \in L^* \\ 0 & \text{otherwise} \end{cases} \tag{3.13}$$

where L^* is the best tour found during the current iteration. The objective is to increase the

attraction of edges that were related to best quality solutions of each iteration.

3.3.2 Ant Colony System

Ant Colony System (ACS) [DG97] is presented as an improved version of Ant System for solving TSP. The main differences between ACS and AS are:

1. the *Pseudo-random rule*, the *state transition rule* used in ACS
2. the *global pheromone updating rule* is particularly defined to reinforce higher quality paths
3. a *local pheromone updating rule*, applied during the construction process

The *Pseudo-random rule* is defined to produce balance between exploration of new edges and exploitation of accumulated knowledge. The idea is to give priority to edges that are related to lower cost (shorter distances for TSP) and higher amounts of pheromone. An ant positioned in a component i will visit the next candidate vertex j defined by:

$$j = \begin{cases} argmax_{u \in J^k(i)}\{[\tau_{iu}] * [\eta_{iu}]^\beta\} & \text{if } q \leq q_0 \\ S, & \text{otherwise} \end{cases} \tag{3.14}$$

where $J^k(i)$ are the allowed vertex to be visited, $q \sim U(0,1)$ and q_0 is a parameter of ACS. Here, S is a vertex randomly selected according to probability:

$$p_{ij}^k = \frac{[\tau_{ij}][\eta_{ij}]^\beta}{\sum_{u \in J^k(i)}[\tau_{iu}][\eta_{iu}]^\beta} \tag{3.15}$$

similar to Equation 3.6 but in this case, without the α parameter.

The objective of q_0 is to balance exploration and exploration: if $q \leq q_0$, ant k will exploit the most promising edges. Otherwise, if $q > q_0$, ACS will explore other vertices giving an importance to heuristic knowledge defined by β parameter.

In ACS, the *global pheromone updating rule* is defined for reinforcing paths related to best quality solutions. For this, the heuristic H determines that the best quality solution, of the current iteration or of the execution, will be marked with pheromone. The rule is defined as:

$$\tau_{ij}^{new} = (1 - \rho_1) * \tau_{ij}^{old} + \rho_1 * \Delta\tau_{ij} \tag{3.16}$$

where $\rho_1 \in [0,1]$ is a parameter that controls the evaporation and increment of pheromone globally and $\Delta\tau_{ij}$ is defined as:

$$\Delta\tau_{ij} = \begin{cases} \frac{1}{L*} & \text{if (i,j) is in } I_C^B \\ 0 & \text{otherwise} \end{cases} \tag{3.17}$$

and I_C^B is the best quality solution. Moreover, I_C^B can be defined as:

- *iteration-best*, as the best solution found of the current iteration or,
- *global-best*, as the best solution found during the execution

A *local pheromone updating rule* is also defined in ACS, in addition to the global rule. The idea of this rule is to make the desirability of edges change dynamically during the construction process. The rule is defined as:

$$\tau_{ij}^{new} = (1-\rho_2) * \tau_{ij}^{old} + \rho_2 * \tau_0 \tag{3.18}$$

where $\rho_2 \in [0,1]$ is a parameter that controls the evaporation and increment of pheromone locally and τ_0 is the amount that will deposited. Authors experimentally defined that $\tau_0 = (n_G * L)^{-1}$ provided good results [DG97], considering that n_G is the number of vertices in construction graph G and L is the length of a tour generated using *nearest-neighbor* heuristic for TSP [RSL77].

Ant-Q [GD95] is a variant of Ant Colony System inspired by Q-learning [Wat89]. Ant-Q is defined as a family of algorithms with components similar to AS and ACS with some minor modifications. However, two components are important to be mentioned:

- Ant-Q considers a global and local pheromone updating rule defined as:

$$\tau_{ij}^{new} = (1-\rho)\tau_{ij}^{old} + \rho * (\Delta\tau_{ij} + \gamma * \max_{u \in J^k(i)} \{\tau_{iu}^{old}\}) \tag{3.19}$$

  where γ is a learning step size parameter. This rule is applied as a *local pheromone updating rule* considering $\Delta\tau_{ij} = 0$ until all ants have completed their construction process. On the other hand, this equation is reduced to the *global pheromone updating rule* of ACS when $\gamma = 0$. The objective is to influence the future decisions of the colony with the best edge (j,u) from the edge (i,j).

- The $\lambda-$branching factor is proposed to determine the stagnation of the algorithm. Authors defined it as an indicator of the dimension of the search space and they experimentally found that it should decrease monotonically during the execution of Ant-Q. A search process will be considered as stagnating if the average branching factor is lower than a small positive value (ϵ):

$$\frac{\sum_{i \in V} \lambda_i}{N_G} < \epsilon \tag{3.20}$$

  where λ_i is defined as the number of edges exiting from vertex i which have an associated pheromone value greater than $\lambda * \gamma_r + \tau_{ij}$, $\lambda \in [0, 1]$ is a parameter of the algorithm and $\gamma_r = \tau_{i,max} - \tau_{i,min}$. When the search process is stagnating the pheromone matrix is reinitialized to the maximum allowed values. The objective is to increase the exploration of the Ant-Q algorithm.

3.3.3 $\mathcal{MAX} - \mathcal{MIN}$ Ant System

$\mathcal{MAX} - \mathcal{MIN}$ Ant System ($\mathcal{MMAS}$) [SH96, SH00] was proposed to improve the performance of Ant System (AS) for solving TSP. Moreover, the main motivation was to improve AS in terms of the quality of solutions found and resource consuming. About the quality of the solutions, the larger the number of cities of a TSP instance, the lower was the quality of the solutions found by Ant System. On the other hand, authors remarked that one disadvantage of AS is its high run time. The main differences between $\mathcal{MMAS}$ and AS are:

1. The *global pheromone updating rule* is particularly defined to reinforce the highest quality path,
2. A shorter candidate list is considered in the *transition rule*,
3. Lower and upper bounds of pheromone trails are defined,
4. Pheromone trails are initialized considering the defined upper bound of pheromone.

First, the heuristic H allows only the best quality complete instantiations to be marked with pheromone. For this, the *global pheromone updating rule* of $\mathcal{MMAS}$ is defined as:

$$\tau_{ij}^{new} = (1 - \rho) * \tau_{ij}^{old} + \Delta\tau_{ij}^{best} \tag{3.21}$$

where $\rho \in [0, 1]$ is a parameter that controls the evaporation of pheromone and $\Delta\tau_{ij}^{best}$ is defined as:

$$\Delta\tau_{ij}^{best} = \begin{cases} \frac{1}{L*} & \text{if (i,j) is in } I_C^B \\ 0 & \text{otherwise} \end{cases} \tag{3.22}$$

and I_C^B is the best quality solution. As in ACS, I_C^B can be defined as:

- *iteration-best*, as the best solution found of the current iteration
- *global-best*, as the best solution found during the execution

Moreover, authors propose to use a schedule where iteration-best and global-best can be considered at different moments of the execution of $\mathcal{MMAS}$.

To improve the execution time of Ant System, authors propose to consider less candidates in each decision of the construction process. A *candidate set* is defined with a fixed number of nearest neighbors. The next component to be included in a partial instantiation will be taken from the candidate set. The size of the candidate set is a parameter of the algorithm.

The main components of $\mathcal{MMAS}$ are related with the pheromone management. The idea is to prevent the stagnation of the search process, preventing pheromone to be concentrated only in a subset of components or edges. Here, lower (τ_{min}) and upper (τ_{max}) limits for pheromone values are defined. The upper bound is defined as:

$$\tau_{max} = \frac{1}{1-\rho} * \frac{1}{L^*} \tag{3.23}$$

where L^* is the best found solution during the execution. On the other hand, the lower bound τ_{min} is defined considering the average number of components an ant has to choose in every decision and the probability of an ant to construct the optimal solution for a problem instance. Further details can be found in [SH00].

About the initialization of pheromone in $\mathcal{MMAS}$, pheromone trails are initialized with the maximum allowed amount of pheromone (τ_{max}).

Other component that is called Pheromone Trail Smoothing (PTS) has been studied with the objective of controlling the increment of pheromone:

$$\tau_{ij}^{new} = \tau_{ij}^{old} + \delta * (\tau_{max} - \tau_{ij}^{old}) \tag{3.24}$$

where $\delta \in (0, 1)$ and τ_{ij}^{new} is the pheromone trail after the smoothing. The objective of PTS is to increase the exploration of the search process by increasing the probability of selecting solution components with low pheromone trail.

Finally, the inclusion of a local search procedure was studied. More specifically, for solving TSP instances, a *2-opt* best-improvement local search procedure was considered after some ants have finished their construction process. This procedure was based on *2-opt*, where two edges of the current solution are removed and replaced by two other edges. The best ant of each iteration or a fixed number of ants perform this local search procedure.

3.3.4 Ranked Ant System

AS_{rank}, a variant of Ant System with elitism and a ranking method, was proposed in [BHS97]. As in Ant System, this variant was experimentally evaluated with TSP instances. The heuristic H allows n_e elitist ants to deposit pheromone and also, a ranked-based procedure is defined. Here, the m ants are ranked considering the constructed tour length. Moreover, the contribution of an ant to the trail level depends of its assigned rank. The main differences with Ant System are:

1. As the elitism described for AS, the best ant of each iteration deposits an amount of pheromone,

2. The pheromone updating rule considers a ranking of ants based in the quality of the obtained complete instantiation.

Here, the *global pheromone updating rule* is defined as:

$$\tau_{ij}^{new} = (1 - \rho) * \tau_{ij}^{old} + n_e * \Delta\tau_{ij}^{e} + \Delta\tau_{ij}^{r} \tag{3.25}$$

where n_e is the fixed number of elitist ants and $\Delta\tau_{ij}^{e}$ is defined as in Equation 3.13. If the m ants are sorted such that $f(I_C^1) \leq f(I_C^2) \leq \cdots \leq f(I_C^m)$, then:

$$\Delta\tau_{ij}^{r} = \sum_{\sigma=1}^{n_e} \Delta\tau_{ij}^{\sigma} \tag{3.26}$$

where $\Delta\tau_{ij}^{\sigma}$ considers only the best quality complete instantiations of n_e ants and it is defined as:

$$\Delta\tau_{ij}^{\sigma} = \begin{cases} \frac{(n_e-\sigma)*Q}{f(x^{\sigma})} & \text{if the ant } \sigma \text{ goes from i to j} \\ 0 & \text{otherwise} \end{cases} \tag{3.27}$$

where σ indicates the rank of the corresponding ant, $f(x^{\sigma})$ is the quality of the complete instantiation constructed by ant σ. Here, the better the ranking, the more the contribution. This updating rule increases the importance of edges related to best quality complete instantiations.

3.3.5 Anti-pheromone

The concept of *anti-pheromone* [MR02] was proposed as a different type of pheromone that will be deposited on the components of poor quality solution. The idea is the same as pheromone: components with higher values of anti-pheromone will be less attractive and repelled in the intermediate decisions during the construction process. Considering the Ant Colony System for TSP as the baseline algorithm, three variants were proposed: Subtractive Anti-pheromone (SAP), Preferential Anti-pheromone (PAP) and Explorer Ants.

Subtractive Anti-pheromone (SAP) This version is specifically focused in detecting edges that are related to poor quality solutions. As it was suggested in [SBHR97], pheromone will be removed from those elements that make up the worst solution in each iteration. For this, the global pheromone update rule of ACS includes a decreasing rate γ:

$$\tau_{ij}^{new-w} = \gamma * \tau_{ij}^{new-w} \tag{3.28}$$

where τ_{ij}^{new-w} is the updated pheromone value of an edge that is part of the worst w quality solution of an iteration and $\gamma \in [0, 1]$ is the decreasing rate. It is expected that edges of the worst quality solution will not be selected in the following construction processes.

Preferential Anti-pheromone (PAP) In this version, two types of pheromone are used simultaneously: one for good solutions and one for poor quality solutions. Here, a parameter θ controls the preference of pheromone versus anti-pheromone. The transition rule of ACS is redefined as:

$$j = \begin{cases} argmax_{u \in J^k(i)}\{[\theta * \tau_{iu} + (1-\theta) * anti\tau_{iu}] * [\eta_{iu}]^\beta\} & \text{if } q \leq q_0 \\ S, & \text{otherwise} \end{cases} \tag{3.29}$$

where $\theta \in [0,1]$, $anti\tau_{iu}$ defines the amount of *anti-pheromone* in the edge (i,u). Then, S is a vertex randomly selected according to probability:

$$p_{ij}^k = \frac{[\theta * \tau_{iu} + (1-\theta) * anti\tau_{iu}][\eta_{ij}]^\beta}{\sum_{u \in J^k(i)}[\theta * \tau_{iu} + (1-\theta) * anti\tau_{iu}][\eta_{iu}]^\beta} \tag{3.30}$$

The update of *anti-pheromone* is a *global pheromone update rule* where the worst tour from the current iteration deposits *anti-pheromone*.

Explorer Ants This variant defines a set of ants named *Explorer Ants* for which the preference for existing pheromone is reversed. These explorer ants will construct complete instantiations considering pheromone as undesirable. For this, an explorer ant k in the city i will go to city j defined by:

$$j = \begin{cases} argmax_{u \in J^k(i)}\{[\tau_{max} - \tau_{iu}] * [\eta_{iu}]^\beta\} & \text{if } q \leq q_0 \\ S^e, & \text{otherwise} \end{cases} \tag{3.31}$$

where τ_{max} is the highest current level of pheromone in the system. Then, S^e is a vertex randomly selected according to probability:

$$p_{ij}^k = \frac{[\tau_{max} - \tau_{ij}][\eta_{ij}]^\beta}{\sum_{u \in J^k(i)}[\tau_{max} - \tau_{iu}][\eta_{iu}]^\beta} \tag{3.32}$$

The number of explorer ants is a parameter of the algorithm.

3.4 Applications in Combinatorial Problems

All the presented ant-based algorithms were proposed for solving the well-known Traveling Salesman Problem. In this section, applications of ant-based algorithms to other combinatorial problems are detailed. The idea is to describe how these algorithms can be severally adapted for solving different problems. Also, the two baseline ant-based algorithms, Ant Solver and Ant Knapsack, are here described.

3.4.1 Quadratic Assignment Problem

The Quadratic Assignment Problem (QAP) is a *NP-hard* combinatorial optimization problem, that can be described as the problem of assigning a set of facilities to a set of locations with given distances and given flows between the facilities. The objective is to minimize the cost related to the distance between the locations and the flow between facilities.

$\mathcal{MMAS}-QAP$ [Stü97] is a $\mathcal{MMAS}$ algorithm proposed for solving QAP. It searches for feasible complete instantiations which consists in assigning every facility to exactly one location, and vice versa. Here, the problem is represented in a construction graph where vertices represent the facilities and the locations of the problem description. Pheromone represents the desirability of assigning a facility i to an available location j. $\mathcal{MMAS}-QAP$ does not make use of any heuristic information. On the other hand, it considers a local search procedure to improve the constructed candidate solutions.

At each construction step of $\mathcal{MMAS}-QAP$, ant k first randomly selects the next facility i to be instantiated. Then, the decision to which location j will be assigned is taken by the following *state transition rule*:

$$p_{ij} = \frac{\tau_{ij}}{\sum_{l \in J^k(i)} \tau_{il}} \tag{3.33}$$

where τ_{ij} is the pheromone trail between the facility i and location j, l is a unassigned location from the set of feasible locations $J^k(i)$.

When each ant has constructed a complete instantiation ϕ, solutions are improved using a local search procedure. Two local search procedures have been evaluated: iterative improvement local search (*2-opt*) and Robust Tabu Search (*Ro-TS*). Then, the pheromone trails are updated according to the following global pheromone updating rule as in Equation (3.21).

Finally, $\mathcal{MMAS}-QAP$ has an additional diversification feature that resets all the pheromone trails τ_{max}. Then, during the next q iterations, the best solution found of each iteration will be used in the pheromone update procedure.

3.4.2 Constraint Satisfaction Problems

Ant Solver [Sol02] is a well-known $\mathcal{MMAS}$ algorithm proposed to solve Constraint Satisfaction Problems (CSP). Ant Solver searches for a solution that minimizes the number of unsatisfied constraints. At each step, each ant k constructs a complete instantiation for the CSP. Unlike classical ant algorithms, when applied to CSP, the construction of a solution considers two decisions at each assignment: the selection of the next variable, according to some given criterion, and the selection of the value, made probabilistically according to the *state transition rule*. The pheromone is deposited on a binary graph whose vertices $\langle X_i, v\rangle$ represent the assignment of value v to variable X_i and the edges between two vertices $(\langle X_i, v\rangle, \langle X_j, w\rangle)$ represent those simultaneous assignments of values.

Equation 3.34 shows the transition rule used by Ant Solver. Here, it does not only depend on local relations between the candidate vertex and the last visited vertex, but also on a global relation between the candidate vertex and the whole set of visited vertices I_p. Hence, the pheromone factor of vertex $\langle X_j, v\rangle$ depends on pheromone deposited on all edges between $\langle X_j, v\rangle$ and the vertices in I_p.

$$p_{I_p}(\langle X_j, v\rangle) = \frac{[\tau_{I_p}(\langle X_j, v\rangle)]^{\alpha} * [\eta_{I_p}(\langle X_j, v\rangle)]^{\beta}}{\sum_{w \in D(X_j)} [\tau_{I_p}(\langle X_j, w\rangle)]^{\alpha} * [\eta_{I_p}(\langle X_j, w\rangle)]^{\beta}} \tag{3.34}$$

The heuristic knowledge in Ant Solver is defined to measure the number of possible new violated constraints for each candidate assignment. This factor also depends on the whole set of currently visited vertices in I_p:

$$\eta_{I_p}(\langle X_j, v\rangle) = \frac{1}{1 + F(\langle X_j, v\rangle \cup I_p) - F(I_p)} \tag{3.35}$$

where F is the evaluation function of the algorithm that counts the number of conflicts of a partial or complete instantiation. Ants that find the best quality solutions in each iteration are allowed to deposit pheromone. Pheromone is deposited on each pair of assignments in a complete instantiation. Equation (3.36) shows the amount of pheromone ($\Delta\ \tau$) deposited, where I_C^{LBest} is

the best quality complete instantiation of an iteration.

$$\Delta\tau(I_C^{LBest}, i, j) = \frac{1}{F(\mathbf{I_C^{LBest}})} \tag{3.36}$$

Ant Solver includes *pre-* and *post-* processing steps that use a *min-conflicts* based local search procedure. The pre-processing phase repeatedly performs a local search to collect information that will be used to initialize pheromone trails. The post-processing phase performs a local search after each ant has constructed a complete instantiation.

3.4.3 Multidimensional Knapsack Problem

Multidimensional Knapsack Problem (MKP) is an *NP-hard* combinatorial optimization problem, variant of the well-known Knapsack Problem. MKP is defined as a knapsack with T resource constraints and a set of objects $O = \{o_1, \ldots, o_N\}$. Each object i has a defined profit p_i and a weight w_{it} in each problem dimension t. The idea is to select a subset of $n \leq N$ objects in such a way that each capacity constraint, in dimension t, is satisfied maximizing the total profit. The capacity constraints define the maximum capacity b_t for each dimension.

Ant Knapsack (AK) [ASG04] is a well-known $\mathcal{MMAS}$ algorithm proposed to solve MKP. AK only constructs feasible complete instantiations and it uses a binary representation where each bit represents the decision of including an object into the knapsack. For this, each ant k includes objects iteratively in a partial instantiation I_P^k, until the dimensions are filled and no object can be included. At each step, a list of candidates J^k is constructed with the allowed candidate objects. The heuristic knowledge is focused on the current capacity (CC), considering the objects were already included in the partial instantiation. About the pheromone management, it is deposited on each pair of objects (o_i, o_j) (with $i \neq j$) that are part of the best quality solution of each iteration. Then, the *state transition rule* is defined as:

$$p_{I_P^k}(o_i) = \frac{[\tau_{I_P^k}(o_i)]^\alpha * [\eta_{I_P^k}(o_i)]^\beta}{\sum_{z \in J^k(j)} [\tau_{I_P^k}(o_z)]^\alpha * [\eta_{I_P^k}(o_z)]^\beta} \tag{3.37}$$

where:

- $\tau_{I_P^k}(o_i)$ is defined as the sum of the pheromone trails between the object i and objects v (that are already in I_P^k):

$$\tau_{I_P^k}(o_i) = \sum_{o_v \in I_P^k} \tau(o_i, o_v), \tag{3.38}$$

- $\eta_{I_P^k}(o_i)$ is the heuristic knowledge defined as a rate between the profit p_i of the object i and the current capacity factor $h_{I_P^k}(i)$ considering objects in I_P^k:

$$\eta_{I_P^k}(o_i) = \frac{p_i}{h_{I_P^k}(i)} \tag{3.39}$$

where $h_{I_P^k}(i)$ is defined as:

$$h_{I_P^k}(i) = \sum_{t=1}^{T} \frac{w_{it}}{CC_t} \tag{3.40}$$

where w_{it} is the weight of object i in dimension t, CC_t is the current capacity in dimension t defined as:

$$CC_t = b_t - \sum_{o_v \in I_P^k} w_{vt} \tag{3.41}$$

where b_t is the capacity of the dimension t and $\sum_{o_v \in I_P^k} w_{vt}$ is the sum of the weights of the objects in the partial instantiation.

At the end of each iteration, the artificial ant that constructed the best quality complete instantiation of the iteration is allowed to mark every pair of objects present in its solution and deposits an amount $\Delta\tau$ of pheromone defined as:

$$\Delta\tau = \frac{1}{1 + F(L_{best\_found}) - F(L_{best\_it})} \tag{3.42}$$

where L_{best_found} is the best solution found during the execution and L_{best_it} the best solution found in the current iteration.

3.5 Summary

This chapter presented how the behavior of real ants has been represented by artificial ants in metaheuristic algorithms. Most relevant experiments were explained to understand how artificial ants were defined first in Ant System algorithm. A revision of the most important ant-based algorithms was presented in this chapter: Ant System, Ant Colony System, $\mathcal{MAX} - \mathcal{MIN}$ Ant System and Ranked Ant System. In general, these approaches try to communicate to a colony of artificial ants which edges should be attractive to be traversed in a construction graph. Each algorithm considers a particular balance between exploration and exploitation, mostly defined by how pheromone is updated and which ants are allowed to deposit pheromone. Two different types of rules are used to update pheromone: global and local. The first one is related to modify a trail

in a particular iteration and it is performed when all ants have finished their construction process. On the other hand, a local pheromone updating rule is used to modify the desirability of nodes (or edges) during the construction process. In general, these updating rules are specifically defined in each ant-based algorithm.

As Ant System was originally proposed for solving the Traveling Salesman Problem, all these variants were also proposed for the same problem. As a consequence, their main components are defined considering the importance of the existing distance between nodes in a construction graph. For example, the heuristic knowledge in all these approaches was defined to represent the distance existing between each pair of nodes. The idea is to give priority to edges that are associated with shorter distances.

We presented details of the concept of *anti-pheromone*. Three variants of Ant Colony System were proposed that incorporate the concept of learning about edges related with poor quality solutions. The idea of biasing the construction process of artificial ants considering "extra" information about the problem being tackled was an inspiration for this thesis.

Finally, three applications of ant-based algorithms were detailed for different combinatorial problems: $\mathcal{MMAS} - QAP$ for the Quadratic Assignment Problem, Ant Solver for solving Constraint Satisfaction Problems and Ant Knapsack for Multidimensional Knapsack Problem. The last two algorithms are the baseline approaches for our work. In the following chapter we present our approach that was inspired in Opposite Learning literature and the concept of *anti-pheromone*.

Chapter 4

An Opposite Inspired Learning strategy for Ant-based Algorithms

In an ant-based algorithm, each artificial ant constructs complete instantiations for a selected problem. The construction process of partial instantiations is based on *intermediate* decisions taken using two components: pheromone and heuristic knowledge. Pheromone is used as a communication method among the artificial ants in the colony, to guide the search process to interesting regions of the search space. On the other hand, heuristic knowledge measures the effect of each possible assignment using information on the problem being solved.

When the problems are complex to solve, some of the decisions made during the construction process can lead the algorithm to complete instantiations with less quality than expected. Here, we are focused on ant-based algorithms that have shown to be able to solve complex combinatorial problems, but in some stages of the construction of partial instantiations they make some decisions that could be improved. In other words, the construction process can be biased by the attraction of some characteristic of the candidate components. We name such characteristic as *undesirable* ($u\mathcal{D}$) and our objective is to provide useful information to an ant-based algorithm, to guide the search process in a better way, when decisions can lead to poor quality complete instantiations.

We propose an Opposite Inspired Learning strategy to identify an $u\mathcal{D}$-characteristic from complete instantiations. The idea is to reduce the attraction of this particular $u\mathcal{D}$-characteristic, giving a chance to other components that the original algorithm would not consider. The details of this strategy are presented in Section 4.1. To learn about an $u\mathcal{D}$-characteristic, we propose to divide the search process into two steps: a *First Step* to identify a characteristic from the complete instanti-

ations and a *Second Step* to solve the problem of interest. The main idea is to use the information obtained in the *First Step* during the *Second Step* to change the decisions of the algorithm during its construction process. Sections 4.2 and 4.3 present details of how this knowledge should be obtained during a previous learning step. Finally, there are some components of this strategy that are necessary to be defined. Details of these components are presented in Section 4.4.

4.1 $u\mathcal{D} - Characteristic$

In the last chapter, we presented an ant-based algorithm A designed to solve a combinatorial problem P using m artificial ants. As we already explained, I_C^* is the optimal solution of P and this combinatorial problem can be represented as a construction graph $G_C = (C, L)$ whose vertices are solution components C, and L is the set of edges that connect pairs of vertices. Artificial ants perform walks in G_C cooperating among them, depositing pheromone on paths they visit. Let us consider that pheromone in A will be deposited on edges with the objective of detecting promising pairs of vertices for solving P. Moreover, this information is collected in a pheromone matrix (τ), where the cell τ_{ij} represents how desirable it can be to visit vertex j from vertex i. At every iteration of A, each ant k incrementally constructs a complete instantiation I_C^k, making stochastic *intermediate* decisions to include components into a partial instantiation I_P^k. To decide which path the ant k should visit next, pheromone information and heuristic knowledge of all candidate vertices are considered in a *state transition rule*. This rule can be any of what we have already mentioned in Section 3.3: *random-proportional rule*, *pseudo-random rule*, among others.

We are interested in improving the search process of A, in terms of the quality of the complete instantiations that A can build. The idea is to maintain the original design and components of A. For this, we are interested in provide useful information to A in order to improve its *intermediate* decisions. Let us suppose that ant k constructs a complete instantiation $I_C^k = \{(x_1, v^1), \ldots, (x_n, v^n)\}$ that $F(I_C^k) < F(I_C^*)$, considering P as a maximization problem. Although the structure of I_C^* can be unknown, we can expect that I_C^k and I_C^* are structurally different in terms of their assignments.

In most cases, components are included in I_C^k because a certain preference related to their heuristic knowledge and pheromone information was considered in the stochastic *intermediate* decisions of A. As heuristic knowledge function is particularly defined in A and the information in pheromone matrix is limited by the vertices which were already visited during the current execution, in some cases, the information provided to perform *intermediate* decisions might be poor. This information can lead the construction process of A to complete instantiations with less

quality than expected.

Let us suppose that I_C^k has some characteristic w that can be measurable and related to:

- a structural property of I_C^k,
- a quality feature of I_C^k,
- a feature related to the (in)feasibility in I_C^k,
- a P-specific property feature,

among others. This characteristic may be present on some components by themselves or in a relationship between components in I_C^k. Here, we can expect that w is not only present in I_C^k, but also the construction process of I_C^k could be influenced by w.

During the construction process of I_C^k, *intermediate* decisions are biased by some characteristic w that makes some components appear more promising than others. We name this characteristic as *undesirable* ($u\mathcal{D}$). We propose to learn about an $u\mathcal{D}-$characteristic w in I_C^k to decrease the attraction to components that A considered promising. It is important to remark that an $u\mathcal{D}-$characteristic can be not perceptible by the current pheromone information and by the heuristic knowledge, as it is specifically defined in A. The objective is to allow A to consider other *intermediate* decisions during its construction process and finally, obtain better quality solutions.

Let $S_{(A,i)}$ a set of complete instantiations obtained by A during its i^{th} iteration and w an $u\mathcal{D}$-characteristic. As w is measurable, we define a function $\mathcal{W}$ to detect the presence of an $u\mathcal{D}$-characteristic in $S_{(A,i)}$. Some examples can be:

- If w is related to the quality of solutions in $S_{(A,i)}$, $\mathcal{W}$ can be defined as F for partial and complete instantiations.
- Considering that P is an instance of the QAP, w can be related to a poor prioritization of assignments. Let us define (f_i, f_j) as a *priority pair* of facilities where, compared to other pairs of facilities in P, there is a high flow value between f_i and f_j. In this case, $\mathcal{W}$ can penalize solutions that have a (locally) maximum distance for *priority pairs* of facilities like (f_i, f_j). Notice that $\mathcal{W}$ can be also complemented considering a quality measure to compare solutions in $S_{(A,i)}$.
- Considering that P is a binary CSP instance, w can be related to complete instantiations that are difficult to be repaired. In this case, P defines pairs of incompatible values for some

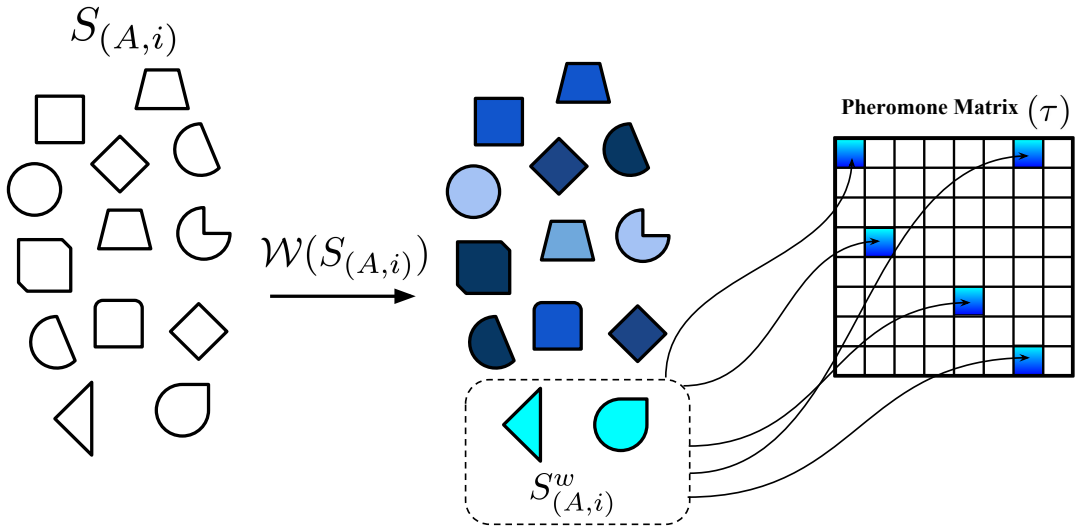


Figure 4.1: Identifying and learning about a characteristic in $\mathcal{I}^C$

pairs of variables. As a consequence, there are some conflicts that have higher number of incompatible values. The higher the number of incompatible values in a complete instantiation, the more difficult the conflicts are to be repaired.

$\mathcal{W}$ can penalize the solutions in $S_{(A,i)}$ that have the highest number of incompatible values. Moreover, $\mathcal{W}$ can also consider the number of conflicts to better discriminate among solutions in $S_{(A,i)}$.

- Considering that P is a MKP instance, w can be related to how the capacity is being consumed during the construction process. In general, dimensions have different capacities and objects have different weight in each capacity. Let us define a dimension as *critical*, if it has the lowest capacity value compared to the other dimensions. $\mathcal{W}$ can evaluate candidate solutions in $S_{(A,i)}$, considering the profit of the selected objects and penalizing those solutions that consumed a higher capacity in a *critical* dimension. The idea is to discriminate about objects that are promising for A, in terms of their profit, and are also part of poor quality solutions.

Figure 4.1 explains how $u\mathcal{D}$-characteristic will be identified. In the left part of the figure, the set of complete instantiations $S_{(A,i)}$ obtained by A in the i^{th} iteration is shown. In the center of the figure, solutions in $S_{(A,i)}$ are colored representing their evaluation value $\mathcal{W}(S_{(A,i)})$, where the lighter the color the more the presence of w. We define $S^w_{(A,i)}$ as the set of complete instantiations

that have more presence of w.[1]

As pheromone produces a modification of the way the problem is represented and perceived by artificial ants [DS04], we decided to use pheromone to learn about $u\mathcal{D}$-characteristics. Moreover, the components of the complete instantiations in $S^w_{(A,i)}$ will be marked with pheromone. Inspired in the concept of *anti-pheromone* we will name pheromone for marking the solution in $S^w_{(A,i)}$ as *anti-pheromone*. This *anti-pheromone* information will be used during the *intermediate* decisions taken by A, to include components to a partial instantiation I_P. The hypothesis is: if we consume an amount of resources in identifying and learning about some $u\mathcal{D}$-characteristic w in $S_{(A,i)}$, the search process could be further focused making decisions using this knowledge and obtain better quality complete instantiations. For this, we propose to divide the search process of A into two steps. Next section presents the details of the learning process.

4.2 Division of the Search Process

This section presents the details for learning about an $u\mathcal{D}$-characteristic during the execution of A. First, we propose to divide the search process of A into two steps: a *First Step* to learn about w in $S_{(A,i)}$ and a *Second Step* performed by A using the obtained knowledge in the *First Step*. As shown in Figure 4.1, pheromone will be used to mark a complete instantiation in $S^w_{(A,i)}$ during the *First Step*.

Inspired in the concept of *anti-pheromone* [SBHR97], the idea is to learn about the components of solutions in $S^w_{(A,i)}$ and their relationship. For now on, pheromone used during the *First Step* will be named as *anti-pheromone*. In this case, a pair of components (c_i, c_j) that were more related with a complete instantiation in $S^w_{(A,i)}$ during *First Step* will have a higher amount of anti-pheromone $anti\tau_{ij}$. The idea is to produce a repellent effect to some pairs of components, allowing A to consider other components that originally it usually would not be able to include. To determine on which candidate solutions the anti-pheromone will be deposited, the heuristic H will be defined considering the $\mathcal{W}$ function. Our goal is to identify an $u\mathcal{D}$-characteristic using our Opposite Inspired Learning strategy, represented by the anti-pheromone matrix ($anti\tau$). The decisions performed during the construction process of A in the Second Step will then be influenced by this *anti-pheromone* translated in the pheromone matrix of A.

Let A° an ant-based algorithm that will perform the *First Step* and *anti-pheromone* will be

[1] The size of $S^w_{(A,i)}$ will be particularly defined for each application of this strategy. For simplicity, from now on we will consider that $S^w_{(A,i)}$ contains only one candidate solution.

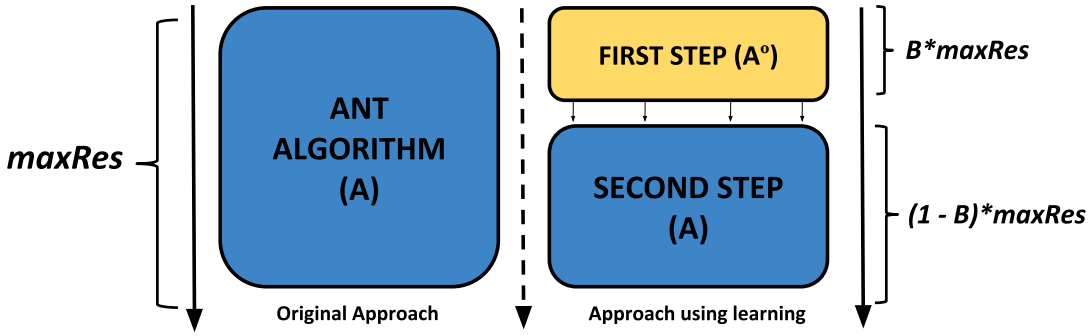


Figure 4.2: Resources distributed by parameter B

used to reinforce paths that are related with complete instantiations in $S^{w}_{(A,i)}$. A° has the following components equally defined as A:

- the defined representation for P,
- the evaluation function F,
- the structure of the *state transition rule*

However, the heuristic knowledge η, the anti-pheromone management and the parameters of the algorithm will be particularly defined in A°. More specifically, the global anti-pheromone updating rule and which artificial ants will be allowed to deposit anti-pheromone in each iteration. Also, a new parameter value should be considered in A° for each parameter in A (e.g, α_{FS}, β_{FS}, ρ_{FS}, among others, where FS means *First Step*).

Figure 4.2 shows how the search process is divided into two steps. In the left part of the figure, the execution of algorithm A is shown where an amount $maxRes$ of resources has been consumed for solving P. In the right part of the figure, the *First Step* is performed by A° consuming an amount $B * maxRes$, where B is a parameter that defines the percentage of resources designed to the learning step. At the end of the *First Step*, an anti-pheromone matrix will be obtained, that will be translated into pheromone and used by A as its initial pheromone matrix. Further details of how the translation process can be made are described in the following sections. Finally, A performs its search process considering the remaining $(1 - B) * maxRes$ resources.

In the following section, different methods to identify and learn about some $u\mathcal{D}$-characteristic will be presented. Also, the anti-pheromone management will be particularly defined for each method.

4.3 Methods

In order to explore and compare different possibilities to identify an $u\mathcal{D}$-characteristic, we propose three different *Methods*. Each method will consider a different definition for the heuristic knowledge and anti-pheromone management for A°. As a consequence, each method will perform a different search process during the *First Step* and defines how to identify an $u\mathcal{D}$-characteristic. The methods are named *Soft Opposite-Learning* (SOL), *Worst Opposite-Learning* (WOL) and *Half Opposite-Learning* (HOL). The first two methods are related to the quality of complete instantiations and the last one is related to problem-specific features.

***Soft Opposite-Learning* (SOL)** This method is focused on identifying an $u\mathcal{D}$-characteristic related to the quality of complete instantiations but trying to perform a similar search process as in A. For this, the heuristic knowledge η of A° will be the same as in A. Considering an heuristic knowledge defined for pairs of components, a partial instantiation I_P and an ant that has currently included component i in I_P, the heuristic knowledge of a candidate component j will be defined in A° as:

$$\eta_{ij}^{A^{\circ}} = \eta_{ij}^{A} \tag{4.1}$$

On the other hand, the heuristic H will be defined as follows: anti-pheromone will be deposited in edges that are related to the lowest quality solution of each iteration constructed by ant^L. The *global anti-pheromone updating rule* can be defined as:

$$anti\tau_{ij}^{new} = (1 - \rho_{FS}) * anti\tau_{ij}^{old} + \Delta\tau_{ij}^{L} \tag{4.2}$$

where ρ_{FS} is the evaporation rate for the *First Step* and $\Delta\tau_{ij}^{L}$ is the amount of anti-pheromone that will be deposited by ant^L. H and the amount to be deposited ($\Delta\tau_{ij}^{L}$) depends on how they are defined in A. For example, if A considers a fixed number of elitist ants, this can be considered also in Equation 4.2 for anti-pheromone deposit.

The idea is that SOL method will consider the same edges that are attractive for A in terms of the heuristic knowledge, but will focus on constructing poor quality complete instantiations. The objective is to identify complete instantiations where an *intermediate* decision (i.e, from vertex i to vertex j that was an attractive path to take for ant k), can lead the search process to less quality complete instantiations than expected. The information obtained during a *First Step* performed by the SOL method, stored in the *anti-pheromone* matrix, will reduce the level of attraction produced

A	**SOL**	**WOL**	**HOL**
η	η	$\overline{\eta}$	$\eta^{\mathbb{A}^\circ}$
H	H	H	H
F	$\overline{F}$	$\overline{F}$	$\overline{F}$

Table 4.1: Summary of Methods

by heuristic knowledge in the corresponding *intermediate* decisions of A.

Worst Opposite-Learning **(WOL)** This method is focused on evaluating the effect of taking totally opposed decisions to the objective of the problem P. The idea is to identify components that are considered simultaneously in bad quality complete instantiations. For this, the construction process is performed against the objective of P. In this method, the heuristic knowledge should be inverted in each *intermediate* decision:

$$\eta_{ij}^{A^\circ} = \max_{u \in J^k(i)} (\eta_{iu}^A) + \min_{u \in J^k(i)} (\eta_{iu}^A) - \eta_{ij}^A \tag{4.3}$$

where the maximum and minimum heuristic knowledge values are considered from a list $J^k(i)$ of u candidate components allowed to be included, and η_{ij}^A is the heuristic knowledge as it is defined in A. The idea is to make more attractive the less interesting components. Furthermore, H is equally defined as in A but in this case, $\overline{F}$ will be optimized: anti-pheromone is deposited on the lowest quality solution obtained at each iteration (as in Equation 4.2). We expect that using the WOL method, the search process of A° explores regions of the search space that the original algorithm could never visit.

Half Opposite Learning **(HOL)** This method is focused on detecting a problem-specific $u\mathcal{D}$-characteristic w. Let us suppose that, as the heuristic knowledge and pheromone management are defined in A, any information on w cannot be obtained. As a consequence, the heuristic knowledge η^{A° should be redefined in A° to detect this problem-specific feature. The construction process should be guided considering η^{A°, allowing the search process to consider information on the presence of w in the obtained complete instantiations. Moreover, it is necessary to particularly define $\mathcal{W}$ in order to measure the presence of w in $S_{(A^\circ,i)}$. Anti-pheromone should be deposited in the complete instantiation defined by H.

Table 4.3 shows a summary of the main features of each *Method*. As it was already presented, $\mathbb{A}$ algorithm considers a heuristic knowledge η, a heuristic H that defines which trails will be marked

with pheromone and F is the function to be optimized in the problem $\mathbb{P}$ being solved. In *SOL Method*, η and H are maintained as in the original algorithm. Here, the algorithm will deposit anti-pheromone in solution(s) that optimize $\overline{F}$. About *WOL Method*, the heuristic knowledge is translated ($\overline{\eta}$) and $\overline{F}$ will be optimized. As in *SOL*, the solution that optimizes $\overline{F}$ will be marked with anti-pheromone (H). In *HOL Method*, $\overline{F}$ will be optimized and H will be maintained as in $\mathbb{A}$. However, a different heuristic knowledge should be considered ($\eta^{\mathbb{A}^\circ}$).[2]

It is important to mention that, which method will obtain useful information during the *First Step*, will directly depend on the selected ant algorithm and on the problem being solved. It is important to define some components that will be detailed in the following section.

4.4 Components

To implement our strategy, considering any of these methods, it is necessary to define and study the following components:

- The amount of anti-pheromone $\Delta anti\tau$ that will be deposited during the *First Step*. This amount must be consistent with the quantity of pheromone deposited by the ants in A. The amount of anti-pheromone should not be constant and can consider different measures to guide the search process. Moreover, the amount of anti-pheromone should not exceed the maximum pheromone that can be deposited during the *Second Step*.

- Which ants will be allowed to deposit anti-pheromone during *First Step*. As we already mentioned, this component can be mapped from its definition in A.

- A *translation rule*, to initialize the pheromone values for the *Second Step* using the information obtained during the *First Step*. If the ant based algorithm is a $\mathcal{MAX}-\mathcal{MIN}$ Ant System ($\mathcal{MMAS}$), it is necessary to translate the anti-pheromone considering the maximum and minimum allowed boundaries for pheromone.

- The percentage B defines the effort that will be invested in the *First Step*. As shown in Figure 4.2, a maximum amount of resources $maxRes$ is distributed. It is important to remark that the effort invested in the whole process (both steps) should be the same as the one used by A: the higher the percentage for the *First Step*, the lower the effort during the

[2]For this, $\eta^{\mathbb{A}^\circ}$ can be defined considering an existing heuristic knowledge proposed in the literature, based in a heuristic already proposed on literature for $\mathbb{P}$ or propose a new one.

Second Step. On the other hand, a very low percentage B will not allow the anti-pheromone to converge.

4.5 Conclusions

In this chapter we propose an Opposite Inspired Learning strategy to improve the search process of ant-based algorithms designed for solving Combinatorial Problems. The strategy consists in detecting an $u\mathcal{D}$-characteristic, an undesirable characteristic that leads the construction process of an ant-based algorithm to poor quality solutions. To detect this characteristic, we propose to divide the search process of an ant-based algorithm into two steps: a *First Step* focused on detecting such $u\mathcal{D}$-characteristic and a *Second Step* to solve the problem of interest.

Our strategy tries to identify a $u\mathcal{D}$-characteristic during a *First Step* using a defined amount of resources. When an amount of $B*maxRes$ resources is consumed, the anti-pheromone is translated into pheromone to define the initial values for the original ant-based algorithm A. Finally, A will be executed using the remaining $(1-B)*maxRes$ resources for solving the problem being tackled.

We proposed three different methods to perform the *First Step*: *Soft Opposite-Learning* (SOL), *Worst Opposite-Learning* (WOL) and *Half Opposite-Learning* (HOL). Each method performs a particular search process, with specifically defined heuristic knowledge and pheromone management components.

To implement this strategy, it is important to define how much anti-pheromone will be deposited at each iteration, which ants will be allowed to deposit anti-pheromone, how anti-pheromone will be translated into pheromone and the amount of resources designed to the *First Step*.

In the following chapters, we present the application of this strategy to two well-known ant-based algorithms for solving different combinatorial problems. For each algorithm, these methods and components are particularly defined considering how the original algorithms were designed.

Chapter 5

OL for Multidimensional Knapsack Problem

This chapter presents the application and evaluation of our strategy in Ant Knapsack (AK). Ant Knapsack is a well-known ant-based algorithm proposed for solving the Multidimensional Knapsack Problem (MKP). As Ant Knapsack was already described in Section 3.4.3, this chapter only includes a summary of most important details and components of the algorithm in Section 5.2.

To implement our strategy, it is also necessary to study the problem considered to design the ant-based algorithm. In this case, we are interested in Multidimensional Knapsack Problem (MKP). This combinatorial problem was proposed by a mathematician named Tobias Dantzig near to 1900, and it has been severally studied. However, it is still considered by the community to design and evaluate algorithms. Section 5.1 presents formal description of the MKP and some related work proposed on literature.

Our strategy requires that an ant-based algorithm performs a First Step to obtain information about some undesirable ($u\mathcal{D}$) characteristic present in complete instantiations. We propose three methods to perform different search processes and evaluate which information could be more valuable for the selected algorithm. As we described earlier, in this chapter, the ant-based algorithm A will be AK and A°, that will be used during the First Step. A° requires the definition of some components and some details for each method. Section 5.3 details the inclusion of our strategy in Ant Knapsack.

This chapter also presents the experimental evaluation of the inclusion of our strategy in Ant

Knapsack. The objective is to evaluate how the information about some $u\mathcal{D}$-characteristic can improve the search process of Ant Knapsack. Section 5.4 shows the description of experimental setup and the set of benchmark instances used to evaluate our approaches. Moreover, results for these instances, statistical analysis, boxplots and fitness-distance plots are presented.

5.1 Multidimensional Knapsack Problem

Multidimensional Knapsack Problem (MKP) is a *NP-hard* combinatorial optimization problem, variant of the well-known Knapsack Problem. MKP is defined as a knapsack with multiple resource constraints and a set of objects $O = \{o_1, \ldots, o_N\}$. Each object i has a defined profit p_i and weight w_{it} in each problem dimension t. The problem consists in selecting a subset of $n \leq N$ objects in such a way that each capacity constraint, for the T dimensions, is satisfied maximizing the total profit. The capacity constraints define the maximum capacity b_t of each dimension. Formally, given:

$$x_i = \begin{cases} 1 & \text{if object } i \text{ is included in the knapsack} \\ 0 & \text{otherwise} \end{cases} \tag{5.1}$$

$$\text{Max } Z = \sum_{i=1}^{N} p_i * x_i \tag{5.2}$$

Subject to:

$$\sum_{i=1}^{N} w_{it} * x_i \leq b_t, \forall t = 1, \ldots, T \tag{5.3}$$

where p_i is the profit of the object i, w_{it} is the weight of the object i in dimension t and b_t is the capacity of dimension t. To measure the quality of objects, some efficiency measures have been proposed related the cost of obtaining one unit of profit.[1] Considering the T dimensions of a problem instance, an object o_i can be light in one dimension and be very heavy in other one. Moreover, some dimensions can have less capacity than others. For this reason, it is there no obvious efficiency measure for objects [PRP10].

[1] Definition of Efficiency: "The ratio of the useful work performed by a machine or in a process to the total energy expended or heat taken in".(Available in `https://en.oxforddictionaries.com/definition/efficiency`)

The most typical efficiency measure e_i can be defined as:

$$e_i = \frac{p_i}{\sum_{t=1}^{T} w_{it}}. \tag{5.4}$$

Moreover, as constraints consider different orders of magnitude in their capacities and weights, a scaled version can be defined as:

$$e_i^{scaled} = \frac{p_i}{\sum_{t=1}^{T} \frac{w_{it}}{b_t}}. \tag{5.5}$$

In some cases, the average of the weights in all dimensions of each object $\overline{w_i}$ can be considered:

$$e_i^{avg} = \frac{p_i}{\overline{w_i}} \tag{5.6}$$

An alternative efficiency measure that considers how restricted are some dimensions of the problem was proposed in [KPP04]. For this, relevance values r_t are computed for every constraint:

$$e_i = \frac{p_i}{\sum_{t=1}^{T} r_t * w_{it}} \tag{5.7}$$

where r_t are computed as follows:

$$r_t = \frac{(\sum_{t=1}^{T} w_{it}) - b_t}{\sum_{t=1}^{T} w_{it}} \tag{5.8}$$

Here, the lower the capacity of dimension t and the higher the sum of weights in t, the more restricted is the dimension.

There are applications of the Multidimensional Knapsack Problem in different areas. For example, for capital budgeting [MCS01], optimizing a portfolio of Research & Development projects [BMM01], among others. Also, there are several approaches proposed for solving MKP and in the following, some important works are presented.

In [Fid05], four different heuristic knowledge functions were proposed for ACO algorithms, each one with a different efficiency measure. These functions are:

1. Considering $s_i = \sum_{t=1}^{T} w_{it}$, the first heuristic knowledge function is defined as:

$$\eta_i = \frac{p_i^{d_1}}{s_i^{d_2}} \tag{5.9}$$

   where $d_1 > 0$ and $d_2 > 0$ are parameters of the algorithm. Here, objects with greater profit and lower weight will be more attractive. This heuristic is similar to Equation 5.4, but the parameters d_1 and d_2 try to produce a balance between profit and the sum of weights.

2. The second heuristic function is similar to Equations 5.9 and 5.5, but considering $s_i = \sum_{t=1}^{T} w_{it}/b_t$. Here, objects with greater profit that use less capacity will be more desirable.

3. In this case, η_i is calculated dynamically for each candidate object. Considering a partial instantiation I_P that is being constructed, the heuristic knowledge is defined as in Equation 5.9, but in this case $d_1 = d_2$ and s_i is defined as:

$$s_i = \sum_{t=1}^{T} \frac{w_{it}}{CC_t} \tag{5.10}$$

   where CC_t is the current capacity defined as:

$$CC_t = b_t - \sum_{o_v \in I_P^k} w_{vt} \tag{5.11}$$

   where b_t is the capacity of the dimension t and $\sum_{o_v \in I_P^k} w_{vt}$ is the sum of the weights of the objects that are already included in the partial instantiation.

4. The fourth heuristic is equal to the third one but considering $d_1 \neq d_2$ in Equation 5.9. The idea is to analyze the effect of giving different importance to the profit and the weights.

A Genetic Algorithm for solving MKP was proposed in [CB98]. This approach considers a binary representation, binary tournament parent selection, uniform crossover and a specific repair operator. For this, *pseudo-utility ratios* are calculated using the surrogate duality approach [Pir87] and the *surrogate relaxation* problem of the MKP (SR-MKP) considering Equation 5.2 and:

$$\sum_{i=1}^{N}(\sum_{t=1}^{T} \omega_t * w_{it}) * x_i \leq \sum_{t=1}^{T} \omega_t * b_t, \forall t = 1, \ldots, T \tag{5.12}$$

where $\omega = \{\omega_1, \ldots, \omega_T\}$ is the set of surrogate multipliers. SR-MKP is single constraint knapsack problem and the idea is to obtain the optimal set of multipliers that minimizes the

solution value. These multipliers can be obtained using a linear programming solver [CB98]. Then, the *pseudo-utility ratios* are calculated as:

$$u_i = \frac{p_i}{\sum_{t=1}^{T} \omega_t * w_{it}} \tag{5.13}$$

These ratios are used in the repair operator to decide which objects should be discarded from a complete instantiation and which objects are more promising to be included.

Other important contribution are 270 instances proposed for the OR Library by Chu and Beasley[2]. For this, the following details were considered:

- Weights were randomly generated with a discrete value in $\sim U(0, 1000)$,
- Capacity of each dimension was defined as:

$$b_t = tr * \sum_{i=1}^{N} w_{it} \tag{5.14}$$

  where $tr \in [0.25, 0.50, 0.75]$ is the tightness ratio,

- Profits were generated considering:

$$p_i = \frac{\sum_{t=1}^{T} w_{it}}{T} + 500 * q_i \tag{5.15}$$

  where $q_i \sim U(0, 1)$.

Instances with different number of dimensions (5, 10, 30 and 50) and with different number of objects (100, 250 and 500) were proposed. These instances have been severally used in literature to evaluate and compare different approaches designed for solving MKP. Also, we used a subset of these instances to evaluate the inclusion of our strategy into Ant Knapsack.

An important Tabu Search approach based on an oscillation strategy was proposed [GK96]. This strategy tries to balance the exploration and exploitation of the algorithm, "oscillating" between feasibility and infeasibility during the construction of partial instantiations. The inclusion of an object that produces the infeasibility of a partial instantiation is considered a *critical event*. The Tabu List determines if a potential move will be performed considering information about the recency and frequency of *critical events*. Surrogate constraint information is used to compute ratios for guiding the decisions of the algorithm. These ratios are used to decide which objects should

[2]Instances available in `http://people.brunel.ac.uk/~mastjjb/jeb/orlib/mknapinfo.html`

be added and dropped. Other similar Tabu Search approach was proposed in [HF98] including different rules to control the oscillation between feasibility and infeasibility.

A relative mean resource occupation ($\mathcal{O}$) for each object was proposed in [KGOL15]. The relative mean resource occupation of an object i is defined as:

$$\mathcal{O}_i = \frac{(\frac{1}{T}) * \sum_{t=1}^{T} \frac{w_{it}}{b_t}}{p_i} \tag{5.16}$$

The objective was to propose a different surrogate relaxation considering pseudo-utility ratios that could be easier to calculate. This metric was used in a repair operator of an Harmony Search approach similar to the oscillation strategy already described.

Several different metaheuristic approaches are proposed in literature for solving the MKP [Fre04, KGOL15, MP17, LHGL18]. Also, there are approaches for a multi-objective variant of MKP [LT12].

5.2 Ant Knapsack

Ant Knapsack (AK) [ASG04] is a $\mathcal{MMAS}$ proposed for solving MKP. Algorithm 2 shows the structure of Ant Knapsack. AK constructs feasible complete instantiations considering m artificial ants. The first object to be included is randomly selected (lines 5-6). After an object is included into a partial instantiation, the current capacities are updated (line 7) and a list of candidate objects J^k is constructed (line 8). More specifically, J^k contains the objects allowed to be included considering the current capacity in each dimension. This process is repeated until J^k is empty (line 9). Pheromone is evaporated at the end of each iteration (line 15).

As AK is a $\mathcal{MMAS}$ algorithm, H allows best quality solutions to be marked with pheromone. Then, an amount of pheromone ($\Delta\tau$) is deposited on each pair of objects of the best solution found in the current iteration (L_{best_it}):

$$\Delta\tau = \frac{1}{1 + F(L_{best\_found}) - F(L_{best\_it})} \tag{5.17}$$

where L_{best_found} is the best solution found during the execution (line 16). On the other hand, heuristic knowledge is defined as:

$$\eta_{I_P^k}(o_i) = \frac{p_i}{\sum_{t=1}^{T} \frac{w_{it}}{CC_t}} \tag{5.18}$$

considering the profit and the weight in the current capacity. Here, the idea is to give priority to

Algorithm 2 Ant Knapsack algorithm

```
1: InitializePheromoneTrails(τ_max)
2: while Stop criterion is not met do
3:    for ant_k, k = 1 to m do
4:       I_p^k = ∅
5:       o_next ← ChooseRandomlyObject()
6:       I_p^k ← {o_next}
7:       UpdateCurrentCapacities(o_next , b)
8:       CheckCandidates(J^k, b)
9:       while Candidates ≠ ∅ do
10:         o_next ← ChooseAndAddNextObject()
11:         UpdateCurrentCapacities(o_next , b)
12:         CheckCandidates(J^k, b)
13:      end while
14:   end for
15:   EvaporatePheromone(ρ)
16:   DepositPheromone(L_best_it)
17: end while
```

objects with higher profit and less capacity consuming.

5.3 Opposite Learning in Ant Knapsack

During the First Step of our strategy, an ant based algorithm will perform a search process in order to obtain information about an $u\mathcal{D}$-characteristic. This search process is defined considering one of the three methods already described: SOL, WOL and HOL. SOL and WOL methods obtain information about some $u\mathcal{D}$-characteristic considering quality features of the complete instantiations. On the other hand, HOL is focused in obtain information related to some problem-specific feature. This section presents the details of the implementation of our strategy in Ant Knapsack. First, we present details about how each method is implemented for Ant Knapsack. Then, details about how some algorithm-specific components were defined are presented.

5.3.1 Methods for Ant Knapsack

Let us consider AK° as a modified version of Ant Knapsack that will be used to perform the First Step. As we presented in Section 4.2, AK° uses the same representation for P, the same evaluation function F, and the same structure of the *state transition rule* of Ant Knapsack. Moreover, heuristic knowledge and anti-pheromone management is particularly defined for each method, with the objective of obtaining information about an $u\mathcal{D}$-characteristic.

- *Soft Opposite Learning Ant Knapsack* (SOL-AK): this method tries to perform a search process similar as in Ant Knapsack. For this, the heuristic knowledge used during First Step $\eta_i^{AK^\circ}$ is the same as in Ant Knapsack. As presented in Section 4.3, the lowest quality solution of each iteration I_C^w will be marked with anti-pheromone (as in Equation 4.2). As in Ant Knapsack, each pair of objects included in I_C^w will be marked with an amount $\Delta anti\tau$ of anti-pheromone. $\Delta anti\tau$ will be explained in the following section.

- *Worst Opposite Learning Ant Knapsack* (WOL-AK), considers an inversion of the heuristic knowledge during the First Step. Similar as in Equation 4.3, this inversion is defined as:

$$\eta^{AK^\circ}(o_i) = \max_{u \in J^k(i)}(\eta_u^{AK}) + \min_{u \in J^k(i)}(\eta_u^{AK}) - \eta^{AK}(o_i) \tag{5.19}$$

  where the maximum and minimum heuristic values of candidate objects are considered and $\eta^{AK}(o_i)$ is the heuristic value of object i. Notice that the heuristic knowledge $\eta^{AK^\circ}(o_i)$ is calculated considering the same heuristic knowledge function of Ant Knapsack.

  Anti-pheromone management is equally defined as in SOL-AK. Anti-pheromone is deposited on each pair of objects of the lowest quality solution of each iteration. As we mentioned earlier, the objective of this method is to give priority to worst decisions during the First Step.

- *Half Opposite Learning Ant Knapsack* (HOL-AK):

  In general, objects can be heavier in some dimensions than others. As it is currently defined in Ant Knapsack, an object i that has an interesting profit value compared to the other candidate objects, can use several capacity in some dimensions and be lighter in the other ones. This situation is undesired because this kind of objects still be interesting for Ant Knapsack. For this, the $u\mathcal{D}$-characteristic is related to detect assignations that are part of (poor quality) candidate solutions where the way the capacity is being consumed can be improved.

  To detect this situation, we define the heuristic knowledge based in a heuristic presented in Equations 5.7 and 5.8. This heuristic considers *relevance values* (r_t) to evaluate how restricted are some dimensions. Here, the lower the capacity of dimension t and the higher the sum of weights in t, the more restricted is the dimension. This heuristic defines a fixed efficiency value for each object i. To obtain an efficiency value that considers the current state of the partial instantiation being constructed, we considered the candidate allowed objects and the current capacity (CC) in each dimension. The heuristic knowledge of the

HOL method, for a candidate object i, is defined as:

$$\eta_i^{\mathbb{A}^\circ} = \frac{p_i}{\sum_{t=1}^{T} r_t * w_{it}} \tag{5.20}$$

where r_t are the relevance values computed as follows:

$$r_t = \frac{(\sum_{j \in J^k(i)} w_{jt}) - CC_t}{\sum_{j \in J^k(i)} w_{jt}} \tag{5.21}$$

Notice that when all the candidate objects can be included in the dimension t (the sum of their weights are less than the current capacity), the numerator can be a negative value or zero. When this situation happens, the dimension t will not be considered in the efficiency of all objects of the current *intermediate* decision.

The idea is to give priority to objects that use a lower capacity and also have a higher profit. As H is equally defined as in AK, pairs of objects related to poor quality candidate solutions, that use a lower capacity and have interesting profits will be marked with anti-pheromone.

5.3.2 Components in Ant Knapsack

It is necessary to define four components related with the management of anti-pheromone during First Step:

- The amount of anti-pheromone $\Delta anti\tau$ is defined similarly as in Ant Knapsack:

$$\Delta anti\tau = \frac{1}{1 + |F(L_{worst\_found}) - F(L_{worst\_it})|} \tag{5.22}$$

where L_{worst_it} is the worst solution found in the current iteration and L_{worst_found} is the worst solution found in during the execution.

- As Ant Knapsack is a $\mathcal{MMAS}$ algorithm, the *translation rule* was defined considering the maximum value of pheromone allowed τ_{max}. For each cell of the anti-pheromone matrix, its pheromone equivalent is calculated as:

$$\tau_{ij} = \tau_{max} - anti\tau_{ij} \tag{5.23}$$

where τ_{ij} is the translated amount of anti-pheromone into pheromone. This will produce that the higher the amount of anti-pheromone, the lower the amount of pheromone translated.

- The artificial ants allowed to deposit anti-pheromone are also similarly defined as in Ant Knapsack. In this case, the artificial ants that construct the poorest quality solution of each iteration will be allowed to deposit anti-pheromone.
- The percentage B defines the percentage of iterations that will be assigned for the First Step. The value of B can be experimentally defined or using a tuner algorithm.

In the following section, we present the experimental evaluation of this strategy considering all these methods.

5.4 Experiments

In this section we present the experiments performed in order to evaluate our approaches. The main objective of these experiments is to compare the quality of solutions found by Ant Knapsack versus the quality found when an OIL strategy is incorporated to AK. For these experiments, we considered 30 independent executions per instance, each with 60000 evaluations.

5.4.1 Instances

We considered 90 large-scale benchmarks instances from the OR-Library proposed by Chu and Beasley.[3] These instances were designed considering the following details:

- Weights are randomly generated with a discrete value in $\sim U(0, 1000)$,
- Capacity of each dimension is defined as:

$$b_t = tr * \sum_{i=1}^{N} w_{it} \tag{5.24}$$

  where $tr \in [0.25, 0.50, 0.75]$ is the tightness ratio,

- Profits are generated considering:

$$p_i = \frac{\sum_{t=1}^{T} w_{it}}{T} + 500 * q_i \tag{5.25}$$

  where $q_i \sim U(0, 1)$.

[3]Instances are available in http://www.cs.nott.ac.uk/∼jqd/mkp/index.html

Set	T	N	tr
5x500	5	500	0.25
			0.50
			0.75
10x500	10	500	0.25
			0.50
			0.75
30x500	30	500	0.25
			0.50
			0.75

Table 5.1: Details of each set of instances used for our experiments

Algorithm	First Step					Second Step				
	m	α	β	ρ	B	m	α	β	ρ	τ_{max}
AK	-	-	-	-	-	50	2.3688	5.6	0.009	10.3
SOL-AK	60	8.0	9.1	0.206	0.010	50	2.3688	5.6	0.009	10.3
WOL-AK	30	5.2	6.7	0.369	0.010	50	2.3688	5.6	0.009	10.3
HOL-AK	45	18.3	0.1	0.151	0.085	50	2.3688	5.6	0.009	10.3

Table 5.2: Parameter values used for these experiments

Table 5.1 shows the three sets of instances used in our experiments. Each set contains 30 instances, considering 10 instances for each tuple (T, N, tr). These instances have been severally used to evaluate and compare different approaches on literature designed for solving MKP.

5.4.2 Parameter Values

Parameter values for Ant Knapsack and our approaches were defined using a tuner algorithm called Evolutionary Calibrator (EVOCA) [MR13]. Table 5.2 shows the obtained values by the tuning process for each algorithm, after an average of 5000 evaluations of the tuner algorithm. Randomly selected instances from the three sets were considered for this process. About the amount of resources for the *First Step*, for SOL and WOL methods the 1% of the evaluations will be considered ($60000 * 0.01 = 600$ evaluations). On the other hand, 5114 evaluations will be considered for the *First Step* in the HOL method.

5.4.3 Results

To compare Ant Knapsack and our approaches, we consider a percentage gap distance between the Best Known (BK) solution and a solution found S_{found} calculated as follows:[4]

[4]Best Known solutions are available in http://www.eecs.qmul.ac.uk/∼jdrake/bestresults.html

T	N	tr	AK	SOL-AK	WOL-AK	HOL-AK
5	500	0.25	0.109	**0.093**	0.102	0.095
		0.50	0.041	0.041	**0.037**	0.049
		0.75	0.019	0.021	0.017	**0.015**
AVG 5x500			0.056	**0.052**	**0.052**	0.053
10	500	0.25	0.227	**0.215**	0.237	0.226
		0.50	**0.089**	0.094	0.102	0.093
		0.75	0.040	**0.039**	0.040	**0.039**
AVG 10x500			0.119	**0.116**	0.126	0.119
30	500	0.25	0.512	**0.485**	0.516	0.530
		0.50	0.211	**0.203**	0.217	0.226
		0.75	**0.101**	0.102	0.106	0.110
AVG 30x500			0.275	**0.263**	0.280	0.289
AVG			0.150	**0.144**	0.153	0.154

Table 5.3: Average percentage gap between the Best Known solution and the best solution found, considering instances from each tuple (T, N, tr)

$$gap = 100 * \frac{(BK - S_{Found})}{BK} \tag{5.26}$$

Table 5.3 shows the average percentage gap between the BK solution and the best reached solution by each algorithm, over all the instances in each tuple (T, N, tr). Here, bold numbers in the table show when an algorithm outperforms another approach. Also, the average (AVG) percentage gap for each set and an overall average are shown. Considering the set 5×500, results show that the performance of AK was slightly improved using these three methods. Moreover, the greatest improvements were obtained using the SOL and WOL methods, where the average gap of AK was reduced from 0.056 to 0.052.

About the set 10×500, the information obtained by the SOL method was useful for AK. Here, the average percentage gap of AK was reduced from 0.119 to 0.116. Also, the search process of AK was slightly worsened using the information provided by the WOL method.

In the set 30×500, the information obtained by the SOL method was useful for AK. Here, the average percentage gap of AK was reduced from 0.275 to 0.263 In summary, SOL-AK obtained the lowest overall average percentage gap (0.144), followed by AK (0.150), WOL-AK (0.153) and HOL-AK (0.154).

Table 5.4 shows the average percentage gap between the Best Known solution and the average quality of the 30 independent executions per instance, over all the instances in each tuple (T, N, tr).

T	N	tr	AK	SOL-AK	WOL-AK	HOL-AK
5	500	0.25	0.222	**0.216**	0.221	0.228
		0.50	0.098	**0.096**	0.414	0.099
		0.75	0.049	**0.047**	**0.047**	0.046
AVG 5x500			0.123	**0.120**	0.227	0.124
10	500	0.25	**0.427**	0.430	0.438	0.456
		0.50	0.182	0.181	**0.173**	0.187
		0.75	0.086	0.083	**0.080**	0.084
AVG 10x500			0.232	0.231	**0.230**	0.242
30	500	0.25	0.922	**0.864**	0.896	0.906
		0.50	0.354	**0.351**	0.360	0.366
		0.75	**0.171**	0.175	0.176	0.176
AVG 30x500			0.483	**0.464**	0.478	0.482
AVG			0.279	**0.272**	0.312	0.283

Table 5.4: Average percentage gap between the Best Known solution and the average quality of 30 seeds per instance, considering results from each tuple (T, N, tr)

In the set 5×500, the information provided by the SOL method was useful to AK to decrease its average percentage gap. Here, the average percentage gap of AK was reduced from 0.123 to 0.120. For the set 10×500, the information provided by the WOL and SOL methods were useful for AK. On the other hand, HOL-AK was outperformed by AK. About the set 30×500, our proposed algorithms outperformed AK obtaining a lower average overall percentage gap. Finally, considering all the executions per instance, the SOL method obtained the lowest overall average percentage gap (0.272), followed by AK (0.279), HOL-AK (0.283) and WOL-AK (0.312).[5]

Table 5.5 shows the average execution time of each algorithm, over all the instances per tuple (T, N, tr). In general, the execution times are higher using our strategies. However, the execution times of WOL-AK are slightly lower than the AK's.

5.4.4 Statistical Analysis

We have used the pair-wise Wilcoxon non-parametric test to compare the performance of Ant Knapsack and our approaches. For this, we used the percentage gap to the best known solution of all the independent executions of each algorithm on every instance. The hypotheses considered in this test are:

- H_0: Algorithm X found same quality solutions as Algorithm Y

[5]Tables with the obtained results in the 90 large-scale instances are available in appendix A

T	N	tr	AK	SOL-AK	WOL-AK	HOL-AK
5	500	0.25	2977	3451	2917	3469
		0.50	4755	5548	5607	5090
		0.75	6490	6997	4765	6399
AVG 5x500			4741	5332	4429	4986
10	500	0.25	3233	3230	2760	3362
		0.50	5202	5524	4706	5489
		0.75	7483	7064	6081	6838
AVG 10x100			5306	5273	4515	5230
30	500	0.25	3309	3266	2839	3286
		0.50	5893	5788	4961	5712
		0.75	7207	7564	6540	7509
AVG 30x500			5470	5539	4780	5502
AVG			5172	5381	4575	5239

Table 5.5: Average execution times of each algorithm (in seconds)

Comparison	Negative Ranks	Positive Ranks	p-value
SOL-AK - AK	1464	1236	0.01
WOL-AK - AK	1400	1300	0.13
HOL-AK - AK	1303	1397	0.00

Table 5.6: Results of the Wilcoxon tests

- H_1: Algorithm X found different quality solutions than Algorithm Y

Table 5.6 shows the results of the tests. Considering a confidence level of 95%, results show that the SOL and HOL approaches are statistically different from AK. On the other hand, no conclusion about the statistical difference between WOL and AK can be made. As the MKP is a maximization problem, the negative ranks (NR) show the cases where our approaches outperform AK. Positive ranks show the cases where AK outperforms our strategies. All these computations were done using the statistical software package $PSPP$.

5.4.5 Boxplots

This section presents boxplots to show the dispersion of the solutions found by each algorithm. Boxplots per set of instances are presented, considering the percentage gap to the Best Known solution and the boxes are grouped by the tightness ratio value ($tr \in [0.25, 0.50, 0.75]$). Moreover, all independent executions of each algorithm are considered. Figure 5.1 shows boxplots for the set 5×500. In general, medians and boxes of the four algorithms are similar for all values of tr. However, for $tr = 0.25$ and $tr = 0.50$, the median of SOL method is the closer one to zero.

Moreover, considering $tr = 0.25$, lower whiskers of our approaches are closer to zero than AK ones.

Figure 5.2 shows boxplots for the set 10×500. For all tr values, boxes of AK are the closest to zero. However, considering $tr = 0.50$, the size of the boxes of SOL and WOL method are smaller than the box of AK.

Figure 5.3 shows boxplots for the set 30×500. Considering $tr = 0.25$, medians of our approaches are the closest to zero. Moreover, in the same case, the boxes of are smaller than the box of AK.

In general, for the three sets, the higher the tr value the closer to zero the percentage gap reached by all the algorithms. Also, boxes are smaller when the value of tr increases. Moreover, the performance and the distribution of the solutions obtained by the four algorithms is similar when $tr = 0.75$. On the other hand, the main differences in the boxes sizes and whiskers are presented when $tr = 0.25$.

5.4.6 Comparison with the Literature

This section presents a comparison of our approaches with some existing approaches from the literature. We considered two approaches: *TS+LP*, a Tabu Search algorithm combined with Linear Programming [VV05] and *CF-LAS*, a Selection Hyperheuristic framework [DÖB16]. To the best of our knowledge, *TS+LP* is known as the best metaheuristic algorithm for solving MKP instances with $N = 500$. Table 5.7 shows the comparison of these approaches with AK and SOL-AK, considering the percentage gap between the Best Known solution and the best reached solution per algorithm. In general, the lowest percentage gaps are obtained by *TS+LP*. However, AK and SOL-AK algorithms are competitive to both state-of-the-art algorithms, outperforming *CF-LAS* and obtaining an overall percentage gap near to *TS+LP*. Moreover, the use of the information provided by the SOL method allows AK to get closer to the performance of the best known algorithm for solving these problem instances.

5.4.7 Discussion

This section presents an analysis of our OIL strategy applied in AK. First, we present a comparison between the typical initialization of AK and using our strategies to initialize AK. The idea is to visualize the initial matrices produced by one of our strategies and AK.

Then, we present Fitness-Distance plots in order to visualize the effect of using opposite information, in terms of the structure and quality of the obtained solutions. Several studies have been proposed in literature to understand the structure of the search space of combinatorial prob-

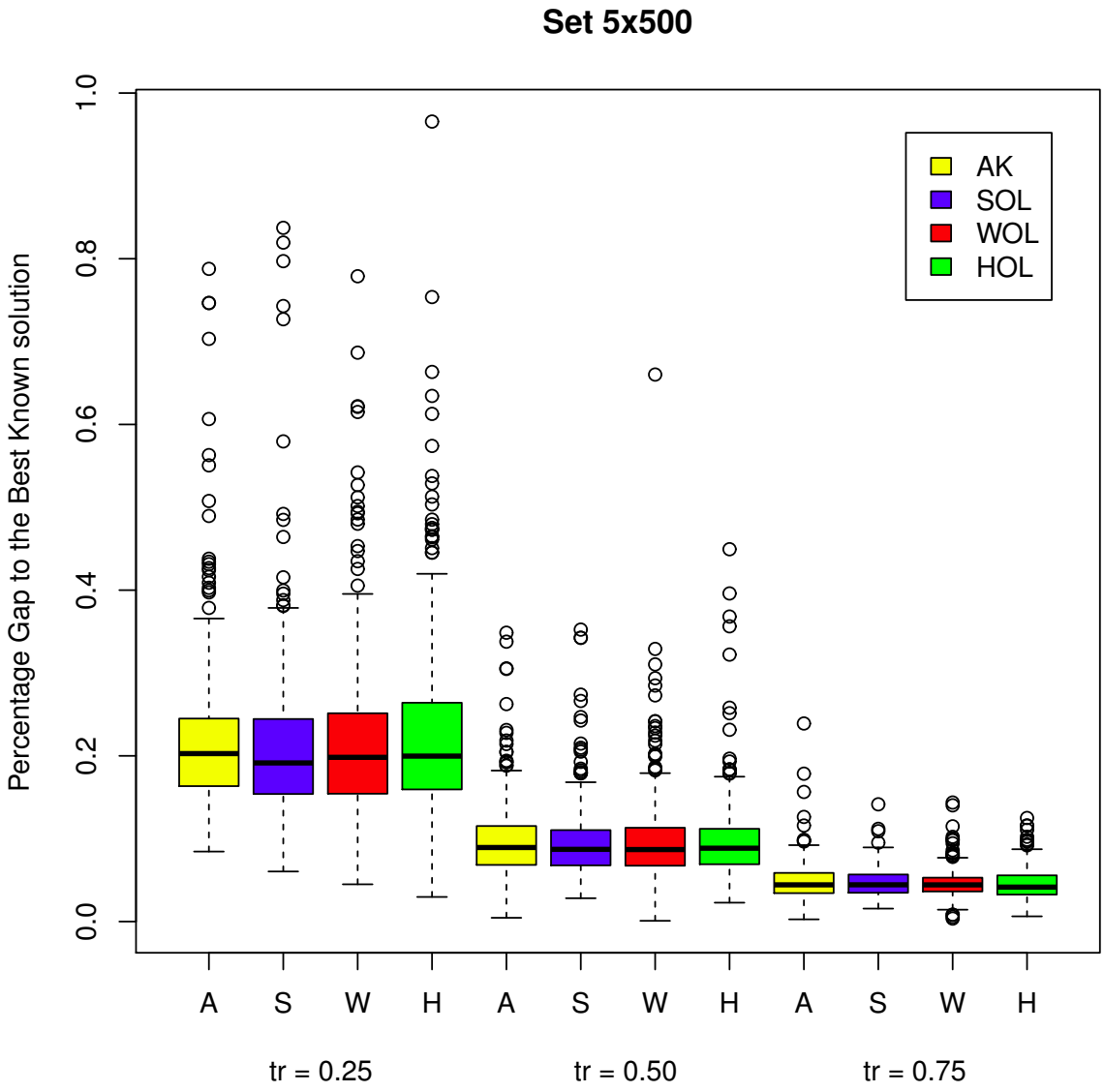


Figure 5.1: Boxplot of the set 5×500

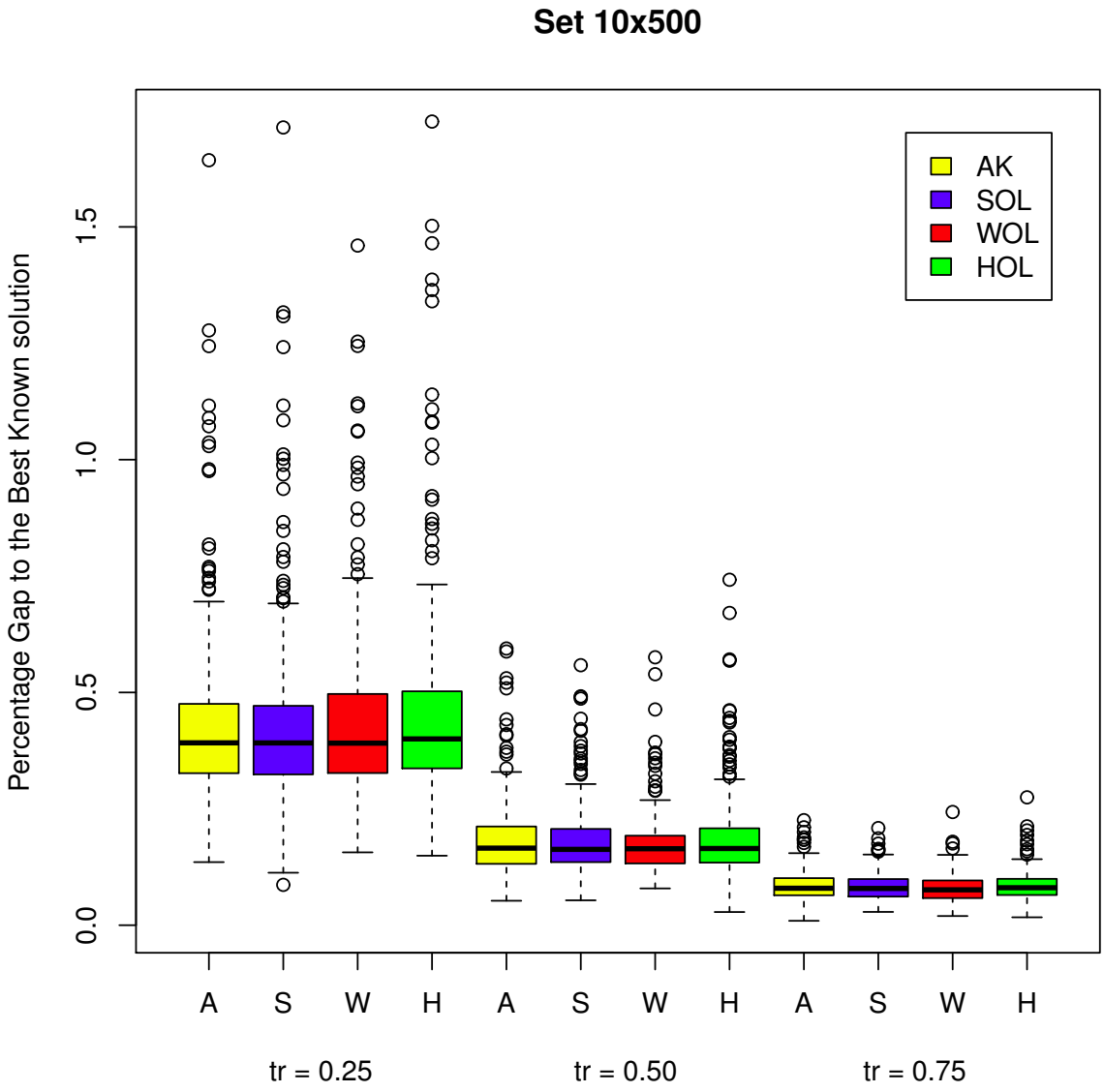


Figure 5.2: Boxplot of the set 10×500

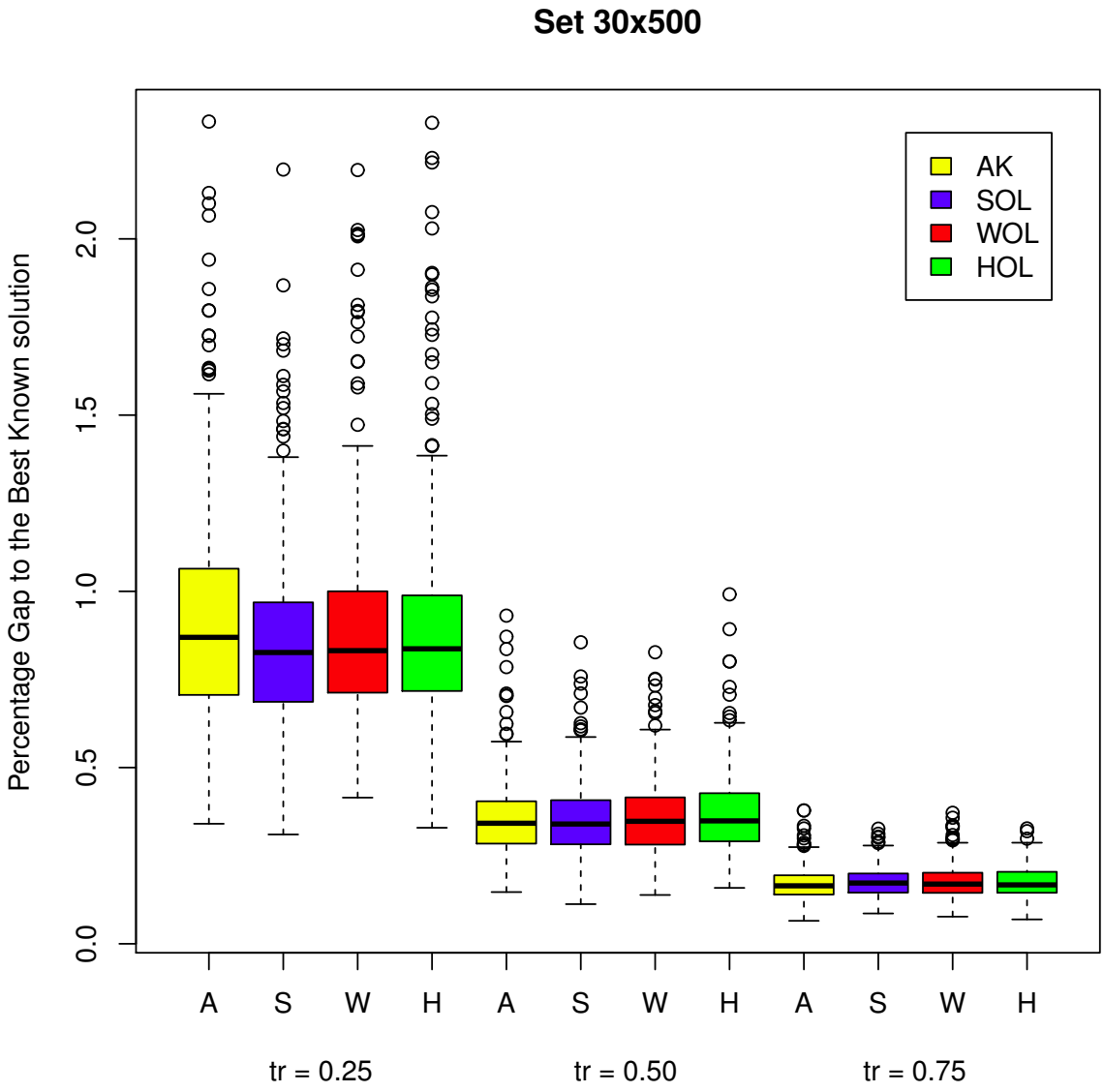


Figure 5.3: Boxplot of the set 30×500

T	N	tr	AK	SOL-AK	TS+LP	CF-LAS
5	500	0.25	0.109	0.093	**0.070**	0.190
		0.50	0.041	0.041	**0.040**	0.100
		0.75	**0.019**	0.021	0.020	0.060
10	500	0.25	0.227	0.215	**0.170**	0.400
		0.50	0.089	0.094	**0.080**	0.180
		0.75	0.040	**0.039**	0.060	0.120
30	500	0.25	0.512	0.485	**0.480**	0.920
		0.50	0.211	**0.203**	0.210	0.390
		0.75	**0.101**	0.102	0.140	0.230
AVG 500			0.150	0.144	**0.140**	0.290

Table 5.7: Comparison with the literature - Percentage gap between the best known solution and the best quality solution obtained by each algorithm.

lems using Fitness Landscape analysis techniques [OQB09, Wat10, OTVD08, TPC08, TPC06]. In this case, as most of the optimal solutions are structurally unknown, we computed the hamming distance between each solution obtained by AK and the best reached (BR) solution by all the algorithms here evaluated. The idea is to show how the use of opposite information can produce a modification of the search process of AK, considering new and different intermediate decisions.

Figure 5.4 shows two pheromone matrices obtained by AK without using opposite information. We used a MKP instance named *MKNAP_1_6* that considers 39 objects and 5 dimensions. As each cell of the matrix is a pair of objects, to better understanding of the matrices, we select an instance with a small number of dimensions and objects.[6] Also, the maximum allowed value of pheromone is $\tau_{max} = 0.1$ and the minimum allowed value is $\tau_{min} = 0.01$. For both matrices, each cell represents the amount of pheromone in a pair of objects where the darker the color the higher the amount of pheromone. In the left part of the figure, the initial pheromone matrix is shown, where all values are equal to $\tau_{max} = 0.1$. In the right part of the figure, the final pheromone matrix of AK is shown.

Figure 5.5 shows two pheromone matrices using opposite information in AK. In the left part of the figure, an anti-pheromone matrix obtained by the SOL method is shown (that will be the initial pheromone matrix for AK). Here, some pairs of objects will have small initial pheromone values because they were part of candidate solutions related to an $u\mathcal{D}$-characteristic. In the right part of the figure, the final pheromone matrix obtained by AK is shown. Comparing the two final matrices, the SOL matrix allows AK to include objects that were not considered originally or were discarded by the algorithm. More specifically, this can be seen in some grey regions in the final

[6]Matrices obtained for a larger size instance can be found in appendix B

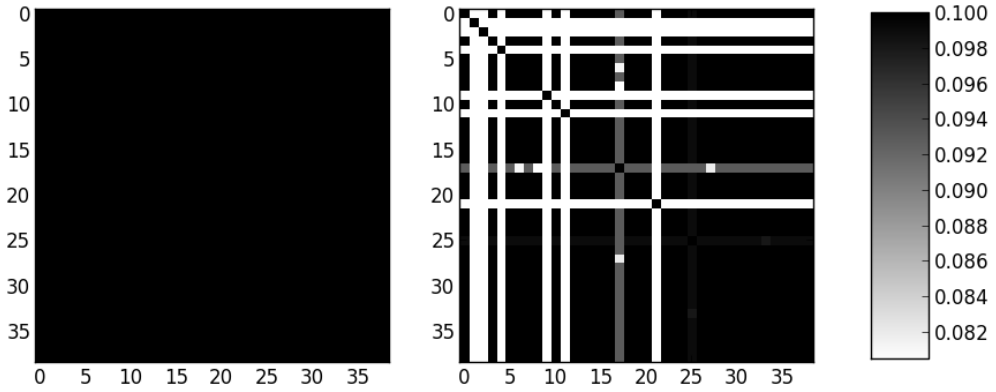


Figure 5.4: Initial and final matrices for Ant Knapsack - Instance MKNAP_1_6

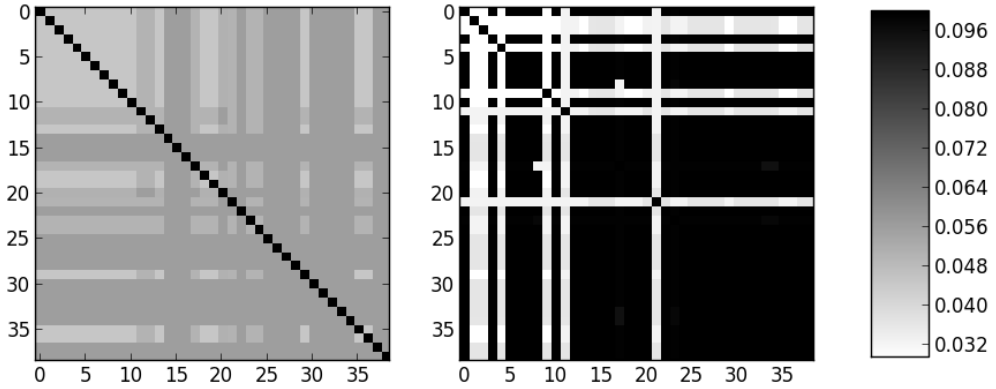


Figure 5.5: Initial and final matrices using SOL in Ant Knapsack - Instance MKNAP_1_6

matrix using opposite information. Moreover, some cells that are white in the first final matrix are now in a darker color when opposite information is used.

Figures 5.6 and 5.7 shows fitness-distance plots for an instance from the 30×500. For our strategies, we only present the solutions obtained during the *Second Step*. First, the increment of the exploration of AK can be seen in the three plots. Solutions obtained by Ant Knapsack are clustered in two regions: near to a 12% gap and near to a 2.5% gap. In the left plot, the percentage gaps obtained using the information provided by the SOL method range from 16% and 0%. In the center plot, the percentage gaps of the solutions found using the WOL method range from 17% and 0.5%. In the right plot, the percentage gaps obtained using the HOL method range between 15% and 0.5%. These plots show how the use of opposite information allows to visit different solutions from the ones obtained by the original algorithm, in terms of their structure and quality.

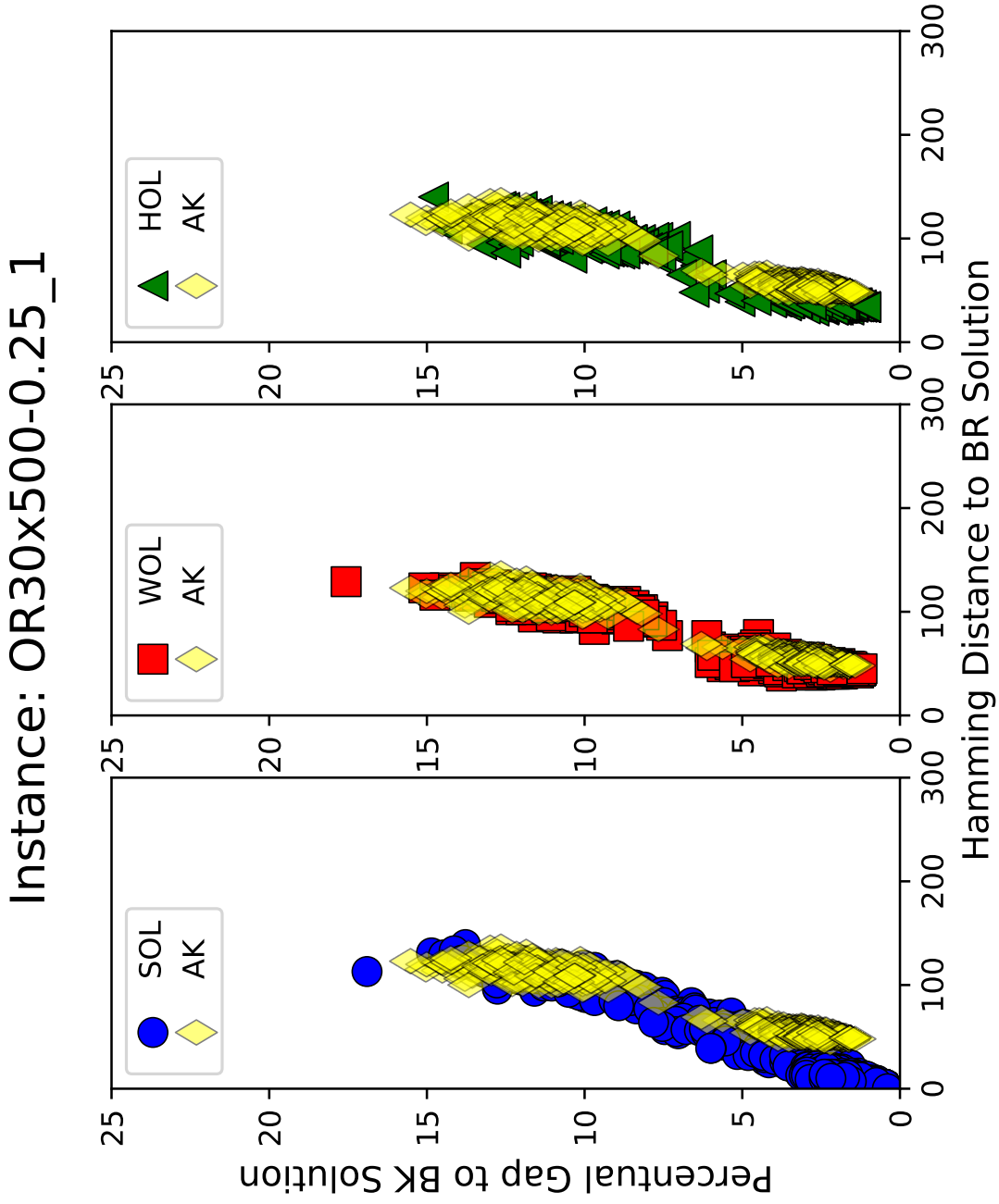


Figure 5.6: Fitness-Distance plots with Ant Knapsack and each Method separately for an instance from the set 5×500

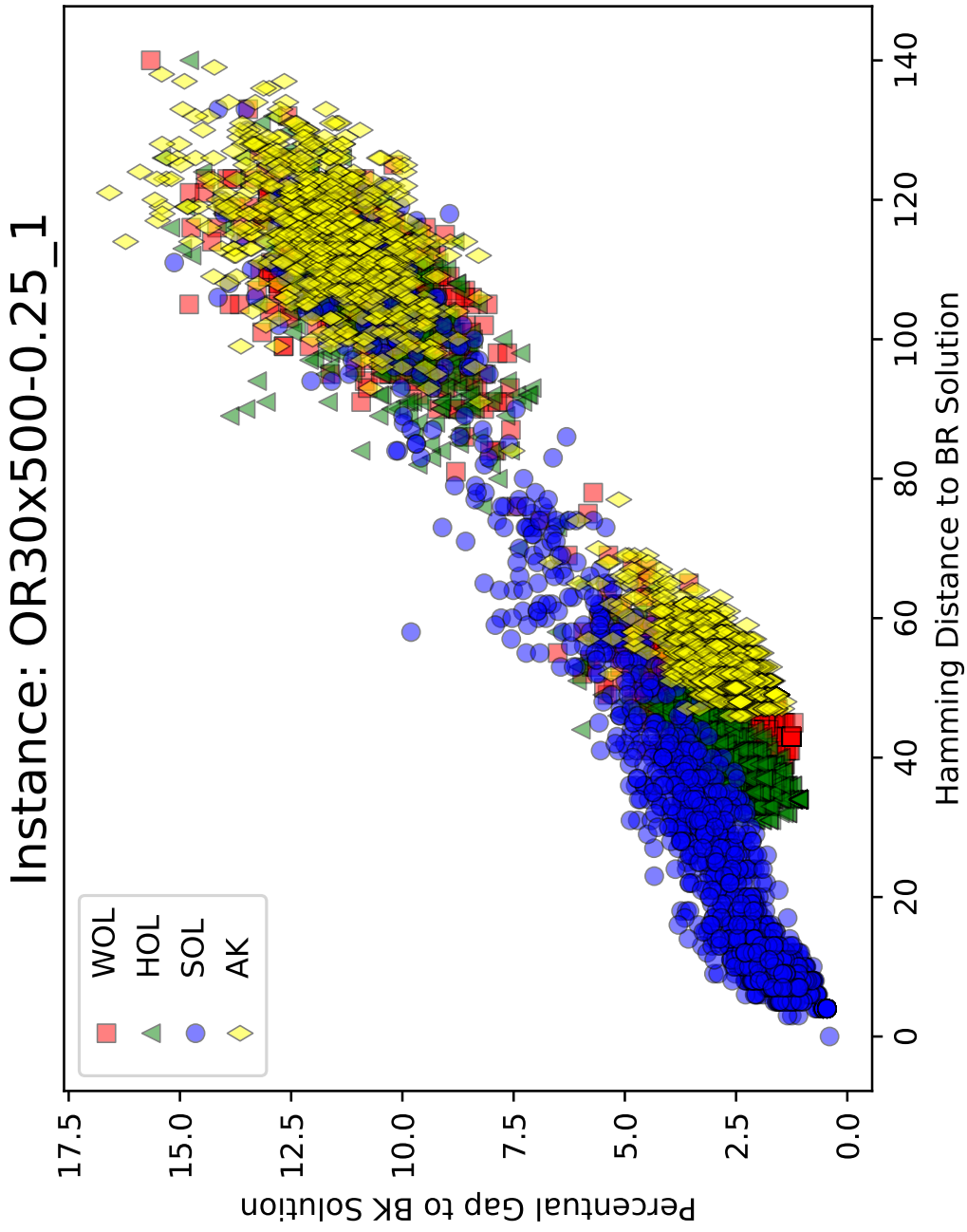


Figure 5.7: Fitness-Distance plots with all approaches for an instance from the set 5×500

5.5 Conclusions

In this chapter we present the evaluation of the inclusion of our strategy in Ant Knapsack, an ant-based algorithm proposed for solving the Multidimensional Knapsack Problem. The main objective of these experiments is to compare the quality of the solutions found by Ant Knapsack versus the quality found when our strategy is incorporated to AK. We evaluated the three different methods to learn about $u\mathcal{D}$-characteristics: *Soft Opposite-Learning* (SOL), *Worst Opposite-Learning* (WOL) and *Half Opposite-Learning* (HOL).

In these experiments, three sets of large-scale benchmark instances were considered ($N = 500$): 5×500, 10×500 and 30×500. These sets differ in the number of dimensions ($5, 10$ and 30) and also, different tightness ratio values were considered ($tr \in [0.25, 0.50, 0.75]$). Results show that, in general, the SOL methods allowed Ant Knapsack to reach better quality solutions. The highest improvements were obtained for the set 30×500, where the average percentage gap of Ant Knapsack was decreased from 0.275 to 0.263. Considering the three sets of instances, the information provided by the SOL method was the more effective one.

A comparison with two state-of-the-art approaches is presented: *TS+LP* [VV05] and *CF-LAS* [DÖB16]. To the best of our knowledge, *TS+LP* is known as the best metaheuristic algorithm for solving MKP instances with $N = 500$. Results show that Ant Knapsack, using the opposite information provided by the SOL method, can be competitive to both algorithms.

Statistical analysis, boxplots and fitness-distance plots were presented to show how the search process of Ant Knapsack is modified using opposite information, obtaining different solutions in terms of its quality and its structure.

Two articles entitled "Using Anti-pheromone to Identify Core Objects for Multidimensional Knapsack Problems: A Two-step Ants Based Approach" [RMRM15] and "Learning from the opposite: Strategies for Ants that solve Multidimensional Knapsack Problem" [RMRM16b] were published presenting preliminary results and experiments of the application of our strategy in Ant Knapsack. In the following chapter, the application and evaluation of our strategies in Focused Ant Solver is presented, an ant-based algorithm proposed for solving Constraint Satisfaction Problems.

Chapter 6

OL for Constraint Satisfaction Problems

This chapter presents the application and evaluation of our strategy in Focused Ant Solver. In order to compare and evaluate our strategies in well-designed algorithms, we proposed Focused Ant Solver (FAS) [RMRN18] as a contribution of this Thesis. FAS is an improved version of the well-known Ant Solver [Sol02] that was introduced in Section 3.4.2. Section 6.2 presents a description of FAS, considering the main components and details of the algorithm.

As Ant Solver, FAS was proposed for solving Constraint Satisfaction Problems. The study of Constraint Satisfaction Problems (CSP) begins in the Artificial Intelligence community in early 1970's by Waltz [Wal72], Montanari [Mon74] and Mackworth [Mac77]. In these almost 50 years of research, CSPs have been severally studied and also, different techniques have been proposed, considering exact and approximate approaches. This chapter presents a formal definition of CSPs, some particular details related to the difficulty of these problems and a brief review of the literature.

Similarly as proposed in the last chapter, to implement our strategy, FAS is considered as the A algorithm. In order to obtain information about some undesirable characteristic, a modified version of FAS is considered for the First Step (A°). Moreover, the three proposed methods are implemented and evaluated. Section 6.3 details the inclusion of our strategy in Focused Ant Solver.

An experimental evaluation of the inclusion of our strategy in Focused Ant Solver is presented in Section 6.4. The objective is to evaluate how the information about some $u\mathcal{D}$-characteristic can improve the search process of Focused Ant Solver. For this, instances from the transition

phase were randomly generated. Results for these instances, statistical analysis, boxplots and fitness-distance plots are presented.

6.1 Constraint Satisfaction Problems

A Constraint Satisfaction Problem (CSP) can be defined as a triple $P = (X, D, C)$ where:

- X is the set of variable $X = \{x_1, x_2, \dots x_n\}$,
- D is the set of variable domains $D = \{D_1, D_2, \dots, D_n\}$,
- Ω is the set of constraints among variables.

This definition is similar to the Combinatorial Problem described in Definition 3.2.1, but in this case, F is not considered. However, a complete instantiation I_C can be defined as presented in Equation 3.2 and a partial instantation I_P as in Equation 3.3. A constraint $\Omega_{ijk\dots p} \in \Omega$ is a relationship between some variables $x_i, x_j, x_k, \dots, x_p$ that defines the subset of possible combinations of values of $x_i, x_j, x_k, \dots, x_p$. In other words, $\Omega_{ijk\dots p} \subseteq D_i \times D_j \times D_k \cdots \times D_p$ specifies the combination of values which the constraint defines as incompatible. Each assignment of values to variables ruled out is called a *nogood* [GMP+01].

A *feasible* solution to a CSP is a complete instantiation, in such a way that every constraint is satisfied. Here, the problem P is considered as *satisfiable*. On the other hand, if there is no assignment of values to variables from their respective domains for which all constraints are satisfied, then the problem is *unsatisfiable*.

In this chapter, binary CSPs are used to evaluate FAS and our approaches. Thus, each constraint can affect only two variables and P can be represented as a *constraint graph* (G_Ω). Here, each vertex represents a variable and an edge represents the existence of a constraint between two vertices.

Randomly-generated binary CSPs were used to evaluate our strategy in Focused Ant Solver. To generate these instances, four parameters $\langle n, m, p_1, p_2 \rangle$ should be defined:

- n defines the number of variables,
- m corresponds to the number of possible values in the domain of each variable,
- $p_1 \in [0, 1]$ represents the connectivity (or density) of the constraint graph, i. e. it determines the number of constraints and

- $p_2 \in [0, 1]$ corresponds to the tightness of the constraints, i. e. the probability that a pair of values is incompatible.

There exist some models to generate these problems, that differ in how the constraint graph is generated and how incompatible values are chosen. These models are:

- Model A: each one of the $n * (n-1)/2$ possible edges in G_Ω is independently selected with probability p_1, and for each selected edge, each one of the m^2 possible pair of values is selected as incompatible with probability p_2.
- Model B: exactly $p_1 * n * (n-1)/2$ edges are randomly selected for G_Ω, and exactly $p_2 * m^2$ pairs of values are randomly selected.
- Model C: each one of the $n * (n-1)/2$ possible edges in G_Ω is independently selected with probability p_1, and exactly $p_2 * m^2$ pairs of values are randomly selected.
- Model D: exactly $p_1 * n * (n-1)/2$ edges are randomly selected for G_Ω, and each one of the m^2 possible pair of values is selected as incompatible with probability p_2.
- Model E: exactly $p * m^2 * n * (n-1)/2$ of *nogoods* are selected uniformly, independently and with repetitions.

Notice that p_1 and p_2 are considered, in some cases, as a portion or as a probability.

There is a situation with randomly-generated problems related to *flawed* values/variables. A value for a variable is *flawed* if, when the value is assigned to the variable, there exists an adjacent variable in the constraint graph that cannot be assigned a value without violating the constraint between the two variables. As a consequence, a problem with a flawed variable cannot have a solution. In [AKK+97], authors prove that if $p_2 \geq 1/m$ then, as $n \to \infty$, it is probable that all values of at least one variable will be *flawed*.

In general, CSPs are considered NP-Complete problems [BPS99]. Empirical studies showed that the hardest instances to solve, in a set of NP-complete problems, often occur around a transition from solubility to insolubility [CKT91]. Considering the tuple $\langle n, m, p_1, p_2 \rangle$, some critical values produce a boundary that divides the space of problems into two regions: under-constrained problems (high number of solutions) and over-constrained problems (where problems cannot have a solution). Hard problems occur in the *transition phase*, on the boundary between these regions. It is important to remark that the transition phase is an algorithm-independent feature.

The transition phase can be predicted considering a *constrainedness* measure (κ) [GMPW96]. In particular, for binary CSPs (model B), κ can be defined as follows [MPSW98]:

$$\kappa(n, m, p_1, p_2) = \frac{n-1}{2} * p_1 * \log_m(\frac{1}{1-p_2}) \tag{6.1}$$

Considering that $\kappa \in [0, \infty[$, when $\kappa \approx 0$, problems are under-constrained and soluble. On the other hand, when $\kappa \approx \infty$, problems are over-constrained and insoluble. However, when $\kappa \approx 1$ problems are between solubility and insolubility and it is difficult to find a solution or to prove there are none.

Several Real-world or Artificial Intelligence problems can be considered as Constraint Satisfaction Problems. Scheduling and planning problems, floor plan design, satisfiability problems, graph problems, circuit design, diagnostic reasoning, are some examples of problems that can be viewed as CSPs [Kum92]. As a consequence, several approaches have been proposed to solve these long list of problems. However, it is important to mention some important proposed methods for solving CSPs.

First, propagating constraints methods have been proposed [Mac77]. The idea is to check the consistency of nodes or arcs in a problem instance, considering the feasible values allowed by the constraints. In example, if the domain D_i of a variable x_i contains a value a that does not satisfy a unary constraint on x_i, then the instantiation of $x_i = a$ will always result as an immediate failure. Similar situation can be produced in arcs when values are *flawed*.

The most basic method to solve CSPs (and other Combinatorial Problems) is Generate-and-Test, where each possible combination of variable-values is systematically generated and tested to evaluate if all constraints are satisfied. This method evaluates a number of combinations equal to the Cartesian product of all the variable domains.

The following methods perform a search tree to represent the current state of the search. *"Each node of the tree corresponds to a partial instantiation in which the values of some variables are determined (past variables). The values of future variables remain to be decided"* [BPS99]. Backtracking is a method where variables are sequentially instantiated and constraints are checked when involved variables are instantiated. When a partial instantiation violates any constraint, backtracking is performed to the most recently instantiated variable that still has alternatives available. Notice that the algorithm only checks the constraints between the current variable and the past variables.

Forward Checking (FC) [HE80] is a look-ahead algorithm that checks the constraints between the current, past and future variables. When a value is assigned to the current variable, any value in the domain of a future variable in conflict with the assignment is (temporary) removed from the domain. An inconsistency is found when a domain becomes empty. As backtracking, when this situation happens, the algorithm continues with the most recently instantiated variable that still has alternatives available.

Improved versions of these algorithms have been proposed like Conflict-Directed Backjumping (CBJ) [Pro93a], Graph-Based Backjumping (GBJ), Maintaining Arc Consistency (MAC) [SF94] and also, some hybrid techniques like FC-CBJ [Pro93b], among others. Variable and value ordering heuristics have been proposed with the objective of studying the sparseness found in the constraint network and the simplicity of the tree-structured CSPs [DP87].

There are several metaheuristic approaches for solving CSPs like ant-based approaches [Sol02, RMRN18], Tabu Search [DBFS$^+$00, GH97, NI98], Simulated Annealing [FCM$^+$92], Genetic Algorithms [KMN95], among others. In the following section, a description of Focused Ant Solver is presented.

6.2 Focused Ant Solver

Focused Ant Solver (FAS) [RMRN18] is an ant-based algorithm proposed for solving CSPs, based in the well-known Ant Solver (AS) [Sol02]. FAS searches for a solution that minimizes the number of unsatisfied constraints. The construction of a solution considers two decisions at each assignment: the selection of the next variable, according to some given heuristic, and the selection of the value, made probabilistically according to the transition rule. The pheromone is deposited on a binary graph whose nodes $\langle X_i, v\rangle$ represent the assignment of value v to variable X_i and the edges between two nodes $(\langle X_i, v\rangle, \langle X_j, w\rangle)$ represent those simultaneous assignments of values.

Equation 6.2 shows the transition rule used by FAS. Here, it does not only depend on local relations between the candidate node and the last visited node, but also on a global relation between the candidate node and the whole set of visited nodes I_p. Hence, the pheromone factor of node $\langle X_j, w\rangle$ depends on pheromone deposited on all edges between $\langle X_j, w\rangle$ and the nodes in I_p.

$$p_{I_p}(\langle X_j, w\rangle) = \frac{[\tau_{I_p}(\langle X_j, w\rangle)]^\alpha * [\eta_{I_p}(\langle X_j, w\rangle)]^\beta}{\sum_{v \in D(X_j)} [\tau_{I_p}(\langle X_j, v\rangle)]^\alpha * [\eta_{I_p}(\langle X_j, v\rangle)]^\beta} \tag{6.2}$$

Equation 6.3 shows the heuristic knowledge considered in Focused Ant Solver.

$$\eta_{I_p}(\langle X_j, w\rangle) = \frac{1}{1 + F(\langle X_j, w\rangle \cup I_p) - F(I_p)} + \frac{1}{(1 + \mathcal{E}(\langle X_j, w\rangle, I_P))^2} \tag{6.3}$$

where $\langle X_j, w\rangle$ is a candidate assignment, F is the evaluation function of the algorithm that counts the number of conflicts of a partial or complete instantiation and $\mathcal{E}(\langle X_j, w\rangle, I_P)$ is defined as:

$$\mathcal{E}(\langle X_j, w\rangle, I_P) = \sum_{\langle X_r, a\rangle \in I_P} \mathcal{U}(\langle X_j, w\rangle, \langle X_r, a\rangle) \tag{6.4}$$

where $\langle X_r, a\rangle$ is an assignment in I_P that has a conflict with $\langle X_j, w\rangle$ and $\mathcal{U}(\langle X_j, w\rangle, \langle X_r, a\rangle)$ the number of pairs of values in conflict between two variables X_j and X_r.

The objective is to give priority to assignments that have a low number of conflicts and also, conflicts with a low number of incompatible pairs of values.

In FAS, the heuristic H determines that pheromone will be deposited on the feasible edges of the complete instantiation I_C that has the lowest number of conflicts and also the lowest pairs of values in conflict. To calculate the number of pairs of values in conflict in a complete instantiation, $\mathcal{E}$ function is redefined considering each pair of variables in conflict:

$$\mathcal{E}(I_C) = \sum_{(\langle X_j, w\rangle, \langle X_r, a\rangle) \in I_C} \mathcal{U}(\langle X_j, w\rangle, \langle X_r, a\rangle) \tag{6.5}$$

where I_C is a complete instantiation, X_j and X_r are two variables in conflict because the simultaneous instantiation $(\langle X_j, w\rangle, \langle X_r, a\rangle)$ is considered in I_C. Then, an amount of pheromone Δ_τ will be deposited defined by:

$$\Delta_\tau(I_C^{\mathcal{E}}, (\langle X_j, w\rangle, \langle X_r, a\rangle)) = \frac{1}{F(I_C^{\mathcal{E}})} \tag{6.6}$$

where $I_C^{\mathcal{E}}$ is the complete instantiation that has the lowest $\mathcal{E}$ function value during the current iteration, $(\langle X_j, w\rangle, \langle X_r, a\rangle)$ is a non conflict pair of assignments and $F(I_C^{\mathcal{E}})$ is the quality of $I_C^{\mathcal{E}}$.

As in Ant Solver, FAS includes *pre-* and *post-* processing steps that use a *min-conflicts* based local search procedure. The pre-processing phase repeatedly performs a local search to collect information that will be used to initialize pheromone trails. The post-processing phase performs a local search after each ant has constructed a complete assignment.

Component	Details
Amount of anti-pheromone deposited	$\Delta anti\tau = \frac{1}{1+\lvert F(L_{worst\_found}) - F(L_{worst\_it})\rvert}$
Translation of anti-pheromone	$\tau_{ij} = \tau_{max} - anti\tau_{ij}$
Resources assigned for the First Step	Percentage B defined by a tuner algorithm

Table 6.1: Summary of components

6.3 Opposite Learning in Focused Ant Solver

This section details the implementation of our Opposite-Learning strategy on Focused Ant Solver. As in the implementation for Ant Knapsack, it is important to previously define some components for implementing our approach: the amount of anti-pheromone, how anti-pheromone will be translated and the percentage B. These components were equally defined as it was shown in Section 5.3 for Ant Knapsack. However, table 6.1 presents a summary of the definition of these components.

About the heuristic knowledge for the *HOL Method*, we considered an existing heuristic knowledge originally proposed in Ant Solver [Sol02], a well-known ant-based algorithm for solving CSPs. In this case, the heuristic knowledge will be defined as:

$$\eta_{I_p}(\langle X_j, w\rangle) = \frac{1}{1 + F(\langle X_j, w\rangle \cup I_p) - F(I_p)} \tag{6.7}$$

Here, the $u\mathcal{D}$-characteristic is related only to detect feasible assignations that are part of poor quality candidate solutions. As the selected heuristic knowledge does not include information about the ϵ function, the H should consider only the available information. For this, the anti-pheromone will be deposited only on feasible arcs of the lowest quality solution obtained in each iteration.

The following section presents the experiments made for testing and evaluating the inclusion of these strategies on Focused Ant Solver.

6.4 Experiments

This section presents the evaluation of our Opposition-Inspired Learning strategy and a comparison with the selected baseline algorithm Focused Ant Solver. We considered a set of 100 binary CSP instances from the transition phase, where the most difficult instances are found [GMP$^+$01]. To define the stopping criterion, we considered the average conflict checks consumed by the pre- and post-processing phases of Focused Ant Solver. For this, the stopping criterion is defined as 4×10^9 conflict checks. The hardware platform adopted for all these experiments was a Power Edge R630

p_2	0.22	0.23	0.24	0.25	0.26	0.27	0.28	0.29	0.30	0.31
κ	0.828	0.871	0.915	0.959	1.003	1.049	1.095	1.141	1.189	1.237

Table 6.2: κ values for the selected p_2 values

server with 2 Intel(R) Xeon(R) CPU E5-2680 v3 @ 2.50GHz, 128 GB of RAM under Ubuntu x64 16.10 distribution.

6.4.1 Instances

Randomly generated binary CSPs were considered for our experiments. We generated hard to solve instances belonging to the phase transition region. For our experiments, 100 instances were generated using Model A. For our experiments we considered $n = 100$ variables, $m = 8$ domain sizes, $p_1 = 0.14$ and p_2 ranging from 0.22 to 0.31. Ten instances were generated for each category of p_2. Table 6.2 shows the values of κ for the different values of p_2. Finally, we have added a random generated solution to each one in order to assure that each instance is solvable.

6.4.2 Parameter Values

Parameter values for Focused Ant Solver and our apporaches were defined using a tuner algorithm called Evolutionary Calibrator (EVOCA) [MR13]. As we already mentioned, Focused Ant Solver performs a pre-processing step to initialize the pheromone matrix. During this step some instances from $p_2 = [0.22, 0.23, 0.29, 0.30, 0.31]$ categories can be solved. In order to correctly evaluate parameter configurations, instances only from $p_2 = [0.25, 0.26, 0.27]$ categories were used during the tuning process. Table 6.3 shows the obtained values by the tuning process for all the approaches, using an average of 8000 evaluations of the tuner algorithm. Again, in our approaches, parameter values of both steps were tuned. About the amount of resources for the First Step, for SOL and WOL methods, a number of $0.01 * 4 * 10^9 = 4 * 10^7$ conflict checks will be assigned. On the other hand, the First Step of the HOL method will consider a number of $0.0728 * 4 * 10^9 = 2.912 * 10^8$ conflict checks.

6.4.3 Results

We have evaluated these algorithms using a total number of 100 solvable CSP instances. Each algorithm was tested using 20 different seeds per instance. To compare these algorithms we defined a success rate (SR) as:

Algorithm	First Step					Second Step				
	m	α	β	ρ	B	m	α	β	ρ	τ_{max}
FAS	-	-	-	-	-	46	15.5000	14.4000	0.0005	26.3931
SOL-FAS	89	4.2995	15.2449	0.0005	0.0100	46	16.8411	14.4000	0.0005	26.3931
WOL-FAS	46	15.5000	14.4000	0.0005	0.0100	46	15.5000	14.4000	0.0005	26.3931
HOL-FAS	64	1.1098	19.7270	0.0005	0.0728	63	17.8627	2.9034	0.0005	25.0454

Table 6.3: Parameter values obtained by EVOCA

Category	FAS	SOL-FAS	WOL-FAS	HOL-FAS
0.22	100.0	100.0	100.0	100.0
0.23	**93.5**	92.5	92.0	88.5
0.24	**73.5**	70.5	71.0	69.5
0.25	65.5	**75.0**	62.5	70.0
0.26	65.0	**69.0**	66.5	68.5
0.27	65.5	84.5	66.5	**86.0**
0.28	99.0	**100.0**	**100.0**	**100.0**
0.29	100.0	100.0	100.0	100.0
0.30	98.0	**98.5**	**98.5**	98.0
0.31	100.0	100.0	100.0	100.0

Table 6.4: Success rates obtained by FAS and OL-FAS approaches in each category

$$SR = 100 * \frac{SuccessRuns}{TotalRuns} \tag{6.8}$$

where $SuccessRuns$ is the number of independent runs where a complete instantiation without conflicts was obtained and $TotalRuns$ is the total number of independent runs executed. Table 6.4 shows the success rates obtained by each algorithm in all the categories. Results in bold show that an algorithm obtains the highest success rate for a category.

Results show that SOL-FAS outperformed FAS in the categories 0.25, 0.26, 0.27, 0.28 and 0.30. Here, improvements ranging from 0.5 to 19 percent were produced. On the other hand, using the HOL method, improvements in categories 0.25, 0.26, 0.27 and 0.28 were obtained, ranging from 1 to 20.5 percent. However, small decrements of the success rate of FAS can be seen in 0.23 and 0.24 categories using the HOL method, ranging from 4 to 5 percent.

About the WOL method, results show that the inclusion of this method produce a decrement of the quality of the solutions found by FAS. However, small improvements can be seen in the category 0.28.

Table 6.5 shows the average execution time in seconds, of each algorithm. In general, the

Category	FAS	SOL-FAS	WOL-FAS	HOL-FAS
0.22	2.881	3.852	5.702	6.5136
0.23	21.188	23.366	37.444	35.407
0.24	36.798	51.966	65.147	84.324
0.25	44.443	82.007	84.467	89.718
0.26	36.364	47.035	67.055	75.743
0.27	35.723	41.835	53.096	38.346
0.28	15.890	14.156	27.423	17.248
0.29	3.189	4.261	5.285	5.512
0.30	4.541	5.012	6.285	6.958
0.31	0.340	0.421	0.511	0.559

Table 6.5: Average execution times (in seconds) of FAS and OL-FAS approaches

Comparison	Negative Ranks	Positive Ranks	Ties	p-value
SOLFAS - FAS	164	100	1736	0.00
WOLFAS - FAS	123	136	1741	0.50
HOLFAS - FAS	149	107	1744	0.01

Table 6.6: Wilcoxon results comparing of FAS with OL-FAS approaches

execution time of our approaches are higher than the algorithm without opposite information. This is mainly produced because the objective of the First Step is not to solve the problem of interest. As a consequence, the solving step will be started when the full percentage of resources, assigned to the First Step, will be consumed.

6.4.4 Statistical Analysis

We performed the pair-wise Wilcoxon non-parametric test to compare the performance of these approaches. Same hypotheses described in Section 5.4.4 are used in these tests. All the independent executions performed by each algorithm are here considered. Table 6.6 shows the results of the tests. Results show that SOL and HOL are statistically different from FAS. On the other hand, no conclusion about the statistical difference between WOL and FAS can be made. As the objective is to minimize the number of conflicts, negative ranks show the number of cases when our approaches outperformed FAS. Results show that FAS obtained better solutions using the information provided by the SOL method, followed by the HOL method. All these computations were done using the statistical software package *PSPP*.

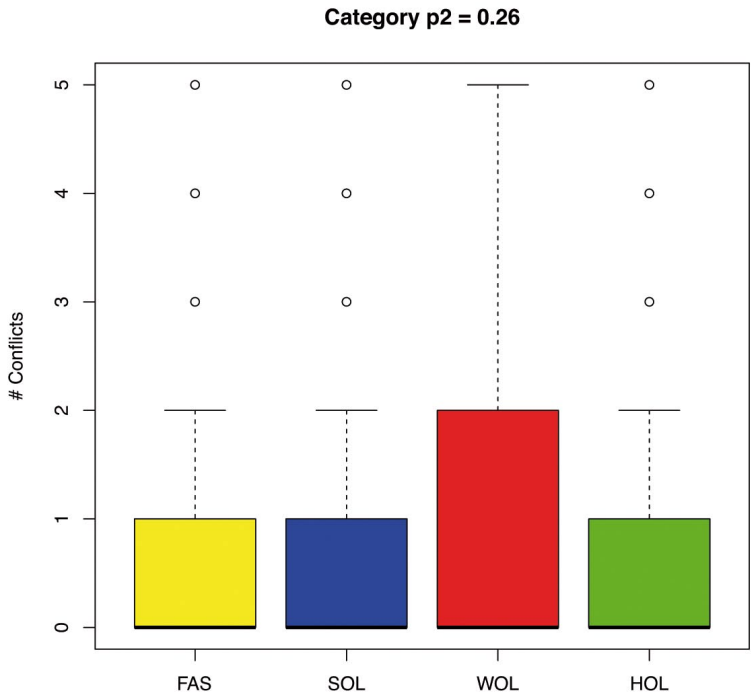


Figure 6.1: Boxplots for instances from the $p2 = 0.26$ category

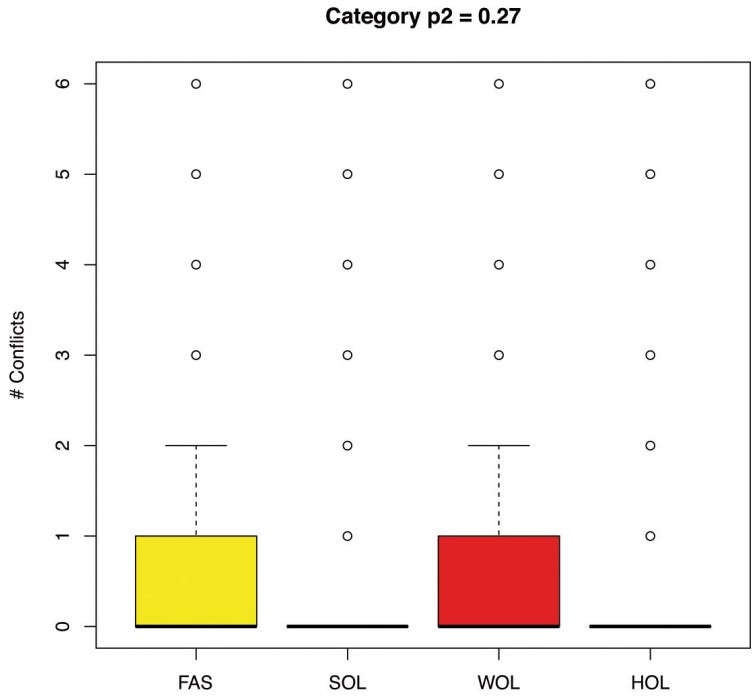


Figure 6.2: Boxplots for instances from the $p2 = 0.27$ category

6.4.5 Boxplots

This section presents boxplots to show the dispersion of the solutions found by each algorithm. Here, all the independent executions of each algorithm are considered. Figure 6.1 shows boxplots for instances from the $p2 = 0.26$ category. First, the medians of the four approaches have 0 conflicts. Moreover, an identical behavior and boxes of FAS, SOL and HOL can be seen. On the other hand, the solutions found by the WOL method are more dispersed with an upper quartile over 2 conflicts and the upper whisker over 5 conflicts.

Figure 6.2 shows boxplots for instances from the $p2 = 0.27$ category. Again, medians of all approaches have 0 conflicts. However, in this case, solutions of SOL-FAS and HOL-FAS are mostly over 0 conflicts. About the boxes of FAS and WOL-FAS, the upper quartiles are over 1 conflict and the upper whiskers are over 2 conflicts.

6.4.6 Discussion

This section presents fitness-distance plots to compare FAS and our approaches in terms of the structure and quality of the obtained solutions. Figure 6.3 shows a fitness-distance plot of an instance from the $p2 = 0.26$ category. As alternative non-conflict solutions were found for this instance, we considered a set of 12 non-conflict solutions (obtained by the four algorithms here evaluated). Thus, for each solution obtained by each algorithm, the lower Hamming distance to a non-conflict solution was considered in these plots. In general, the area covered by the solutions obtained using opposite information is bigger than the obtained by FAS. This shows how the opposite information allows FAS to increase its exploration, in terms of the structure and quality of the found solutions. On the other hand, solutions nearer to a non-conflict solution are reached using opposite information provided by SOL and HOL methods, in terms of the number of conflicts and the Hamming distance. Moreover, non-conflict solutions are shown for the SOL and HOL methods.

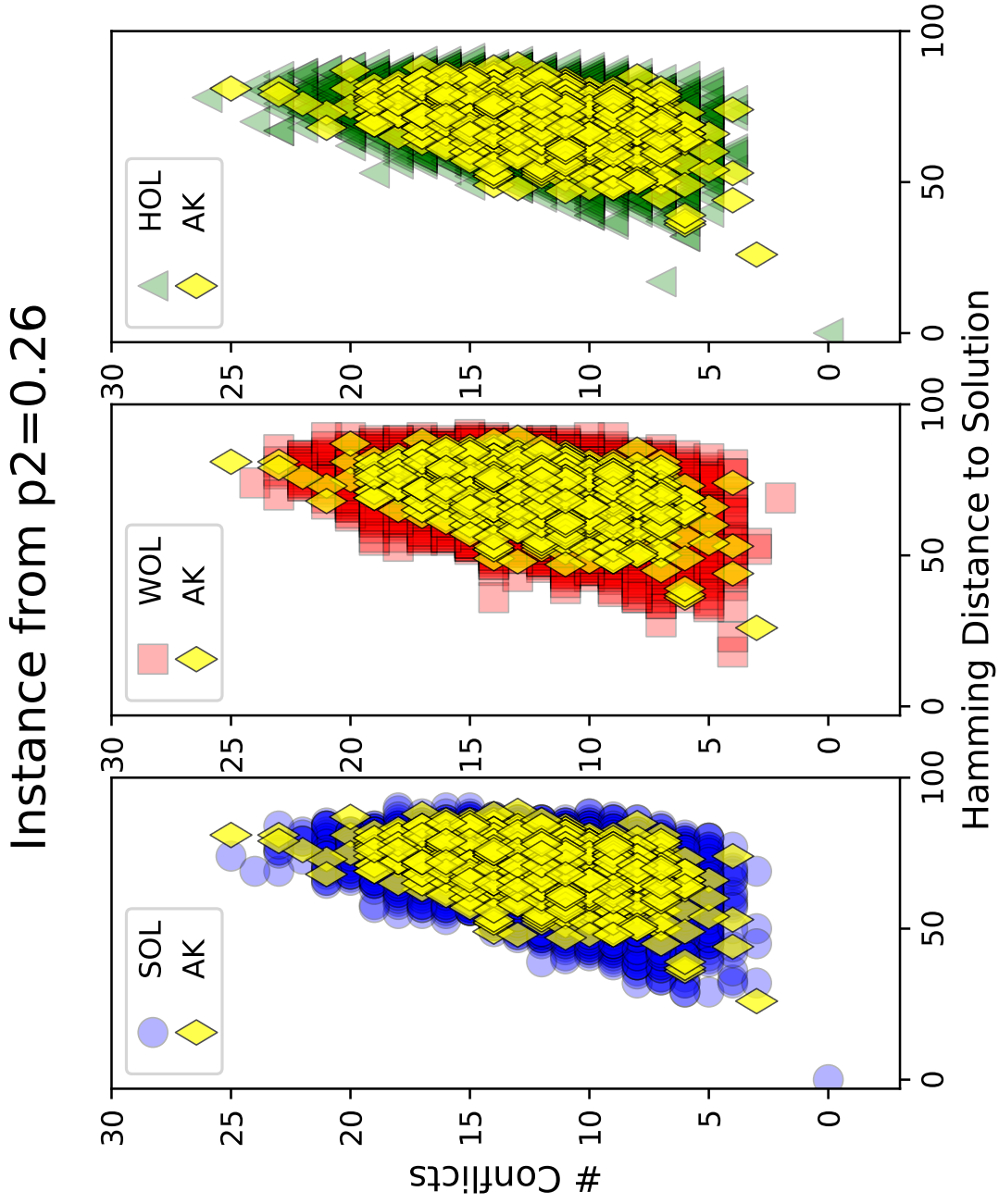


Figure 6.3: Fitness-distance plots for an instance from the $p2 = 0.26$ category

6.5 Conclusions

In this chapter we presented the application of our strategy in Focused Ant Solver, a recently proposed algorithm for solving CSPs and a contribution of this Thesis. Focused Ant Solver is an improved version of Ant Solver that considers heuristics to better discriminate candidate assignments. We previously published an article entitled "Ants Can Learn from the Opposite" [RMRM16a], where results and experiments of the application of our strategy in Ant Solver are presented. In FAS, two heuristics are included to consider information about the feasibility (in terms of minimizing the number of conflicts) and the availability of variable values (related to choosing conflicts "easy" to be repaired).

The three proposed Methods were implemented in Focused Ant Solver: SOL, HOL and WOL. About the HOL method, in this case, we considered an existing heuristic knowledge from literature. Here, it was equally defined as in Ant Solver, including information focused on minimizing the number of conflicts.

To evaluate our approaches, we considered 100 randomly generated instances from the transition phase. Results show that Focused Ant Solver is able to solve problems from all the categories. However, the transition phase can be seen in how the obtained success rate of FAS decreases on the categories where $p_2 \in [0.25, 0.27]$. On the other hand, these success rates were increased with the inclusion of our strategy. Particularly, using the information provided by SOL and HOL methods, an increment between 1% and 20% was obtained. However, the information provided by the WOL method was not useful for FAS. These results were corroborated by the Wilcoxon test and also, the hypothesis test confirms that the SOL method was the most efficient one.

The inclusion of our strategy increases the execution time of Focused Ant Solver. The main reason is related to the objective of the First Step: to learn about an undesired characteristic present in complete instantiations instead of obtaining a non-conflict solution.

In some cases, for Ant Knapsack and Focused Ant Solver, the information provided by the three methods was useful for the target algorithm. This situation motivates a possible collaboration of these methods in order to define an initial pheromone matrix. In the following chapter we present COISA, a collaborative scheme where three sub-colonies cooperate to detect an $u\mathcal{D}$-characteristic.

Chapter 7

A Cooperative Opposition-Inspired Learning Strategy for Ant-based Algorithms

This chapter introduces *COISA*: a Cooperative Opposition-Inspired Learning Strategy for Ant-based algorithms. The obtained results in the implementation of our strategy in Ant Knapsack and Focused Ant Solver show how each method allows the original algorithms to change their intermediate decisions and also, in some cases, reach better quality solutions. However, as each method has a particular objective, the effect in each target algorithm is different. For example, the WOL method was the most useful one for Ant Knapsack, but it worsens the search process of Focused Ant Solver. This situation motivates the design of a cooperative scheme where the information provided by the proposed methods can be considered together. The scheme consists in the cooperation of three sub-colonies of ants to construct an initial pheromone matrix. Each sub-colony will perform a different search process guided by a method. These sub-colonies will obtain information about an $u\mathcal{D}$-characteristic during the *First Step*, that will be considered in the *Second Step* to change its decisions during the construction process. Despite that the key idea of COISA is the same as to our OIL strategy, COISA has some implementation and design differences. Section 7.1 introduces the details of *COISA*.

We select Ant Knapsack to evaluate COISA. Details of the inclusion of COISA in Ant Knapsack are presented in Section 7.3. We considered benchmark instances to evaluate and compare COISA

and each method independently included. Preliminary results are presented in Section 7.4.

As the collaboration of these three sub-colonies can be time consuming, we decided to execute the *First Step* in parallel and the *Second Step* is executed sequentially. The first ideas of the inclusion of parallelism in ant-based algorithms were proposed in 1993, with the objective of improving the quality and the performance of the search process [Dor93]. In general, most common metrics used by the research community to evaluate the performance of parallel approaches are the *Speedup* and *efficiency*. *Speedup* (S_m) evaluates how much faster a parallel approach is than its sequential algorithm and is calculated as:

$$S_m = \frac{T_1}{T_m} \tag{7.1}$$

where T_1 is the execution time of the sequential algorithm and, T_m the execution time of the parallel version using m processors. As Amdahl's law [Amd67] proposes, the sequential part of the algorithm will limit the performance of any parallel approach, depending of how parallelism has been incorporated. Section 7.2 presents the details of the parallelism. In order to compare COISA with the existing parallel ant-based approaches, Section 7.5 presents a brief literature review.

7.1 Cooperation between Methods

This section presents our approach named Cooperative Opposition - Inspired Strategy for Ants (*COISA*). COISA also considers a division of the search process: the *First Step* for learning about an $u\mathcal{D}$-characteristic and a *Second Step* for solving the problem of interest. During the *First Step*, three sub-colonies of ants will cooperate in the construction of a pheromone matrix. Each sub-colony will be focused on obtaining information about an $u\mathcal{D}$-characteristic and will be guided by one *Method*: *SOL*, *HOL* or *WOL*. A defined number of ants will participate in each sub-colony:N_{SOL}, N_{HOL} and N_{WOL}. These methods are equally defined as it was explained in Section 4.3.

The main differences of COISA with our proposed approach are:

- In order to avoid the translation of anti-pheromone into pheromone, we consider to decrease anti-pheromone during the *First Step*. The initial pheromone matrix will be directly provided from the *First* to the *Second Step*.

- As at least one ant from each sub-colony participates in the construction of the pheromone matrix, pheromone evaporation is not considered during *First Step*.

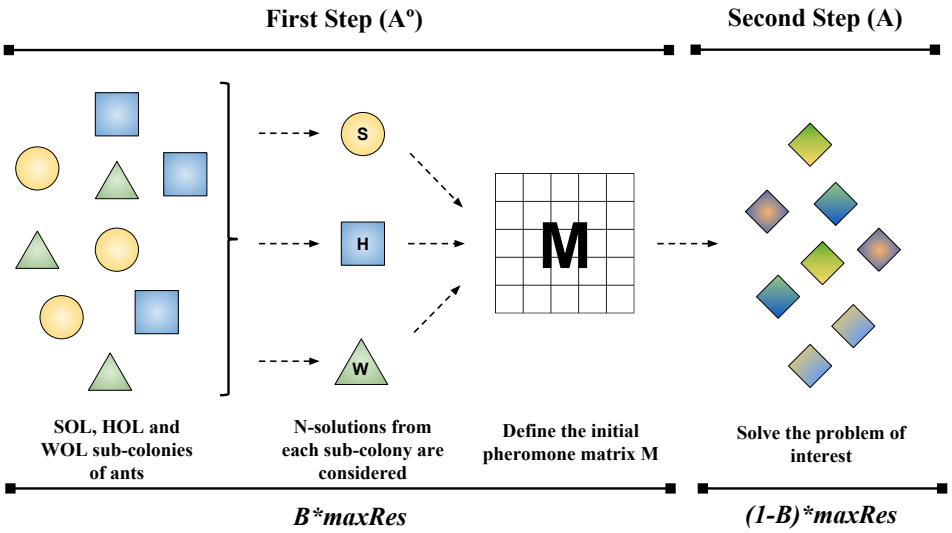


Figure 7.1: Interaction between sub-colonies in COISA

Figure 7.1 shows how *COISA* works. Here, all ants consider the same *anti-pheromone* matrix M to construct solutions. At the end of each iteration, *anti-pheromone* will be updated by ant^{SOL}, ant^{HOL} and ant^{WOL} considering the following rule:

$$anti\tau_{ij}^{new} = anti\tau_{ij}^{old} - \Delta_{ij}^{SOL} - \Delta_{ij}^{HOL} - \Delta_{ij}^{WOL} \tag{7.2}$$

where Δ_{ij}^{SOL}, Δ_{ij}^{HOL} and Δ_{ij}^{WOL} are the amounts of anti-pheromone to be decreased.[1] When the *First Step* ends, M will be the initial pheromone matrix for A. The idea is to decrease the initial pheromone value of components that were part of complete instantiations related to an $u\mathcal{D}$-characteristic.

The execution of the *First Step*, as three sub-colonies are considered, can be time consuming. For this, we consider to perform the *First Step* in parallel and the Second Step is executed sequentially. Next section presents details of this implementation.

7.2 Parallelism in COISA

COISA was implemented in *POSIX Threads*. Figure 7.2 explains the implementation of *COISA*. Two types of threads will be considered: *constructor* or *manager* threads. Considering a total

[1] The number of SOL, HOL or WOL ants that will participate in the anti-pheromone updating should be defined considering how it is defined in A for managing pheromone.

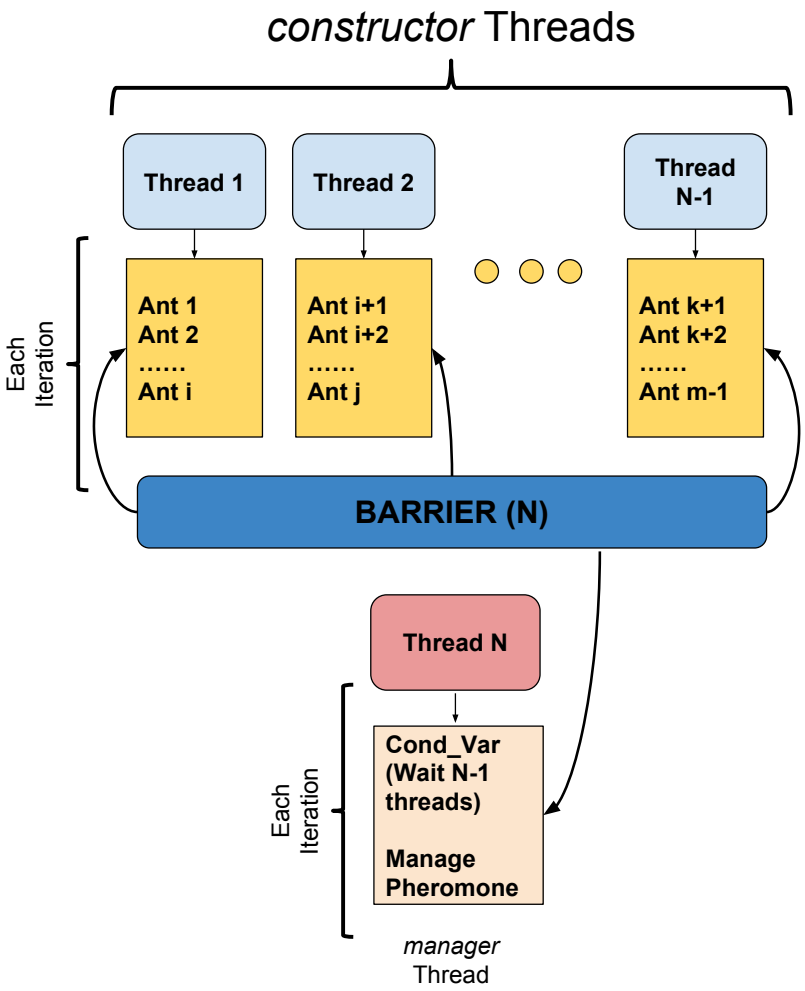


Figure 7.2: Parallel implementation of our proposed COISA

of N threads and m ants, $(m-1)$ ants are assigned to $(N-1)$ *constructor* threads, focused in constructing and evaluating solutions. On the other hand, one ant is assigned to one *manager* thread that will be focused on the pheromone management.

Algorithm 3 shows how *constructor* threads work. The tasks of each *constructor* thread are: (1) construct the candidate solutions of their assigned ants, (2) evaluates each solution and, (3) determines which ant will alter the anti-pheromone matrix (line 3). Ants from different sub-colonies can be assigned to constructor threads. A shared variable *ConstructionDone* is used to count the number of threads that have finished their tasks (line 6).

Algorithm 4 shows how the *manager* thread works. The *manager* thread waits until all constructor threads finish all their tasks to update the pheromone matrix M (line 11). When the

Algorithm 3 Constructor thread

```
1: while signal_cont do
2:    for Each ant k assigned do
3:       Ant[k].ConstructCompleteInstantiation();
4:    end for
5:    pth_mutex_lock(&mX);
6:    ConstructionDone++;
7:    pth_mutex_unlock(&mX);
8:    pth_cond_signal(&cond,&mX);
9:    pth_barrier_wait(&bar);
10: end while
```

Algorithm 4 Manager thread

```
1: while signal_cont do
2:    while !(Done) do
3:       pth_mutex_lock(&mX);
4:       if #(ConstructionDone) then
5:          pth_cond_wait(&cond);
6:       else
7:          Done = 1;
8:       end if
9:       pth_mutex_unlock(&mX);
10:   end while
11:   SubtractPheromone(M);
12:   if (B * maxRes) consumed then
13:      signal_cont = false;
14:   end if
15:   pth_barrier_wait(&bar);
16: end while
17: return Pheromone matrix M
```

resources assigned to the *First Step* are consumed, the Second Step starts executing the original algorithm A with the pheromone matrix M (line 12).

For the synchronization of all the threads, a barrier (*bar*), a conditional variable (*cond*) and a mutex (mX) are used. The barrier allows the constructor threads to wait until the *manager* thread updates the pheromone matrix M.

A special treatment of the seed should be implemented. To make the parallel execution of COISA reproducible, for a given started seed, a local random number generator for each thread was implemented. Moreover, to ensure the reproducibility, a thread always processes the same subset of ants in the same order. A barrier allows the synchronization of each iteration of the algorithm. This makes COISA stochastic and parallel but deterministic when using the same initial seed and parameters. In the following section, we present the study case to evaluate *COISA* in Ant Knapsack.

7.3 COISA for Ant Knapsack

This section presents some details that should be considered before the implementation of *COISA* in Ant Knapsack. First, it is important to define the following details:

- The amount of *anti-pheromone* $\Delta anti\tau$ is defined similarly as in Ant Knapsack:

$$\Delta anti\tau = \frac{1}{1 + F(L_{worst\_found}) - F(L_{worst\_it})} \tag{7.3}$$

  where L_{worst_it} is the worst solution found in the current iteration and L_{worst_found} is the worst solution found in during the execution.

- In Ant Knapsack, ants that constructed the best quality solution of each iteration are allowed to deposit pheromone. In most cases, one ant is allowed to deposit pheromone. In order to maintain this behavior, during the *First Step*, one ant per sub-colony will be allowed to decrease anti-pheromone.

- As Ant Knapsack is a $\mathcal{MMAS}$ algorithm, the initial anti-pheromone value for the *First Step* is τ_{max}.

- The value of the percentage of resources B will be obtained using a tuner algorithm.

About the HOL method, it is necessary to define a heuristic knowledge for guiding its search process. In this case, we consider the same η used in [RMRM16b]:

$$\eta_{I_P^k}(o_j) = \frac{p_j}{\sum_{t=1}^{T} RC_t} \tag{7.4}$$

where RC_t is the *remaining capacity* in the dimension t defined as $RC_t = b_t - w_{jt}$.

The objective is to give priority to objects with higher profit and higher weight in each dimension. The idea is to identify the *core* of objects for which it is hard to decide if they will be part of an optimal solution or not [PRP06].

In the following section, we present the experimental evaluation of this strategy considering all these methods.

Approach	N_{SOL}	N_{HOL}	N_{WOL}	B
COISA-AK	16	11	8	0.211
SOL-AK	20	-	-	0.241
HOL-AK	-	2	-	0.422
WOL-AK	-	-	16	0.408

Table 7.1: Parameter values obtained by EVOCA

7.4 Experiments

In this section we present preliminary experiments performed in order to evaluate COISA. The main objective of these experiments is to compare the quality of solutions found by Ant Knapsack versus the quality found when COISA strategy is incorporated to AK. We considered two sets from the OR Library benchmarks instances proposed by Chu and Beasley. Each set contains 30 instances and have different number of objects and dimensions: 10×100 (10 dimensions and 100 objects) and 5×100 (5 dimensions and 100 objects). We defined the same experimental setup as [ASG04], considering 50 independent runs per instance, each with 60000 evaluations. In order to evaluate and compare the collaboration between the three sub-colonies, we present results obtained by each method independently: SOL-AK, HOL-AK and WOL-AK. For all the executions, we considered a number of ten threads for COISA and the independently evaluated methods. The hardware platform adopted for all these experiments was a Power Edge R630 server with 2 Intel(R) Xeon(R) CPU E5-2680 v3 @ 2.50GHz, 128 GB of RAM under Ubuntu x64 16.10 distribution.

7.4.1 Parameter Tuning

As we mentioned earlier, the idea is to consider Ant Knapsack as a black box. For this, we considered the same parameter values proposed in [ASG04]: $\alpha = 1$, $\beta = 5$, $\rho = 0.01$, $N_{Total} = 30$, $\tau_{max} = 6$ and $\tau_{min} = 0.01$. To determine the parameter values for our approaches we used a parameter tuner algorithm named Evolutionary Calibrator (EVOCA) [MR13]. The objective was to obtain the number of ants for each sub-colony and the percentage B. Notice that in the independent approaches only ants from their belonging methods are considered (and the number of ants from the other methods are set in zero). Table 7.1 shows the values obtained by EVOCA.

7.4.2 Preliminary Results

Table 7.2 shows the results obtained for the 10×100 set and Table 7.3 shows the results for the 5×100 set. We considered 50 independent runs per instance, each with 60000 evaluations

($maxRes$). Light grey cells represent when an algorithm outperforms other approach considering the average quality (AVG) of the 50 seeds adopted. On the other hand, dark grey cells represent when an algorithm outperforms another approach considering the Best quality solution obtained. Also, the standard deviation (SDV) is shown for each instance and algorithm. First, results show that AK could find most of the best known solutions for the instances from both sets (51 of the 60 instances). Moreover, *COISA-AK* outperformed AK obtaining the best known solution in 53 of the 60 instances. This shows that the collaboration between sub-colonies is better than each method on their own. Regarding the average quality, results show that AK obtained better results in the 5×100 set and *COISA-AK* was better for the 10×100 set. Finally, considering the independent and the cooperative approaches, all the best known solutions can be found using opposite information. The non-parametric Wilcoxon test was applied to assess that these algorithms are statistically different ($pvalue = 0.01$).

About the Speedup obtained by *COISA-AK*, the average was 1.8, with a maximum of 4.9 and a minimum of 1.4. Considering that the *First Step* only consumes the 20% of the evaluations, these metrics shows the positive effect of using a parallel architecture for executing the *First Step*.

#	BK	AK			COISA-AK			SOL-AK			HOL-AK			WOL-AK		
		AVG	SDV	BEST	AVG	SDV	BEST	AVG	SDV	BEST	AVG	SDV	BEST	AVG	SDV	BEST
1	23064	**23016.0**	42.2	23064	23014.3	46.4	23064	22998.6	47.5	23057	23008.0	41.0	23064	23006.7	42.7	23064
2	22801	**22714.0**	67.2	22801	22702.2	83.8	22801	22713.8	66.5	22801	22694.6	58.3	22801	22693.6	69.4	22801
3	22131	22034.0	66.9	22131	**22046.6**	56.0	22131	22024.4	69.7	22131	22008.2	69.9	22131	22035.5	65.3	22131
4	22772	**22634.0**	60.6	22717	22613.3	63.9	22763	22623.4	64.0	22772	22598.2	73.7	22772	22601.7	53.8	22709
5	22751	22547.0	66.3	22654	**22559.2**	47.6	22654	22543.2	70.8	22697	22533.0	66.9	22697	22542.7	51.4	22697
6	22777	22602.0	63.3	22716	22593.4	46.8	22716	**22610.3**	51.4	22716	22594.7	46.0	22664	22591.9	40.5	22675
7	21875	21777.0	44.9	21875	**21790.8**	36.7	21875	21773.4	45.5	21875	21780.1	48.6	21875	21774.3	54.2	21875
8	22635	22453.0	89.2	22551	22498.8	54.1	22635	**22512.0**	40.6	22551	22511.7	57.8	22635	22500.1	57.5	22635
9	22511	22351.0	69.4	22511	**22379.6**	47.0	22511	22369.7	40.3	22438	22362.4	51.6	22511	22352.2	62.5	22511
10	22702	22591.0	88.5	22702	**22616.0**	102.9	22702	22600.1	99.8	22702	22576.5	91.0	22702	22572.9	88.8	22702
1	41395	41329.0	48.5	41395	41324.1	47.4	41395	**41329.1**	49.8	41395	41312.4	51.8	41393	41309.0	48.7	41395
2	42344	42214.0	49.5	42344	**42233.5**	47.0	42344	42232.2	60.4	42344	42210.2	45.5	42344	42221.4	54.9	42344
3	42401	42300.0	58.1	42401	42309.0	38.4	42401	42311.5	41.7	42401	**42316.1**	47.2	42401	42313.6	43.5	42401
4	45624	45461.0	73.6	45624	**45484.2**	69.4	45624	45450.2	70.9	45585	45462.3	71.6	45585	45474.4	64.2	45598
5	41884	41739.0	57.3	41884	**41770.0**	53.0	41884	41769.9	52.0	41884	41758.8	53.0	41884	41750.4	50.2	41884
6	42995	42909.0	76.3	42995	42910.6	76.5	42995	42898.8	72.7	42995	42891.3	78.1	42995	**42923.4**	69.8	42995
7	43574	43464.0	71.7	43553	43466.9	50.0	43553	43470.0	43.0	43553	**43479.0**	47.6	43553	43463.7	46.5	43552
8	42970	42903.0	47.7	42970	42904.7	39.4	42970	42901.5	48.1	42970	**42924.6**	35.3	42970	42915.2	40.1	42970
9	42212	42146.0	48.0	42212	**42167.3**	39.8	42212	42165.7	39.7	42212	42160.6	38.4	42212	42162.5	42.2	42212
10	41207	41067.0	89.7	41207	**41098.7**	36.9	41207	41085.9	39.0	41134	41093.5	38.3	41207	41077.7	44.8	41207
1	57375	57318.0	59.5	57375	57295.9	66.1	57375	57307.8	68.1	57375	57311.7	74.1	57375	**57321.9**	61.3	57375
2	58978	58889.0	40.2	58978	**58914.2**	32.4	58978	58899.4	54.6	58978	58886.4	43.3	58934	58898.1	24.1	58978
3	58391	58333.0	29.5	58391	**58337.7**	26.3	58391	58321.2	47.8	58391	58326.8	32.5	58391	58335.4	27.2	58391
4	61966	61885.0	42.4	61966	**61891.2**	36.4	61966	61876.0	47.9	61966	61873.9	40.6	61966	61882.7	36.6	61966
5	60803	60798.0	5.0	60803	**60800.6**	3.0	60803	60799.9	3.2	60803	60800.5	3.1	60803	60800.0	5.2	60803
6	61437	61293.0	52.7	61437	61295.3	55.6	61437	61294.1	52.6	61437	61288.3	48.6	61437	**61297.5**	52.5	61437
7	56377	56324.0	35.7	56377	56319.0	35.4	56377	56311.0	47.0	56377	56313.9	49.1	56377	**56328.3**	33.8	56377
8	59391	59339.0	53.3	59391	59340.7	42.6	59391	59331.5	51.4	59391	59331.2	53.0	59391	**59341.3**	37.5	59391
9	60205	60146.0	62.6	60205	**60167.7**	50.7	60205	60123.1	73.5	60205	60096.8	70.9	60205	60155.8	56.9	60205
10	60633	60605.0	36.1	60633	**60613.9**	32.2	60633	60589.4	47.6	60633	60571.7	48.4	60633	60613.5	30.7	60633

Table 7.2: Results for Set 10×100 from OR Library

#	BK	AK			COISA-AK			SOL-AK			HOL-AK			WOL-AK		
		AVG	SDV	BEST	AVG	SDV	BEST	AVG	SDV	BEST	AVG	SDV	BEST	AVG	SDV	BEST
1	24381	**24342.0**	29.3	24381	24340.6	29.0	24381	24329.4	38.2	24381	24335.1	35.3	24381	24330.4	31.4	24381
2	24274	**24247.0**	38.5	24274	24241.2	35.1	24274	24234.9	42.8	24274	24246.0	33.5	24274	24229.2	38.8	24274
3	23551	**23529.0**	8.0	23551	23527.2	9.2	23551	23526.3	13.6	23551	23525.6	14.1	23551	23527.1	11.9	23551
4	23534	**23462.0**	32.6	23534	23460.1	32.7	23527	23453.3	44.4	23534	23458.4	34.8	23527	23457.8	30.2	23511
5	23991	23946.0	31.8	23991	23942.5	26.8	23991	23934.2	33.8	23991	23940.1	35.0	23991	**23950.5**	29.0	23991
6	24613	**24587.0**	31.3	24613	24585.3	25.8	24613	24583.0	28.9	24613	24573.3	34.2	24613	24579.2	28.8	24613
7	25591	25512.0	43.8	25591	25521.8	41.3	25591	**25524.2**	47.8	25591	25506.2	39.7	25591	25509.5	45.9	25591
8	23410	23371.0	30.3	23410	23375.1	33.5	23410	23378.2	29.6	23410	23378.8	29.4	23410	**23381.5**	31.9	23410
9	24216	24172.0	32.9	24216	**24177.0**	31.1	24216	24171.7	32.7	24216	24163.1	38.0	24216	24164.5	39.3	24216
10	24411	**24356.0**	44.3	24411	24346.1	45.8	24411	24340.5	44.5	24411	24342.9	47.2	24411	24346.7	44.8	24411
1	42757	42704.0	14.3	42757	42706.5	25.4	42757	**42709.8**	21.0	42757	42700.6	11.4	42757	42701.6	14.8	42757
2	42545	42456.0	15.8	42510	42458.4	14.6	42471	**42459.5**	12.9	42494	42455.0	21.3	42545	42458.8	25.4	42545
3	41968	41934.0	22.3	41967	**41939.8**	15.9	41968	41935.2	23.0	41967	41930.9	26.3	41967	41930.8	27.7	41967
4	45090	45056.0	24.0	45071	45056.1	24.1	45071	**45058.3**	23.1	45071	45041.0	29.4	45071	45049.4	31.7	45071
5	42218	42194.0	33.2	42218	42201.9	31.4	42218	42196.0	31.2	42218	42189.6	41.8	42218	**42202.2**	28.1	42218
6	42927	42911.0	33.3	42927	**42913.5**	32.6	42927	42903.5	40.8	42927	42913.0	34.0	42927	42908.0	39.5	42927
7	42009	41977.0	45.2	42009	**41985.2**	40.9	42009	41978.9	42.9	42009	41984.6	40.6	42009	41978.0	49.0	42009
8	45020	44971.0	32.5	45010	**44988.8**	22.1	45020	44984.4	29.2	45020	44969.9	35.5	45010	44979.7	31.9	45010
9	43441	**43356.0**	38.5	43441	43349.1	42.9	43441	43353.6	49.7	43441	43345.1	40.9	43441	43347.1	40.7	43441
10	44554	44506.0	25.2	44554	44512.8	23.5	44554	44513.9	25.5	44554	44510.4	28.2	44554	**44515.8**	25.2	44554
1	59822	59821.0	3.2	59822	**59822.0**	0.0	59822	59815.1	20.2	59822	**59822.0**	0.0	59822	**59822.0**	0.0	59822
2	62081	62010.0	47.1	62081	62010.7	44.1	62081	62003.6	44.8	62081	61994.7	31.0	62081	**62011.0**	49.0	62081
3	59802	59759.0	21.7	59802	59757.9	16.1	59802	59745.8	24.7	59802	59750.2	20.2	59802	**59760.1**	22.2	59802
4	60479	60428.0	21.8	60479	**60444.8**	27.3	60479	60417.2	30.0	60479	60438.6	24.2	60479	60435.8	23.6	60479
5	61091	61072.0	20.0	61091	61075.5	18.7	61091	61066.8	35.9	61091	**61078.4**	21.2	61091	61077.6	18.2	61091
6	58959	**58945.0**	14.5	58959	58940.6	12.3	58959	58929.4	35.5	58959	58943.9	19.4	58959	58943.3	14.1	58959
7	61538	**61514.0**	24.0	61538	61511.7	26.6	61538	61508.9	27.2	61538	61498.9	36.9	61538	61513.6	25.9	61538
8	61520	61492.0	25.6	61520	61494.0	22.7	61520	61473.7	34.7	61520	61475.4	32.7	61520	**61496.5**	23.0	61520
9	59453	**59436.0**	40.5	59453	59434.8	43.1	59453	59413.9	59.1	59453	59427.4	53.4	59453	59435.2	39.7	59453
10	59965	59958.0	8.4	59965	59956.0	11.2	59965	59944.9	26.9	59965	59946.0	25.8	59960	**59959.1**	5.2	59965

Table 7.3: Results for Set 5×100 from OR Library

7.5 Discussion

Several parallel ant-based approaches have been proposed in literature [PNC11]. This section presents a brief review of these techniques.

Parallel ant-based approaches were classified based in how parallelism is included [RL02]: (1) *Parallel Independent Ant Colonies*, based on the well-known *master/slave* approach [Fos95], executing a number of sequential ACO searches in available processors; (2) *Parallel Interacting Ant Colonies*, including exchange of information (in general, pheromone structures) between running colonies; (3) *Parallel Ants*, each ant builds and evaluates its own candidate solution in the assigned thread and a master process is focused on the pheromone management; (4) *Parallel Evaluation of Solution Elements*, obtaining the quality of candidate solutions in parallel; (5) a combination of ants and evaluation of solutions. Also, a Parallel Ants approach is proposed for the Traveling Salesman Problem. Results in eight TSPLIB instances shows a *Speedup* between 0.06 to 3 was obtained using between 2 to 8 processors.

A more complete classification of parallel ant-based algorithms was proposed in [PNC11], considering five models:

- *Master/slave (M/S)* model, considers a master process that manages global information and controls slave processes that perform a defined amount of work. This determines the granularity, coarse, medium or fine-grain, according the tasks assigned to each processor.
- *Cellular* model, defining each colony as a small neighborhood with their own pheromone matrix. The high-quality solutions of a neighborhood can affect other neighborhoods using a diffusion model.
- *Parallel independent runs* model, similar to *Parallel Independent Ant Colonies*.
- *Multicolony* model, similar but more general than *Parallel Interacting Ant Colonies*.
- *Hybrid* model, that includes features from more than one model.

More recently, approaches are implemented for GPU/CUDA platforms, i.e, an ant-based algorithm for the MKP was proposed in [FCMS14]. This approach considers multiple colonies, where each thread block made the computations of one ant.

Three GPU-based Ant Colony System (ACS) approaches are proposed in [Ski16] for solving the TSP. Approaches differ in how pheromone is updated, considering global or local pheromone updating rules. Here, best-so-far solution can be marked with an additional amount of pheromone.

Also, a selective pheromone memory is implemented, considering only a lower number of "important" edges. Results in eight TSPLIB instances, with sizes ranging from 198 to 2392 cities, showed that parallel approaches are more efficient than a sequential ACS algorithm.

COISA considers two types of threads: constructor and manager threads. When the tasks of constructor threads end, the manager thread updates the pheromone matrix. Only one pheromone matrix is considered, by all the sub-colonies, to construct solutions during the *First Step*. Ants from different sub-colonies can be assigned to a constructor thread. These features show that COISA can be classified as an hybrid approach. First, the pheromone is managed similarly as in the *Master/slave* model. Moreover, the search process is very similar as it is presented in the *Parallel Ants* model. However, as only one pheromone matrix is considered, some features from the *Parallel Interacting Ant Colonies* model are present in COISA.

7.6 Conclusions

This chapter presents a Cooperative Opposition-Inspired Strategy for Ants named *COISA*, for focusing the search process of ant-based algorithms. As our original strategy, the objective is to obtain information about such $u\mathcal{D}$-characteristic that could bias the search process to poor quality solutions. Here, the search process is also divided into two steps and during the *First Step*, three sub-colonies of ants cooperate to define an initial pheromone matrix. Each sub-colony is guided by one *Method*: *SOL*, *HOL* and *WOL*.

To evaluate our strategy, we used the well-known Ant Knapsack algorithm for solving the MKP. The obtained initial pheromone matrix will contain information about pairs of objects that were related to a $u\mathcal{D}$-characteristic. Results showed that the inclusion of *COISA* in Ant Knapsack improves its search process, benefits its robustness and helps out to obtain better quality solutions. Moreover, results showed that the cooperation between these methods is better than each one by itself.

An article entitled "A Cooperative Opposite-Inspired Learning Strategy for Ant-based Algorithms" [RMRC18] was published with the contents of this chapter. It is important to remark that this chapter presents a preliminary design and evaluation of COISA, that will be study in a future work.

Chapter 8

Conclusions

This thesis proposes a general Opposition-Inspired Learning strategy for ant-based algorithms, to improve its search process in terms of the quality of the obtained solutions. Our strategy is focused on learning about an undesirable ($u\mathcal{D}$) characteristic that can bias some *intermediate* decisions of the construction process of the target algorithm. These *intermediate* decisions can give a higher priority to locally interesting components and as a consequence, inhibit reaching good quality solutions. The objective of this strategy is to provide valuable information about such an $u\mathcal{D}$-characteristic to an ant-based algorithm, in order to change some of these intermediate decisions. The idea is to give a chance to components that were originally discarded or not considered because of the preference to most (locally) interesting components. It is important to remark that this characteristic is not inherent to the problem of interest, but it cannot be perceptible by the current pheromone information and by the heuristic knowledge, as it is specifically defined in the target algorithm.

We propose to divide the search process into two steps: a *First Step* to learn about such an $u\mathcal{D}$-characteristic and a *Second Step*, to solve the problem of interest. During the *First Step*, complete instantiations related to this $u\mathcal{D}$-characteristic will be marked with anti-pheromone. The intention is to produce a repel effect in components that were part of these complete instantiations. For this, the *First Step* is performed by a modified version of the target algorithm. Three *Methods* were proposed with the objective of evaluate different possibilities to identify an $u\mathcal{D}$-characteristic: Soft Opposite Learning (SOL), Worst Opposite Learning (WOL) and Half Opposite Learning (HOL). Each one performs a different search process considering a different: anti-pheromone management, heuristic knowledge definition and function to be optimized. At the end of the *First Step*,

an initial pheromone matrix will be provided to the *Second Step* for solving the problem of interest.

This work was inspired in the Opposite Learning literature, in both of its types: Opposition-Based Learning (OBL) and Opposition-Inspired Learning (OIL). OBL was proposed with the objective of obtaining complementary candidate solutions, in order to continue the search process with the higher quality solutions. Usually, these complementary candidates are computed using a mapping function. OBL was mainly applied in metaheuristics for solving continuous problems and with the objective of initializing the target algorithm, increasing its exploration, accelerating its convergence, avoiding premature convergence, local improvement, among others. On the other hand, OIL was proposed for specific situations when the desired concept of opposition cannot be directly related to mapping solutions. The strategy proposed in this thesis is an OIL approach and defines the concept of opposite in terms of possible decisions made by ant-based algorithms, that could lead the search process towards poor quality candidate solutions. Basically, the information obtained during this learning step will allow us to consider new and different decisions during the construction process. A paper entitled "A survey and classification of Opposition-Based Metaheuristics" was published with the contents presented in this chapter [RRM17].

Our strategy was proposed for ant-based algorithms that were designed for solving combinatorial problems. In order to correctly implement this strategy, the target algorithm should consider pheromone information and heuristic knowledge in its construction process. The absence of one or both components will probably require to redefine some aspects of the strategy. However, the strategy can be adapted and redefined considering: (1) the type of ant-based algorithm (Ant System, Ant Colony System, $\mathcal{MAX}-\mathcal{MIN}$ Ant System, among others), (2) features of the target algorithm that should be replied in the *First Step* (the use of a local search procedure, special pheromone management rules, scheduling of the construction process, among others).

We evaluated our strategy in two ant-based algorithms: Ant Knapsack (AK), designed for solving the Multidimensional Knapsack Problem (MKP) and Focused Ant Solver (FAS), proposed for solving Constraint Satisfaction Problems (CSP).

AK is a $\mathcal{MAX}-\mathcal{MIN}$ Ant Solver algorithm that is able to solve benchmark instances of the MKP. However, its search process can be improved when the size of instances increase. For this, a set of 90 large-size benchmark instances was considered, with 500 objects and 5, 10 or 30 dimensions. Results show how the opposite information allows AK to reach better quality solu-

tions. Boxplots, statistical analysis and fitness-distance plots confirm that the use of our strategy allows AK to reach different complete instantiations, in terms of quality and structure. The information provided by the SOL method to AK was the most valuable, reaching the lowest average percentage gap followed by WOL-AK and HOL-AK. Moreover, using the SOL method, AK is now closer to state-of-the-art approaches, including the best metaheuristic approach in solving MKP instances with 500 objects. Two articles entitled "Using Anti-pheromone to Identify Core Objects for Multidimensional Knapsack Problems: A Two-step Ants Based Approach" [RMRM15] and "Learning from the opposite: Strategies for Ants that solve Multidimensional Knapsack Problem" [RMRM16b] were published presenting preliminary results and experiments of the application of our strategy in Ant Knapsack.

FAS is a $\mathcal{MAX} - \mathcal{MIN}$ Ant Solver algorithm proposed as an improved version of the well-known Ant Solver algorithm. Binary CSPs instances, belonging from the transition phase, were considered to evaluate the inclusion of our strategy in FAS. Results show that the SOL and HOL methods provide useful information for FAS. On the other hand, the search process of FAS was worsen using the information provided by the WOL method. As in AK, boxplots, statistical analysis and fitness-distance plot confirms that the use of our strategy allows FAS to reach different complete instantiations, in terms of quality and structure. Moreover, the use of our strategy allows FAS to reach alternative non-conflict solutions to these transition phase problems. An article entitled "Ants Can Learn from the Opposite" [RMRM16a] was published presenting results and experiments of the application of our strategy in Ant Solver.

The results of the evaluation in both algorithms confirms our hypothesis that is: if we consume a certain amount of resources in identifying and learning about some $u\mathcal{D}$-characteristic, the search process could be further focused making decisions using this knowledge so that we can obtain complete instantiations of better quality. However, the use of our strategy increases the execution time in both target algorithms. This is mainly produced because the objective of the *First Step* is to obtain information about an $u\mathcal{D}$-characteristic instead of solving the problem of interest. For this, in the three methods, the function that is being optimized is the opposite of the original function ($\bar{F}$).

Comparing the implementation in both algorithms, about the $u\mathcal{D}$-characteristics, a quality-related method was the most efficient one (SOL method). This method can be fully implemented without considering new or different components from the original algorithm. This shows how

generic can be the strategy here proposed. However, to implement Half Opposite-Learning it is necessary to define the heuristic knowledge for the *First Step*. For this, we present three recommendations: (1) use an existing heuristic knowledge from an existing ant-based algorithm, (2) define it based in an existing heuristic from literature and (3) propose a new function. The idea of the first two recommendations is to maintain how general this strategy can be and also, reduce the effort to the designer. In this thesis, for Ant Knapsack, the heuristic knowledge was defined based on an existing heuristic from literature and for Focused Ant Solver, an existing heuristic knowledge from literature was used.

In some cases, for the two target algorithms, the information provided by the three proposed methods was useful to improve their search processes. In order to evaluate the collaboration between the three proposed methods, a cooperative scheme named COISA was proposed. Here, sub-colonies of ants cooperate in the definition of an initial pheromone matrix that contains information about an $u\mathcal{D}$-characteristic. Each sub-colony is guided by one of the proposed method. COISA was implemented in a parallel architecture, considering constructor and manager threads. Ants are assigned to constructor threads that are focused in constructing and evaluating solutions. One manager ant is allowed to manage the pheromone information. To evaluate COISA, we selected Ant Knapsack as the baseline algorithm. Preliminary results in benchmark instances show that the information provided by COISA improves its search process, in terms of the quality of the solutions and its robustness. An article entitled "A Cooperative Opposite-Inspired Learning Strategy for Ant-based Algorithms" [RMRC18] was published with the contents of this chapter.

For future work, we are interested in the following areas:

- Evaluate the inclusion of our strategy in a different ant-based algorithm proposed for a different combinatorial problem (different of the already used),
- Perform a parameter analysis in order to understand the effect of each component of the *First Step*,
- Analyze the application of our proposed strategy in different stages of the search process of a target ant-based algorithm,
- Evaluate and analyze the application of opposite information as an on-line method, where this strategy works with normal pheromone information,

- Study and analyze the changes on the Fitness Landscape of the problems of interest, produced by using opposite information,
- Study in detail the design of COISA, analyzing the effect and benefits of each method in the search process of the target algorithm,
- Create and design an extension of this strategy for other metaheuristic approaches (e.g., Evolutionary Algorithms, Tabu Search, Simulated Annealing, among others).

About the last item of this list, we have evaluated some preliminary ideas in some Harmony Search approaches. For this, we first published a paper entitled "Improving harmony search algorithms by using tonal variation: the case of Sudoku and MKP" [RMR17]. Here, some components inspired in the music improvisation process were proposed. Then, an article entitled "Collaborative Opposite Strategies for HS: The case of MKP" was presented in the OPTIMA 2017 conference.[1] Here, a Harmony Search approach was initialized using the anti-pheromone matrix produced by the strategies here proposed. Results showed that the Harmony Search approach reached better quality solutions using the opposite information. The idea is to evaluate new and different ways of including opposite information in other metaheuristics and also, not necessarily obtained by ant-based algorithms.

[1]The article is available in `https://github.com/nicolasemilio/OPTIMA17`

Appendices

Appendix A

Tables - MKP

This appendix presents the obtained results for each set of instances: 5×500, 10×500 and 30×500. For each set, two tables are presented: (1) the percentage gap between the best known solution and the best obtained solution by each algorithm and, (2) the percentage gap between the best known solution and the average of 30 independent executions. Tables A.1 and A.2 present results for the set 5×500. Tables A.3 and A.4 present results for the set 10×500. Tables A.5 and A.6 present results for the set 30×500.

tr	#	Best Known	AK	SOL-AK	WOL-AK	HOL-AK
0.25	1	120148	0.093	0.136	0.094	0.114
	2	117879	0.119	0.093	0.107	0.109
	3	121131	0.126	0.123	0.118	0.030
	4	120804	0.100	0.107	0.109	0.097
	5	122319	0.102	0.079	0.045	0.072
	6	122024	0.100	0.077	0.111	0.086
	7	119127	0.107	0.060	0.081	0.082
	8	120568	0.127	0.092	0.140	0.102
	9	121586	0.132	0.071	0.134	0.182
	10	120717	0.084	0.094	0.079	0.080
0.50	11	218428	0.060	0.064	0.050	0.075
	12	221202	0.075	0.047	0.074	0.079
	13	217542	0.067	0.065	0.040	0.046
	14	223560	0.022	0.028	0.032	0.023
	15	218966	0.005	0.034	0.025	0.041
	16	220530	0.046	0.030	0.037	0.042
	17	219989	0.045	0.031	0.001	0.043
	18	218215	0.043	0.045	0.053	0.044
	19	216976	0.028	0.033	0.034	0.049
	20	219719	0.017	0.033	0.025	0.046
0.75	21	295828	0.023	0.022	0.034	0.019
	22	308086	0.015	0.026	0.020	0.013
	23	299796	0.020	0.023	0.021	0.020
	24	306480	0.015	0.016	0.014	0.011
	25	300342	0.003	0.016	0.031	0.017
	26	302571	0.017	0.020	0.004	0.006
	27	301339	0.018	0.018	0.006	0.017
	28	306454	0.023	0.020	0.017	0.017
	29	302828	0.029	0.017	0.008	0.010
	30	299910	0.028	0.028	0.014	0.026

Table A.1: Set 5x500 - Percentage gap between the Best Known and the best quality solution obtained

tr	#	Best Known	AK	SOL-AK	WOL-AK	HOL-AK
0.25	1	120148	0.218	0.203	0.203	0.228
	2	117879	0.230	0.222	0.228	0.232
	3	121131	0.224	0.228	0.230	0.203
	4	120804	0.222	0.232	0.187	0.213
	5	122319	0.185	0.208	0.187	0.218
	6	122024	0.208	0.201	0.227	0.194
	7	119127	0.208	0.226	0.262	0.243
	8	120568	0.238	0.202	0.230	0.224
	9	121586	0.237	0.236	0.260	0.302
	10	120717	0.251	0.202	0.194	0.225
0.50	11	218428	0.118	0.121	3.269	0.123
	12	221202	0.124	0.113	0.120	0.135
	13	217542	0.109	0.099	0.103	0.099
	14	223560	0.073	0.077	0.078	0.072
	15	218966	0.080	0.079	0.087	0.093
	16	220530	0.102	0.095	0.109	0.090
	17	219989	0.095	0.091	0.083	0.086
	18	218215	0.099	0.095	0.095	0.086
	19	216976	0.083	0.094	0.085	0.094
	20	219719	0.097	0.101	0.112	0.111
0.75	21	295828	0.053	0.050	0.063	0.050
	22	308086	0.052	0.054	0.051	0.051
	23	299796	0.046	0.047	0.043	0.045
	24	306480	0.048	0.041	0.039	0.049
	25	300342	0.044	0.045	0.048	0.044
	26	302571	0.042	0.042	0.037	0.037
	27	301339	0.047	0.044	0.041	0.037
	28	306454	0.044	0.048	0.047	0.049
	29	302828	0.057	0.048	0.041	0.046
	30	299910	0.055	0.051	0.054	0.052

Table A.2: Set 5x500 - Percentage gap between the Best Known and the average of 30 independent executions

tr	#	Best Known	AK	SOL-AK	WOL-AK	HOL-AK
0.25	1	117821	0.266	0.252	0.225	0.265
	2	119249	0.227	0.203	0.197	0.273
	3	119215	0.216	0.218	0.276	0.149
	4	118829	0.135	0.164	0.220	0.170
	5	116530	0.289	0.282	0.259	0.291
	6	119504	0.223	0.156	0.156	0.209
	7	119827	0.200	0.159	0.248	0.239
	8	118344	0.308	0.235	0.213	0.182
	9	117815	0.243	0.277	0.283	0.246
	10	119251	0.168	0.226	0.290	0.236
0.50	11	217377	0.102	0.133	0.118	0.111
	12	219077	0.121	0.103	0.115	0.028
	13	217847	0.090	0.110	0.117	0.098
	14	216868	0.080	0.108	0.082	0.070
	15	213873	0.101	0.072	0.124	0.116
	16	215086	0.053	0.027	0.079	0.088
	17	217940	0.075	0.095	0.103	0.103
	18	219990	0.085	0.069	0.098	0.101
	19	214382	0.091	0.113	0.098	0.118
	20	220899	0.088	0.058	0.082	0.097
0.75	21	304387	0.033	0.040	0.040	0.035
	22	302379	0.038	0.039	0.030	0.043
	23	302417	0.041	0.045	0.055	0.029
	24	300784	0.050	0.047	0.028	0.017
	25	304374	0.038	0.027	0.020	0.017
	26	301836	0.068	0.058	0.078	0.085
	27	304952	0.030	0.036	0.027	0.030
	28	296478	0.009	0.027	0.035	0.048
	29	301359	0.037	0.038	0.036	0.040
	30	307089	0.051	0.045	0.051	0.044

Table A.3: Set 10x500 - Percentage gap between the Best Known and the best quality solution obtained

tr	#	Best Known	AK	SOL-AK	WOL-AK	HOL-AK
0.25	1	117821	0.459	0.504	0.455	0.547
	2	119249	0.407	0.465	0.428	0.433
	3	119215	0.548	0.436	0.532	0.449
	4	118829	0.321	0.355	0.379	0.337
	5	116530	0.450	0.416	0.437	0.556
	6	119504	0.373	0.373	0.353	0.395
	7	119827	0.411	0.426	0.431	0.481
	8	118344	0.439	0.465	0.446	0.435
	9	117815	0.437	0.426	0.460	0.483
	10	119251	0.422	0.430	0.460	0.443
0.50	11	217377	0.207	0.231	0.188	0.206
	12	219077	0.184	0.185	0.173	0.178
	13	217847	0.200	0.180	0.185	0.205
	14	216868	0.154	0.142	0.166	0.143
	15	213873	0.201	0.200	0.211	0.201
	16	215086	0.131	0.170	0.124	0.154
	17	217940	0.164	0.162	0.152	0.184
	18	219990	0.175	0.191	0.178	0.197
	19	214382	0.209	0.182	0.170	0.194
	20	220899	0.198	0.171	0.179	0.204
0.75	21	304387	0.075	0.083	0.073	0.086
	22	302379	0.080	0.074	0.067	0.079
	23	302417	0.092	0.088	0.087	0.090
	24	300784	0.085	0.088	0.074	0.075
	25	304374	0.070	0.070	0.058	0.072
	26	301836	0.117	0.118	0.122	0.111
	27	304952	0.081	0.065	0.062	0.064
	28	296478	0.085	0.080	0.083	0.091
	29	301359	0.083	0.075	0.085	0.082
	30	307089	0.093	0.084	0.088	0.092

Table A.4: Set 10x500 - Percentage gap between the Best Known and the average of 30 independent executions

tr	#	Best Known	AK	SOL-AK	WOL-AK	HOL-AK
0.25	1	116056	0.504	0.534	0.526	0.492
	2	114810	0.341	0.395	0.445	0.436
	3	116741	0.450	0.539	0.480	0.582
	4	115354	0.551	0.428	0.474	0.451
	5	116525	0.596	0.356	0.513	0.521
	6	115741	0.390	0.310	0.415	0.329
	7	114181	0.483	0.440	0.489	0.462
	8	114403	0.656	0.805	0.727	0.678
	9	115419	0.562	0.533	0.483	0.720
	10	117116	0.590	0.511	0.609	0.628
0.50	11	218104	0.170	0.140	0.190	0.159
	12	214648	0.260	0.287	0.252	0.267
	13	215978	0.147	0.144	0.163	0.191
	14	217910	0.240	0.234	0.243	0.264
	15	215689	0.248	0.211	0.201	0.237
	16	215919	0.279	0.276	0.299	0.312
	17	215907	0.151	0.113	0.138	0.193
	18	216542	0.201	0.224	0.209	0.223
	19	217340	0.148	0.153	0.199	0.163
	20	214739	0.269	0.249	0.271	0.249
0.75	21	301675	0.103	0.099	0.099	0.100
	22	300055	0.102	0.088	0.111	0.105
	23	305087	0.100	0.091	0.077	0.069
	24	302032	0.072	0.107	0.121	0.092
	25	304462	0.110	0.115	0.098	0.136
	26	297012	0.118	0.146	0.121	0.139
	27	303364	0.106	0.086	0.106	0.088
	28	307007	0.110	0.094	0.110	0.121
	29	303199	0.124	0.102	0.117	0.144
	30	300572	0.065	0.093	0.103	0.108

Table A.5: Set 30x500 - Percentage gap between the Best Known and the best quality solution obtained

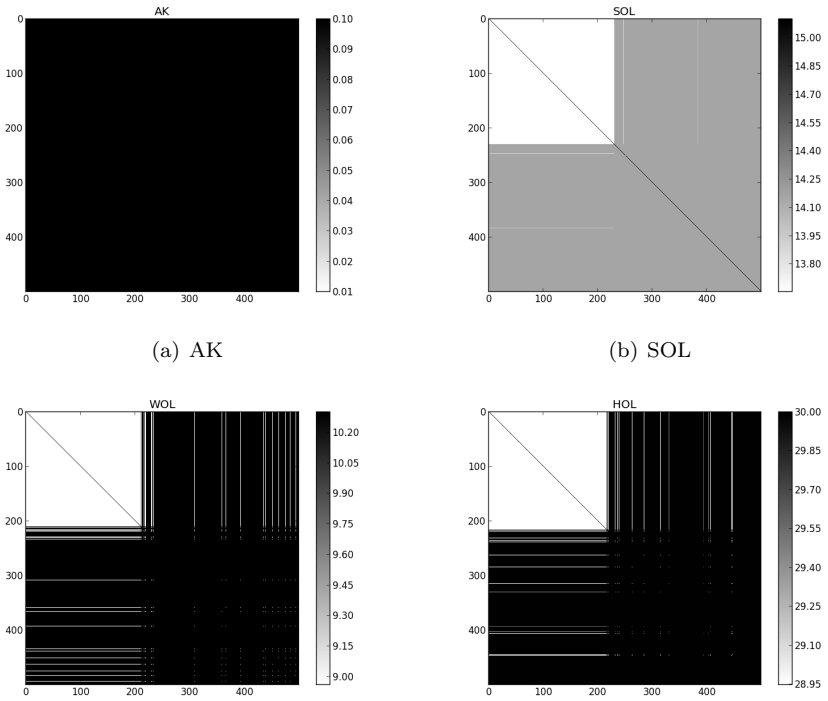
tr	#	Best Known	AK	SOL-AK	WOL-AK	HOL-AK
0.25	1	116056	0.946	0.797	0.872	0.934
	2	114810	0.785	0.740	0.783	0.729
	3	116741	0.929	0.854	0.900	0.874
	4	115354	0.967	0.837	0.960	0.937
	5	116525	0.935	0.874	0.915	1.024
	6	115741	0.814	0.796	0.769	0.777
	7	114181	0.919	0.840	0.891	0.843
	8	114403	1.041	1.067	1.025	1.010
	9	115419	0.921	0.936	0.919	1.059
	10	117116	0.963	0.897	0.929	0.871
0.50	11	218104	0.296	0.308	0.357	0.335
	12	214648	0.401	0.368	0.427	0.434
	13	215978	0.284	0.308	0.269	0.315
	14	217910	0.390	0.382	0.387	0.414
	15	215689	0.389	0.402	0.393	0.390
	16	215919	0.408	0.398	0.431	0.429
	17	215907	0.283	0.268	0.274	0.288
	18	216542	0.370	0.370	0.377	0.362
	19	217340	0.304	0.311	0.287	0.296
	20	214739	0.417	0.398	0.402	0.396
0.75	21	301675	0.151	0.170	0.166	0.172
	22	300055	0.160	0.163	0.169	0.162
	23	305087	0.153	0.147	0.138	0.143
	24	302032	0.179	0.181	0.182	0.170
	25	304462	0.180	0.188	0.190	0.182
	26	297012	0.191	0.193	0.199	0.198
	27	303364	0.173	0.171	0.181	0.178
	28	307007	0.186	0.184	0.187	0.185
	29	303199	0.197	0.194	0.196	0.209
	30	300572	0.143	0.162	0.152	0.157

Table A.6: Set 30x500 - Percentage gap between the Best Known and the average of 30 independent executions

Appendix B

Matrices - MKP

Initial and final pheromone matrices obtained by Ant Knapsack (AK), SOL-AK, WOL-AK and HOL-AK are here presented. These matrices were obtained for the instance #7 of the set $5x500$ (tr=0.50). Figure B.1 shows the initial obtained matrices and figure B.2 shows the final obtained



(a) AK

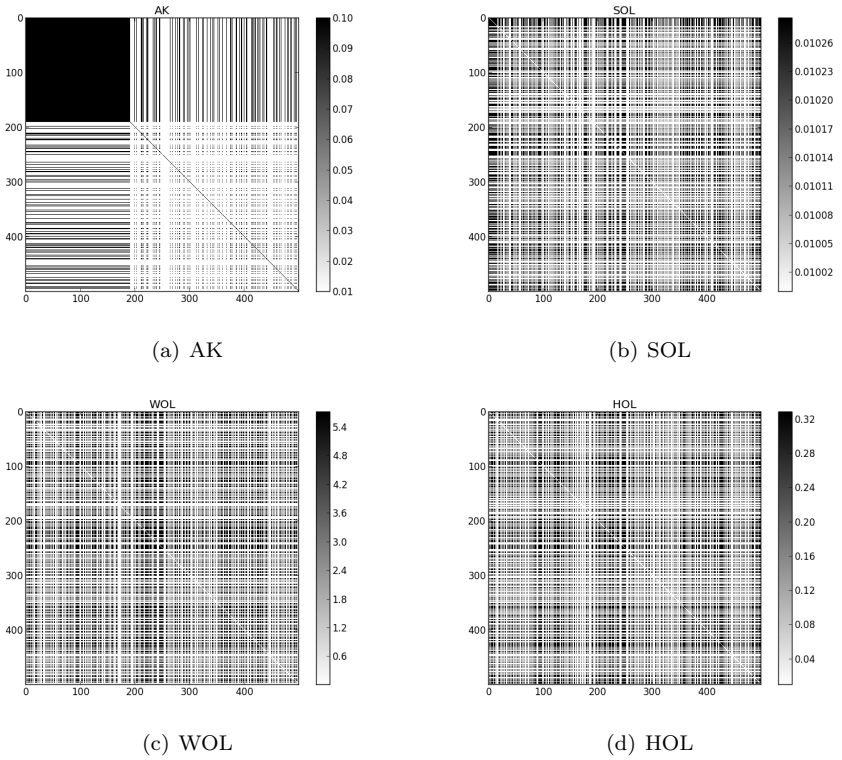
(b) SOL

(c) WOL

(d) HOL

Figure B.1: Initial pheromone matrices

matrices.



(a) AK

(b) SOL

(c) WOL

(d) HOL

Figure B.2: Final pheromone matrices

Appendix C

Fitness-Distance Plots - MKP

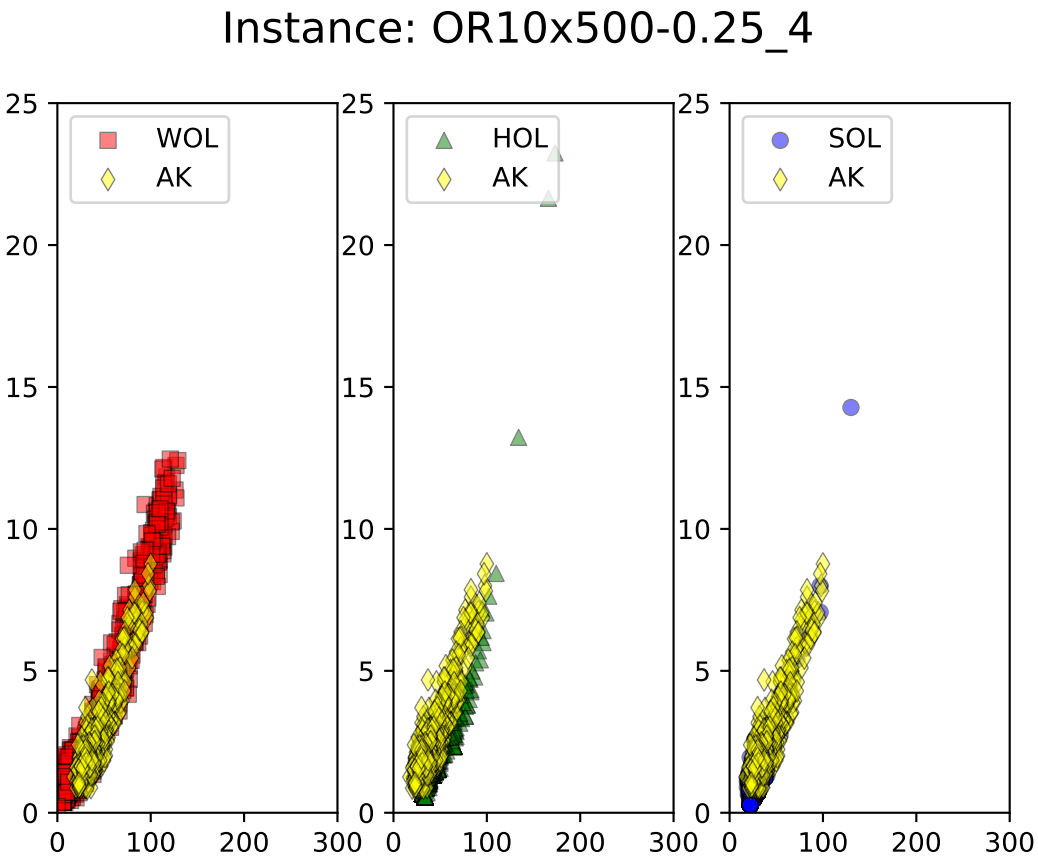


Figure C.1: Fitness-Distance plots of Ant Knapsack and each Method separately for an instance from the set 10×500

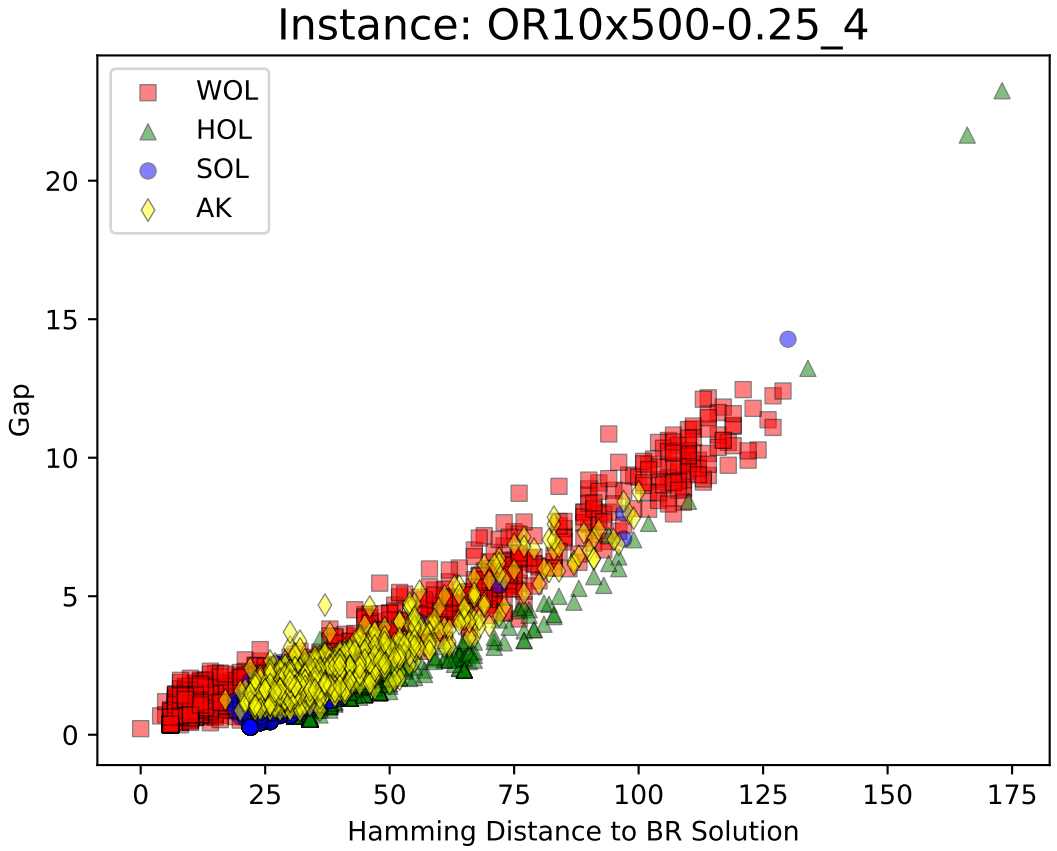


Figure C.2: Fitness-Distance plot with all approaches for instance an from the set 10×500

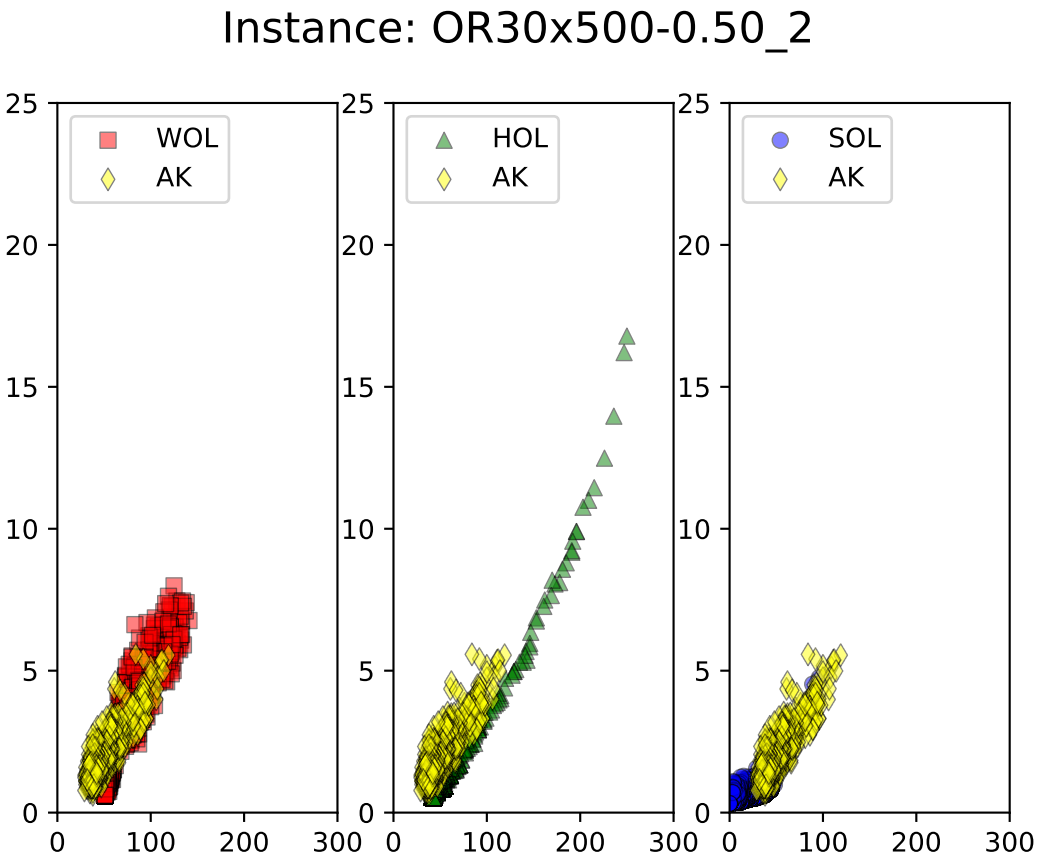


Figure C.3: Fitness-Distance plots of Ant Knapsack and each Method separately for an instance from the set 30×500

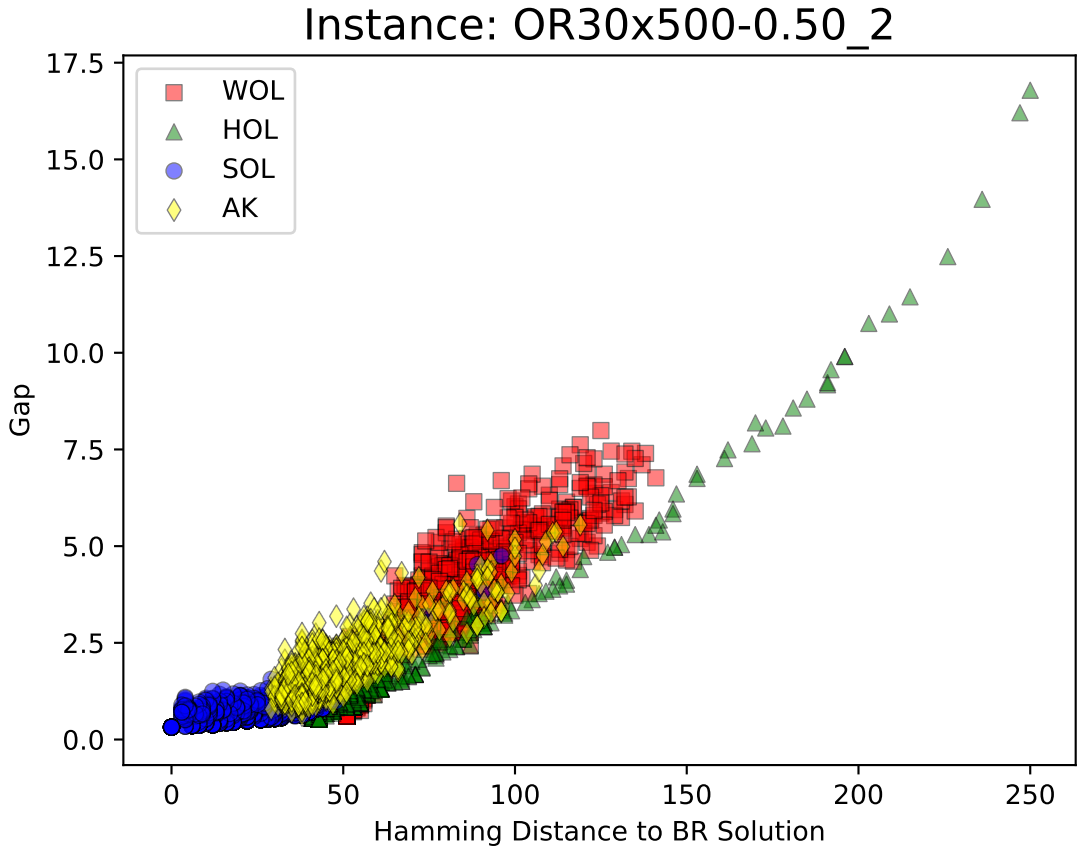


Figure C.4: Fitness-Distance plot with all approaches for an instance from the set 30×500

Bibliography

[AAR12] M. A. Ahandani and H. Alavi-Rad. Opposition-based learning in the shuffled differential evolution algorithm. *Soft Computing*, 16(8):1303–1337, 2012.

[Aha16] M. A. Ahandani. Opposition-based learning in the shuffled bidirectional differential evolution algorithm. *Swarm and Evolutionary Computation*, 26:64–85, 2016.

[AKK+97] D. Achlioptas, L. M. Kirousis, E. Kranakis, D. Krizanc, M. Molloy, and Y. C. Stamatiou. Random constraint satisfaction: A more accurate picture. In G. Smolka, editor, *Principles and Practice of Constraint Programming - CP97, Third International Conference, Linz, Austria, October 29 - November 1, 1997, Proceedings*, volume 1330 of *Lecture Notes in Computer Science*, pages 107–120. Springer, 1997.

[Amd67] G. M. Amdahl. Validity of the single processor approach to achieving large scale computing capabilities. In *American Federation of Information Processing Societies: Proceedings of the AFIPS '67 Spring Joint Computer Conference, April 18-20, 1967, Atlantic City, New Jersey, USA*, volume 30 of *AFIPS Conference Proceedings*, pages 483–485. AFIPS / ACM / Thomson Book Company, Washington D.C., 1967.

[ASG04] I. Alaya, C. Solnon, and K. Ghedira. Ant algorithm for the multi-dimensional knapsack problem. In *International Conference on Bioinspired Optimization Methods and their Applications (BIOMA 2004)*, pages 63–72. Citeseer, 2004.

[Bas16] M. Basu. Quasi-oppositional differential evolution for optimal reactive power dispatch. *International Journal of Electrical Power & Energy Systems*, 78:29–40, 2016.

[BB16] B. Bošković and J. Brest. Genetic algorithm with advanced mechanisms applied to the protein structure prediction in a hydrophobic-polar model and cubic lattice. *Applied Soft Computing*, 45:61–70, 2016.

[BFM97] T. Back, D. B. Fogel, and Z. Michalewicz, editors. *Handbook of Evolutionary Computation*. IOP Publishing Ltd., Bristol, UK, 1st edition, 1997.

[BHS97] B. Bullnheimer, R. F. Hartl, and C. Strauss. A new rank based version of the ant system - a computational study. *Central European Journal for Operations Research and Economics*, 7:25–38, 1997.

[BMG14] A. Banerjee, V. Mukherjee, and S. P. Ghoshal. An opposition-based harmony search algorithm for engineering optimization problems. *Ain Shams Engineering Journal*, 5(1):85–101, 2014.

[BMM01] G. J. Beaujon, S. P. Marin, and G. C. McDonald. Balancing and optimizing a portfolio of r&d projects. *Naval Research Logistics (NRL)*, 48(1):18–40, 2001.

[BPS99] S. C. Brailsford, C. N. Potts, and B. M. Smith. Constraint satisfaction problems: Algorithms and applications. *European Journal of Operational Research*, 119(3):557–581, 1999.

[CB98] P.C. Chu and J.E. Beasley. A genetic algorithm for the multidimensional knapsack problem. *Journal of Heuristics*, 4(1):63–86, 1998.

[CC10] Y. Chi and G. Cai. Particle swarm optimization with opposition-based disturbance. In *Informatics in Control, Automation and Robotics (CAR), 2010 2nd International Asia Conference on*, volume 2, pages 223–226, 2010.

[CDM92] A. Colorni, M. Dorigo, and V Maniezzo. Distributed optimization by ant colonies. In *Toward a Practice of Autonomous Systems: Proceedings of the First European Conference on Artificial Life*, page 134. MIT Press, 1992.

[CdVH02] O. Cordon, I. de Viana, and F. Herrera. Analysis of the best-worst ant system and its variants on the QAP. In M. Dorigo, G. Di Caro, and M. Sampels, editors, *Ant Algorithms, Third International Workshop, ANTS 2002, Brussels, Belgium, September 12-14, 2002, Proceedings*, volume 2463 of *Lecture Notes in Computer Science*, pages 228–234. Springer, 2002.

[CdVHM00] O. Cordon, I. de Viana, F. Herrera, and L. Moreno. A new aco model integrating evolutionary computation concepts: The best-worst ant system. 2000.

[CGM12] A. Chatterjee, S. P. Ghoshal, and V. Mukherjee. Solution of combined economic and emission dispatch problems of power systems by an opposition-based harmony search

algorithm. *International Journal of Electrical Power & Energy Systems*, 39(1):9–20, 2012.

[CKT91] P. C. Cheeseman, B. Kanefsky, and W.M. Taylor. Where the really hard problems are. In J. Mylopoulos and R. Reiter, editors, *Proceedings of the 12th International Joint Conference on Artificial Intelligence. Sydney, Australia, August 24-30, 1991*, pages 331–340. Morgan Kaufmann, 1991.

[CT15] M.-Y. Cheng and D.-H. Tran. Opposition-based multiple objective differential evolution (omode) for optimizing work shift schedules. *Automation in Construction*, 55:1–14, 2015.

[DAGP90] J.-L. Deneubourg, S. Aron, S. Goss, and J. M. Pasteels. The self-organizing exploratory pattern of the argentine ant. *Journal of insect behavior*, 3(2):159–168, 1990.

[DBFS+00] B. De Backer, V. Furnon, P. Shaw, P. Kilby, and P. Prosser. Solving vehicle routing problems using constraint programming and metaheuristics. *Journal of Heuristics*, 6(4):501–523, 2000.

[DCZZ09] C. Dai, W. Chen, Y. Zhu, and X. Zhang. Seeker optimization algorithm for optimal reactive power dispatch. *IEEE Transactions on Power Systems*, 24(3):1218–1231, 2009.

[DDCG99] M. Dorigo, G. Di Caro, and L. M. Gambardella. Ant algorithms for discrete optimization. *Artificial life*, 5(2):137–172, 1999.

[DG97] M. Dorigo and L. M. Gambardella. Ant colony system: a cooperative learning approach to the traveling salesman problem. *IEEE Trans. Evolutionary Computation*, 1(1):53–66, 1997.

[DKZ16] W. Dong, L. Kang, and W. Zhang. Opposition-based particle swarm optimization with adaptive mutation strategy. *Soft Computing*, pages 1–10, 2016.

[DMC96] M. Dorigo, V. Maniezzo, and A. Colorni. Ant System: optimization by a colony of cooperating agents. *IEEE Trans. Systems, Man, and Cybernetics, Part B*, 26(1):29–41, 1996.

[DÖB16] J. H. Drake, E. Özcan, and E. K. Burke. A case study of controlling crossover in a selection hyper-heuristic framework using the multidimensional knapsack problem. *Evolutionary Computation*, 24(1):113–141, 2016.

[Dor92] M. Dorigo. Optimization, learning and natural algorithms. *Ph. D. Thesis, Politecnico di Milano, Italy*, 1992.

[Dor93] M Dorigo. Parallel ant system: An experimental study. *Unpublished manuscript*, 1993.

[DP87] R. Dechter and J. Pearl. Network-based heuristics for constraint-satisfaction problems. *Artificial Intelligence*, 34(1):1–38, 1987.

[DS04] M. Dorigo and T. Stützle. *Ant Colony Optimization*. Bradford Company, Scituate, MA, USA, 2004.

[DW09] N. Dong and Y. Wang. Multiobjective differential evolution based on opposite operation. In *2009 International Conference on Computational Intelligence and Security, CIS 2009*, pages 123–127. IEEE Computer Society, 2009.

[DYWC10] X. Dong, S. Yu, Z. Wu, and Z. Chen. A hybrid parallel evolutionary algorithm based on elite-subspace strategy and space transformation search. In *High Performance Computing and Applications*, pages 139–145. Springer, 2010.

[ER11] A. Esmailzadeh and S. Rahnamayan. Opposition-based differential evolution with protective generation jumping. In *Differential Evolution (SDE), 2011 IEEE Symposium on*, pages 1–8, 2011.

[ESD09] M. Ergezer, D. Simon, and D. Du. Oppositional biogeography-based optimization. In *Proceedings of the IEEE International Conference on Systems, Man and Cybernetics, San Antonio, TX, USA, 11-14 October 2009*, pages 1009–1014. IEEE, 2009.

[FCM+92] T. Friesz, H.-J. Cho, N. J. Mehta, R. L. Tobin, and G. Anandalingam. A simulated annealing approach to the network design problem with variational inequality constraints. *Transportation Science*, 26(1):18–26, 1992.

[FCMS14] H. Fingler, E. N. Cáceres, H. Mongelli, and S. W. Song. A CUDA based solution to the multidimensional knapsack problem using the ant colony optimization. In D. Abramson, M. Lees, V. V. Krzhizhanovskaya, J. J. Dongarra, and P. M. A. Sloot, editors, *Proceedings of the International Conference on Computational Science, ICCS 2014, Cairns, Queensland, Australia, 10-12 June, 2014*, volume 29 of *Procedia Computer Science*, pages 84–94. Elsevier, 2014.

[Fid05] S. Fidanova. Heuristics for multiple knapsack problem. In N. Guimarães and P. T. Isaías, editors, *AC 2005, Proceedings of the IADIS International Conference on Applied Computing, Algarve, Portugal, February 22-25, 2005, Volume 2*, pages 255–260. IADIS, 2005.

[Fos95] I. Foster. *Designing and building parallel programs*, volume 78. Addison Wesley Publishing Company Boston, 1995.

[Fre04] A. Freville. The multidimensional 01 knapsack problem: An overview. *European Journal of Operational Research*, 155(1):1 – 21, 2004.

[GADP89] S. Goss, S. Aron, J.-L. Deneubourg, and J. M. Pasteels. Self-organized shortcuts in the argentine ant. *Naturwissenschaften*, 76(12):579–581, 1989.

[GD95] L. M. Gambardella and Marco D. Ant-q: A reinforcement learning approach to the traveling salesman problem. In A. Prieditis and S. J. Russell, editors, *Machine Learning, Proceedings of the Twelfth International Conference on Machine Learning, Tahoe City, California, USA, July 9-12, 1995*, pages 252–260. Morgan Kaufmann, 1995.

[Gee00] Zong Woo Geem. *Optimal design of water distribution networks using harmony search*. PhD thesis, Korea University, 2000.

[GH97] P. Galinier and J.-K. Hao. Tabu search for maximal constraint satisfaction problems. In G. Smolka, editor, *Principles and Practice of Constraint Programming - CP97, Third International Conference, Linz, Austria, October 29 - November 1, 1997, Proceedings*, volume 1330 of *Lecture Notes in Computer Science*, pages 196–208. Springer, 1997.

[GK96] F. Glover and G. A. Kochenberger. Critical event tabu search for multidimensional knapsack problems. In *Meta-Heuristics*, pages 407–427. Springer, 1996.

[GLH12] W. Gao, S. Liu, and L. Huang. Particle swarm optimization with chaotic opposition-based population initialization and stochastic search technique. *Communications in Nonlinear Science and Numerical Simulation*, 17(11):4316–4327, 2012.

[GMP+01] I. P. Gent, E. Macintyre, P. Prosser, B. M. Smith, and T. Walsh. Random constraint satisfaction: Flaws and structure. *Constraints*, 6(4):345–372, 2001.

[GMPW96] I. P. Gent, E. MacIntyre, P. Prosser, and T. Walsh. The constrainedness of search. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence - Volume 1*, AAAI'96, pages 246–252. AAAI Press, 1996.

[HE80] R. M. Haralick and G. L. Elliott. Increasing tree search efficiency for constraint satisfaction problems. *Artificial Intelligence*, 14(3):263–313, 1980.

[HF98] S. Hanafi and A. Fréville. An efficient tabu search approach for the 01 multidimensional knapsack problem. *European Journal of Operational Research*, 106(2-3):659–675, 1998.

[HH07] L. Han and X. He. A novel opposition-based particle swarm optimization for noisy problems. In *Natural Computation, 2007. ICNC 2007. Third International Conference on*, volume 3, pages 624–629, 2007.

[JJB09] H. Jabeen, Z. Jalil, and A. R. Baig. Opposition based initialization in particle swarm optimization (O-PSO). In *Proceedings of the 11th Annual Conference Companion on Genetic and Evolutionary Computation Conference: Late Breaking Papers*, pages 2047–2052. ACM, 2009.

[Kau13] M. Kaucic. A multi-start opposition-based particle swarm optimization algorithm with adaptive velocity for bound constrained global optimization. *Journal of Global Optimization*, 55(1):165–188, 2013.

[KE95] J. Kennedy and R. Eberhart. Particle swarm optimization. In *Neural Networks, 1995. Proceedings., IEEE International Conference on*, volume 4, pages 1942–1948 Vol.4, 1995.

[KGOL15] X. Kong, L. Gao, H. Ouyang, and S. Li. Solving largescale multidimensional knapsack problems with a new binary harmony search algorithm. *Computers & OR*, 63:7–22, 2015.

[KMN95] H. Kanoh, M. Matsumoto, and S. Nishihara. Genetic algorithms for constraint satisfaction problems. In *1995 IEEE International Conference on Systems, Man and Cybernetics. Intelligent Systems for the 21st Century*, volume 1, pages 626–631 vol.1, 1995.

[KPP04] H. Kellerer, U. Pferschy, and D. Pisinger. *Knapsack problems*. Springer, 2004.

[Kum92] V. Kumar. Algorithms for constraint-satisfaction problems: A survey. *AI Magazine*, 13(1):32–44, 1992.

[LHGL18] X. Lai, J.-K. Hao, F. Glover, and Z. Lu. A two-phase tabu-evolutionary algorithm for the 0-1 multidimensional knapsack problem. *Information Sciences*, 436 - 437:282 – 301, 2018.

[LIS12] M. Lopez-Ibanez and M. Stutzle. The automatic design of multiobjective ant colony optimization algorithms. *IEEE Transactions on Evolutionary Computation*, 16(6):861–875, 2012.

[LQSB06] J. J. Liang, A. K. Qin, P. N. Suganthan, and S. Baskar. Comprehensive learning particle swarm optimizer for global optimization of multimodal functions. *IEEE transactions on evolutionary computation*, 10(3):281–295, 2006.

[LQSHD13] J. J. Liang, B. Y. Qu, P. N. Suganthan, and A. G. Hernández-Díaz. Problem definitions and evaluation criteria for the cec 2013 special session on real-parameter optimization. *Computational Intelligence Laboratory, Zhengzhou University, Zhengzhou, China and Nanyang Technological University, Singapore, Technical Report*, 201212, 2013.

[LT12] T. Lust and J. Teghem. The multiobjective multidimensional knapsack problem: a survey and a new approach. *International Transactions in Operational Research*, 19(4):495–520, 2012.

[LZY12] S. W. Leung, X. Zhang, and S. Y. Yuen. Multiobjective differential evolution algorithm with opposition-based parameter control. In *2012 IEEE Congress on Evolutionary Computation*, pages 1–8. IEEE, 2012.

[Mac77] A. K. Mackworth. Consistency in networks of relations. *Artificial Intelligence*, 8(1):99–118, 1977.

[Mal07] A. R. Malisia. Investigating the application of opposition-based ideas to ant algorithms. Master's thesis, University of Waterloo, 2007.

[Mal08] A. R. Malisia. Improving the exploration ability of ant-based algorithms. In H. R. Tizhoosh and M. Ventresca, editors, *Oppositional Concepts in Computational Intelligence*, volume 155 of *Studies in Computational Intelligence*, pages 121–142. Springer, 2008.

[MCS01] H. Meier, N. Christofides, and G. Salkin. Capital budgeting under uncertainty: an integrated approach using contingent claims analysis and integer programming. *Operations Research*, 49(2):196–206, 2001.

[MLQ+14] X. Ma, F. Liu, Y. Qi, M. Gong, M. Yin, L. Li, L. Jiao, and J. Wu. MOEA/D with opposition-based learning for multiobjective optimization problem. *Neurocomputing*, 146:48–64, 2014.

[Mon74] Ugo Montanari. Networks of constraints: Fundamental properties and applications to picture processing. *Information Science*, 7:95–132, 1974.

[MP17] T. Meng and Q.-K. Pan. An improved fruit fly optimization algorithm for solving the multidimensional knapsack problem. *Applied Soft Computing*, 50:79 – 93, 2017.

[MPSW98] E. MacIntyre, P. Prosser, B. Smith, and T. Walsh. Random constraint satisfaction: Theory meets practice. In M. Maher and J.-P. Puget, editors, *Principles and Practice of Constraint Programming - CP98, 4th International Conference, Pisa, Italy, October 26-30, 1998, Proceedings*, volume 1520 of *Lecture Notes in Computer Science*, pages 325–339. Springer, 1998.

[MR02] J. Montgomery and M. Randall. Anti-pheromone as a tool for better exploration of search space. In M. Dorigo, G. Di Caro, and M. Sampels, editors, *Ant Algorithms, Third International Workshop, ANTS 2002, Brussels, Belgium, September 12-14, 2002, Proceedings*, volume 2463 of *Lecture Notes in Computer Science*, pages 100–110. Springer, 2002.

[MR13] E. Montero and M.-C. Riff. A new algorithm for reducing metaheuristic design effort. In *Proceedings of the IEEE Congress on Evolutionary Computation, CEC 2013, Cancun, Mexico, June 20-23, 2013*, pages 3283–3290. IEEE, 2013.

[MRD17] S. Mahdavi, S. Rahnamayan, and K. Deb. Opposition based learning: A literature review. *Swarm and Evolutionary Computation*, 2017.

[MS15] B. Mandal and T. Si. Opposition based particle swarm optimization with exploration and exploitation through gbest. In *Advances in Computing, Communications and Informatics (ICACCI), 2015 International Conference on*, pages 245–250. IEEE, 2015.

[MT07] A.R. Malisia and H.R. Tizhoosh. Applying opposition-based ideas to the ant colony system. In *2007 IEEE Swarm Intelligence Symposium, SIS 2007, Honolulu, Hawaii, USA, April 1-5, 2007*, pages 182–189. IEEE, 2007.

[NI98] K. Nonobe and T. Ibaraki. A tabu search approach to the constraint satisfaction problem as a general problem solver. *European Journal of Operational Research*, 106(2-3):599–623, 1998.

[NZW+14] Q. Niu, H. Zhang, X. Wang, K. Li, and G. W. Irwin. A hybrid harmony search with arithmetic crossover operation for economic dispatch. *International journal of electrical power & energy systems*, 62:237–257, 2014.

[OL96] I. H. Osman and G. Laporte. Metaheuristics: A bibliography. *Annals of Operations Research*, 63(5):511–623, 1996.

[OQB09] G. Ochoa, R. Qu, and E. Burke. Analyzing the landscape of a graph based hyper-heuristic for timetabling problems. In F. Rothlauf, editor, *Genetic and Evolutionary Computation Conference, GECCO 2009, Proceedings, Montreal, Québec, Canada, July 8-12, 2009*, pages 341–348. ACM, 2009.

[OTVD08] G. Ochoa, M. Tomassini, S. Vérel, and C. Darabos. A study of NK landscapes' basins and local optima networks. In *Proceedings of the 10th annual conference on Genetic and evolutionary computation*, pages 555–562. ACM, 2008.

[Pir87] H. Pirkul. A heuristic solution procedure for the multiconstraint zero-one knapsack problem. *Naval Research Logistics (NRL)*, 34(2):161–172, 1987.

[PNC11] M. Pedemonte, S. Nesmachnow, and H. Cancela. A survey on parallel ant colony optimization. *Applied Soft Computing*, 11(8):5181 – 5197, 2011.

[Pro93a] P. Prosser. Domain filtering can degrade intelligent backtracking search. In R. Bajcsy, editor, *Proceedings of the 13th International Joint Conference on Artificial Intelligence. Chambéry, France, August 28 - September 3, 1993*, pages 262–267. Morgan Kaufmann, 1993.

[Pro93b] P. Prosser. Hybrid algorithms for the constraint satisfaction problem. *Computational Intelligence*, 9:268–299, 1993.

[PRP06] J. Puchinger, G. R. Raidl, and U. Pferschy. The core concept for the multidimensional knapsack problem. In J. Gottlieb and G. R. Raidl, editors, *Evolutionary Computation*

in Combinatorial Optimization, 6th European Conference, EvoCOP 2006, Budapest, Hungary, April 10-12, 2006, Proceedings, volume 3906 of *Lecture Notes in Computer Science*, pages 195–208. Springer, 2006.

[PRP10] J. Puchinger, G. R. Raidl, and U. Pferschy. The multidimensional knapsack problem: Structure and algorithms. *Journal on Computing*, 22(2):250–265, 2010.

[QF11] A. K. Qin and F. Forbes. Dynamic regional harmony search with opposition and local learning. In *Proceedings of the 13th Annual Conference Companion on Genetic and Evolutionary Computation*, GECCO '11, pages 53–54, 2011.

[RL02] M. Randall and A. Lewis. A parallel implementation of ant colony optimization. *Journal of Parallel and Distributed Computing*, 62(9):1421–1432, 2002.

[RMR17] N. Rojas-Morales and M.-C. Riff. Improving harmony search algorithms by using tonal variation: the case of sudoku and mkp. *Connection Science*, pages 1–27, 2017.

[RMRC18] N. Rojas-Morales, M.-C. Riff, and C. Coello. A cooperative opposite-inspired learning strategy for ant-based algorithms. In *Eleventh International Conference on Swarm Intelligence, ANTS 2018, Rome, Italy (in Press)*. Springer LNCS Series, 2018.

[RMRM15] N. Rojas-Morales, M.-C. Riff, and E. Montero. Using anti-pheromone to identify core objects for multidimensional knapsack problems: A two-step ants based approach. In *Proceedings of the Companion Publication of the 2015 Annual Conference on Genetic and Evolutionary Computation*, pages 1469–1470. ACM, 2015.

[RMRM16a] N. Rojas-Morales, M.-C. Riff, and E. Montero. Ants can learn from the opposite. In T. Friedrich, F. Neumann, and A. M. Sutton, editors, *Proceedings of the 2016 on Genetic and Evolutionary Computation Conference, Denver, CO, USA, July 20 - 24, 2016*, pages 389–396. ACM, 2016.

[RMRM16b] N. Rojas-Morales, M.-C. Riff, and E. Montero. Learning from the opposite: Strategies for ants that solve multidimensional knapsack problem. In *IEEE Congress on Evolutionary Computation, CEC 2016, Vancouver, BC, Canada, July 24-29, 2016*, pages 193–200. IEEE, 2016.

[RMRN18] N. Rojas-Morales, M.-C. Riff, and B. Neveu. Feasibility and Availability based Heuristics for ACO algorithms solving Binary CSP (in Press). In *IEEE Congress on Evolutionary Computation, CEC 2018, Rio de Janeiro, Brazil, July 08-13, 2018*, 2018.

[RRM17] N. Rojas-Morales, M.-C. Riff, and E. Montero. A survey and classification of opposition-based metaheuristics. *Computers & Industrial Engineering*, 110:424–435, 2017.

[RSL77] D. J. Rosenkrantz, R. E. Stearns, and P. M. Lewis. An analysis of several heuristics for the traveling salesman problem. *SIAM J. Comput.*, 6(3):563–581, 1977.

[RTS06] S. Rahnamayan, H.R. Tizhoosh, and M.M.A. Salama. Opposition-based differential evolution algorithms. In *IEEE International Conference on Evolutionary Computation, CEC 2006*, pages 2010–2017, 2006.

[RTS07a] S. Rahnamayan, H. R. Tizhoosh, and M. Salama. A novel population initialization method for accelerating evolutionary algorithms. *Computers & Mathematics with Applications*, 53(10):1605 – 1614, 2007.

[RTS07b] S. Rahnamayan, H. R. Tizhoosh, and M. M. A. Salama. Opposition-based differential evolution (ODE) with variable jumping rate. In *IEEE Symposium on Foundations of Computational Intelligence (FOCI 2007)*, pages 81–88. IEEE, 2007.

[RTS07c] S. Rahnamayan, H.R. Tizhoosh, and M.M.A. Salama. Quasi-oppositional differential evolution. In *Proceedings of the IEEE Congress on Evolutionary Computation, CEC 2007, 25-28 September 2007, Singapore*, pages 2229–2236. IEEE, 2007.

[RW08] S. Rahnamayan and G. G. Wang. Solving large scale optimization problems by opposition-based differential evolution (ODE). *WSEAS Trans. on Computation*, 7(10):1792–1804, 2008.

[RW09] S. Rahnamayan and G. G. Wang. Center-based sampling for population-based algorithms. In *Proceedings of the IEEE Congress on Evolutionary Computation, CEC 2009, Trondheim, Norway, 18-21 May, 2009*, pages 933–938. IEEE, 2009.

[SBHR97] R. Schoonderwoerd, J. L. Bruten, O. E. Holland, and L. J. M. Rothkrantz. Ant-based load balancing in telecommunications networks. *Adaptive Behavior*, 5(2):169–207, 1997.

[SBM+09] F. Shahzad, A. R. Baig, S. Masood, M. Kamran, and N. Naveed. Opposition-based particle swarm optimization with velocity clamping (OVCPSO). In *Advances in Computational Intelligence*, pages 339–348. Springer, 2009.

[SD08] K. Socha and M. Dorigo. Ant colony optimization for continuous domains. *European Journal of Operational Research*, 185(3):1155 – 1173, 2008.

[SDB14] T. Si, A. De, and A. K. Bhattacharjee. Particle swarm optimization with generalized opposition based learning in particle's pbest position. In *Circuit, Power and Computing Technologies (ICCPCT), 2014 International Conference on*, pages 1662–1667. IEEE, 2014.

[SF94] D. Sabin and E. C. Freuder. Contradicting conventional wisdom in constraint satisfaction. In A. Borning, editor, *Principles and Practice of Constraint Programming, Second International Workshop, PPCP'94, Rosario, Orcas Island, Washington, USA, May 2-4, 1994, Proceedings*, volume 874 of *Lecture Notes in Computer Science*, pages 10–20. Springer, 1994.

[SH96] T. Stützle and H. H. Hoos. Improving the ant system: A detailed report on the MAX–MIN ant system. *FG Intellektik, FB Informatik, TU Darmstadt, Germany, Tech. Rep. AIDA–96–12*, 1996.

[SH00] T. Stützle and H. H. Hoos. MAX-MIN ant system. *Future Generation Comp. Syst.*, 16(8):889–914, 2000.

[SHL+05] P. N. Suganthan, N. Hansen, J. J. Liang, K. Deb, Y. Chen, A. Auger, and S. Tiwari. Problem definitions and evaluation criteria for the CEC 2005 special session on real-parameter optimization. *KanGAL report*, 2005005:2005, 2005.

[Sim08] D. Simon. Biogeography-based optimization. *IEEE Trans. Evolutionary Computation*, 12(6):702–713, 2008.

[Ski16] R. Skinderowicz. The gpu-based parallel ant colony system. *Journal of Parallel and Distributed Computing*, 98:48–60, 2016.

[SLIP+12] T. Stützle, M. López-Ibánez, P. Pellegrini, M. Maur, M. M. De Oca, M. Birattari, and M. Dorigo. Parameter adaptation in ant colony optimization. In Y. Hamadi, E. Monfroy, and F. Saubion, editors, *Autonomous Search*, pages 191–215. Springer, 2012.

[SM16] G. Shankar and V. Mukherjee. Quasi oppositional harmony search algorithm based controller tuning for load frequency control of multi-source multi-area power system. *International Journal of Electrical Power & Energy Systems*, 75:289–302, 2016.

[SMG14] B. Shaw, V. Mukherjee, and S.P. Ghoshal. Solution of reactive power dispatch of power systems by an opposition-based gravitational search algorithm. *International Journal of Electrical Power & Energy Systems*, 55:29–40, 2014.

[Sol02] C. Solnon. Ants can solve constraint satisfaction problems. *IEEE transactions on evolutionary computation*, 6(4):347–357, 2002.

[SP97] R. Storn and K. Price. Differential evolution – a simple and efficient heuristic for global optimization over continuous spaces. *Journal of Global Optimization*, 11(4):341–359, 1997.

[SSM15] C. K. Shiva, G. Shankar, and V. Mukherjee. Automatic generation control of power system using a novel quasi-oppositional harmony search algorithm. *International Journal of Electrical Power & Energy Systems*, 73:787–804, 2015.

[Stü97] T Stützle. MAX-MIN Ant System for quadratic assignment problems. *Germany: Intellektik Group, Department of Computer Science, Darmstadt University of Technology (Report No. AIDA-97-04)*, 1997.

[Tal02] E.-G Talbi. A taxonomy of hybrid metaheuristics. *Journal of heuristics*, 8(5):541–564, 2002.

[Tal09] E.-G. Talbi. *Metaheuristics: from design to implementation*, volume 74. John Wiley & Sons, 2009.

[TB99] G. Theraulaz and E. Bonabeau. A brief history of stigmergy. *Artificial Life*, 5(2):97–116, 1999.

[TDR16] A. Talukder, K. Deb, and S. Rahnamayan. Maintaining diversity in the bounded pareto-set: A case of opposition based solution generation scheme. In *Proceedings of the 2016 on Genetic and Evolutionary Computation Conference Companion*, pages 945–951. ACM, 2016.

[Tiz05] H.R. Tizhoosh. Opposition-based learning: A new scheme for machine intelligence. In *2005 International Conference on Computational Intelligence for Modelling Control and Automation (CIMCA 2005), International Conference on Intelligent Agents, Web Technologies and Internet Commerce (IAWTIC 2005), 28-30 November 2005, Vienna, Austria*, pages 695–701. IEEE Computer Society, 2005.

[TPC06] J. Tavares, F. Pereira, and E. Costa. The role of representation on the multidimensional knapsack problem by means of fitness landscape analysis. In *2006 IEEE International Conference on Evolutionary Computation*, pages 2307–2314. IEEE, 2006.

[TPC08] Jorge Tavares, Francisco B Pereira, and Ernesto Costa. Multidimensional knapsack problem: A fitness landscape analysis. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, 38(3):604–616, 2008.

[TVR08] H. R. Tizhoosh, M. Ventresca, and S. Rahnamayan. Opposition-based computing. In H. R. Tizhoosh and M. Ventresca, editors, *Oppositional Concepts in Computational Intelligence*, volume 155 of *Studies in Computational Intelligence*, pages 11–28. Springer, 2008.

[TYS+07] K. Tang, X. Yáo, P. N. Suganthan, C. MacNish, Y. Chen, C. Chen, and Z. Yang. Benchmark functions for the CEC'2008 special session and competition on large scale global optimization. *Nature Inspired Computation and Applications Laboratory, USTC, China*, pages 153–177, 2007.

[TZ09] J. Tang and X. Zhao. An enhanced opposition-based particle swarm optimization. In *2009 WRI Global Congress on Intelligent Systems*, volume 1, pages 149–153. IEEE, 2009.

[UKM+14] P. Upadhyay, R. Kar, D. Mandal, S. P. Ghoshal, and V. Mukherjee. A novel design method for optimal iir system identification using opposition based harmony search algorithm. *Journal of the Franklin Institute*, 351(5):2454–2488, 2014.

[VT07] M. Ventresca and H. R. Tizhoosh. Simulated annealing with opposite neighbors. In *Proceedings of the IEEE Symposium on Foundations of Computational Intelligence, FOCI 2007, part of the IEEE Symposium Series on Computational Intelligence 2007, Honolulu, Hawaii, USA, 1-5 April 2007*, pages 186–192. IEEE, 2007.

[VV05] M. Vasquez and Y. Vimont. Improved results on the 0-1 multidimensional knapsack problem. *European Journal of Operational Research*, 165(1):70–81, 2005.

[Wal72] D. L. Waltz. Generating semantic descriptions from drawings of scenes with shadows. Technical report, Cambridge, MA, USA, 1972.

[Wan15] J. Wang. Enhanced differential evolution with generalised opposition–based learning and orientation neighbourhood mining. *International Journal of Computing Science and Mathematics*, 6(1):49–58, 2015.

[Wat89] C. J. C. H. Watkins. *Learning from delayed rewards - Ph. D. dissertation.* PhD thesis, Psychology Department, University of Cambrigde, England, 1989.

[Wat10] Jean-Paul Watson. *An Introduction to Fitness Landscape Analysis and Cost Models for Local Search*, pages 599–623. Springer US, 2010.

[WLL+07] H. Wang, H. Li, Y. Liu, C. Li, and S. Zeng. Opposition-based particle swarm algorithm with cauchy mutation. In *Proceedings of the IEEE Congress on Evolutionary Computation, CEC 2007*, pages 4750–4756, 2007.

[WLWK15] Y. Wen, L. Liu, Z. Wang, and J. Kou. Multi-ucavs targets assignment using opposition-based genetic algorithm. In *The 27th Chinese Control and Decision Conference (2015 CCDC)*, pages 6026–6030. IEEE, 2015.

[WRDM96] D. Whitley, S. Rana, J. Dzubera, and K. E. Mathias. Evaluating evolutionary algorithms. *Artificial Intelligence*, 85(1):245 – 276, 1996.

[WWAS17] Y. Wang, A. Wang, Q. Ai, and H. Sun. A novel artificial bee colony optimization strategy-based extreme learning machine algorithm. *Progress in Artificial Intelligence*, 6(1):41–52, 2017.

[WWL+09] H. Wang, Z. Wu, Y. Liu, J. Wang, D. Jiang, and L. Chen. Space transformation search: a new evolutionary technique. In *Proceedings of the first ACM/SIGEVO Summit on Genetic and Evolutionary Computation*, pages 537–544. ACM, 2009.

[WWR+11] H. Wang, Z. Wu, S. Rahnamayan, Y. Liu, and M. Ventresca. Enhancing particle swarm optimization using generalized opposition-based learning. *Information Sciences*, 181(20):4699–4714, 2011.

[WWRK09] H. Wang, Z. Wu, S. Rahnamayan, and L. Kang. A Scalability Test for Accelerated DE Using Generalized Opposition-Based Learning. In *Ninth International Conference on Intelligent Systems Design and Applications, ISDA 2009, Pisa, Italy , November 30-December 2, 2009*, pages 1090–1095. IEEE Computer Society, 2009.

[XAL+14] W.l. Xiang, M.-q. An, Y.-z. Li, R.-c. He, and J.-f. Zhang. An improved global-best harmony search algorithm for faster optimization. *Expert Systems with Applications*, 41(13):5788–5803, 2014.

[XLM+17] W. Xiang, Y. Li, X. Meng, C. Zhang, and M. An. A grey artificial bee colony algorithm. *Applied Soft Computing*, 60:1–17, 2017.

[XWHW11] Q. Xu, L. Wang, B.M. He, and N. Wang. Modified opposition-based differential evolution for function optimization. *Journal of Computational Information Systems*, 7(5):1582–1591, 2011.

[XWW+14] Q. Xu, L. Wang, N. Wang, X. Hei, and L. Zhao. A review of opposition-based learning from 2005 to 2012. *Engineering Applications of Artificial Intelligence*, 29:1–12, 2014.

[YS15] S. Yazdani and J. Shanbehzadeh. Balanced cartesian genetic programming via migration and opposition-based learning: application to symbolic regression. *Genetic Programming and Evolvable Machines*, 16(2):133–150, 2015.

[YZNL17] K. Ye, C. Zhang, J. Ning, and X. Liu. Ant-colony algorithm with a strengthened negative-feedback mechanism for constraint-satisfaction problems. *Information Sciences*, 406:29–41, 2017.

[ZDT00] E. Zitzler, K. Deb, and L. Thiele. Comparison of multiobjective evolutionary algorithms: Empirical results. *Evolutionary computation*, 8(2):173–195, 2000.

[ZHD17] Y. Zhou, J. K. Hao, and B. Duval. Opposition-based memetic search for the maximum diversity problem. *IEEE Transactions on Evolutionary Computation*, 21(5):731–745, 2017.