PONTIFICIA UNIVERSIDAD CATÓLICA DE VALPARAÍSO
FACULTAD DE INGENIERÍA
ESCUELA DE INGENIERÍA INFORMÁTICA

MULTI-ARMED BANDIT FOR SELECTION OF BINARIZATION SCHEME IN METAHEURISTICS.

PABLO AGUSTÍN ÁBREGO CALDERÓN

INFORME FINAL DE TESIS
PARA OPTAR AL GRADO PROFESIONAL
MAGÍSTER EN INFORMÁTICA Y
TÍTULO DE INGENIERÍA CIVIL INFORMÁTICA

DICIEMBRE, 2022

Pontificia Universidad Católica de Valparaíso
Facultad de Ingeniería
Escuela de Ingeniería Informática

MULTI-ARMED BANDIT FOR SELECTION OF BINARIZATION SCHEMES IN METAHEURISTICS

PABLO AGUSTÍN ÁBREGO CALDERÓN

Profesor Guía: **Broderick Crawford Labrín**

Profesor Co-referente: **Eduardo Rodríguez Tello**

Carrera: **Ingeniería Civil Informática**
Grado: **Magíster en Ingeniería Informática**

Diciembre, 2022

Resumen

Multi-armed bandit (MAB) es un conocido algoritmo de aprendizaje por refuerzo que ha demostrado un gran rendimiento para los sistemas de recomendación y otras áreas. Los resultados presentados en diversas investigaciones nos motivan a trabajar con este algoritmo para el soporte en diferentes etapas dentro de los algoritmos de optimización. Por otra parte, los algoritmos metaheurísticos han ganado mucha popularidad debido a su gran rendimiento resolviendo problemas complejos, con espacios de búsqueda inabarcables. Black Widow Optimizer (BWO) es un novedoso algoritmo evolutivo, inspirado en el comportamiento de la araña viuda negra en su proceso reproductivo, que ha dado grandes resultados. Pendulum Search Algorithm (PSA) es una metaheurística de reciente creación inspirada en el movimiento armónico de un péndulo. Sus principales limitaciones son resolver problemas de optimización combinatorial, caracterizados por utilizar variables en el dominio discreto. Para superar esta limitación, se propone utilizar una técnica de binarización en dos pasos, que ofrece un gran número de combinaciones posibles que llamamos esquemas. Para ello utilizamos MAB como un algoritmo que aprende y recomienda un esquema de binarización durante la ejecución de las iteraciones (online). Con los experimentos realizados se demuestra que ofrece mejores resultados en la resolución del Problema de Cobertura de Conjuntos (SCP) que utilizando un esquema de binarización fijo.

Palabras clave: Multi-armed bandit - Reinforcement learning - Binarization schemes - Pendulum Search Algorithm - Black Widow Optimizer - Set Covering Problem

Abstract

Multi-armed bandit (MAB) is a well-known reinforcement learning algorithm that has shown great performance for recommendation systems and other areas. The results presented in various investigations motivate us to work with this algorithm for support at different stages within the optimization algorithms. On the other hand, metaheuristic algorithms have gained a lot of popularity due to their great performance solving complex problems, with endless search spaces. Black Widow Optimizer (BWO) is a novel evolutionary algorithm, inspired by the behavior of the black widow spider in its reproductive process, which has delivered great results. Pendulum Search Algorithm (PSA) is a recently created metaheuristic inspired by the harmonic motion of a pendulum. Its main limitations is to solve combinatorial optimization problems, characterized by using variables in the discrete domain. To overcome this limitation, it is propose to use a two-step binarization technique, which offers a large number of possible options that we call scheme. For this, we use MAB as an algorithm that learns and recommends a binarization schemes during the execution of the iterations (online). With the experiments carried out, it is shown that it delivers better results in solving the Set Covering Problem (SCP) than using a fixed binarization scheme.

Keywords: Multi-Armed Bandit - Reinforcement Learning - Binarization Schemes - Pendulum Search Algorithm - Black Widow Optimizer - Set Covering Problem

Contents

Lista de Tablas y Figuras

1 Introduction

Seeking better results is the main objective for many areas of the industry and has been a challenge since the beginning of human history [55]. For this reason, optimization research is so important, and has gained popularity in the last decades [50]. Operations research focuses its efforts on finding the best solution, which minimizes or maximizes a pre-established function. Each problem is mathematically modeled with a set of variables and different restrictions that allow limiting the possible values that these can take. In this way, a search field for possible solutions is generated, which in many cases, when the problem is very large, could take a machine years to go through completely [55]. In this situation, two major optimization methods arise: the exact methods and the approximate methods. The first one, make it possible to guarantee an optimal global value in the search field, that is, they go through all the possible solutions and find the best one, while the latter propose good solutions without having to make such an effort. We will focus on the second case.

In the approximate methods, there is a type of algorithm called heuristics, which manage to find good solutions in a reasonable time, without completely going through the search field [10]. If this algorithm is designed to solve a specific problem or an instance of it, we are talking about a specific heuristic algorithm, while if the algorithm is capable of solving various types of problems without the need to make major changes in its logical structure, we are talking about metaheuristic algorithms. The versatility offered by this second technique makes us opt for it in this thesis.

Metaheuristic algorithms can be classified according to the type of decisions that are made when exploring the search field. Based on this, deterministic metaheuristics arise, which from an initial state always reach the same final solution; and stochastic metaheuristics, which apply random rules in its process of searching for new solutions, making it very complex to arrive from an initial solution to the same final solution. This second type of metaheuristics generates problems when the domain of the problem variables are discrete, since random calculations normally return continuous values. That is why, to solve discrete problems, a technique is needed that allows us to take the result of the stochastic operator to the discrete domain, and more specifically, to the binary domain [13]. For this, there are various techniques which can directly affect the quality of the solutions found, making an algorithm more or less efficient. This is why the selection of the scheme to use is essential for the design of good algorithms, that maintain a good balance between exploration and exploitation, and thus avoid problems of premature convergence or exploration that does not reach optimal values.

Moreover, the development of machine learning in computer science has made great contributions to various areas of engineering, and operations research was no exception. The use of these techniques in various optimization processes have catapulted the performance of the previously described algorithms. That is why for this work, an another reinforcement learning algorithm will be implemented to improve the performance in the selection of binarization schemes of metaheuristic algorithms in order to obtain better results in the optimization of combinatorial problems [32]. For this, the Multi-Armed Bandit (MAB) algorithm will be developed, binarizing the Black Widow Optimizer (BWO) and Pendulum Search Algorithm (PSA). The proposed solution is assessed using the well-known Set Covering Problem (SCP) as a test case.

This document continues with an introduction of related work in section 2, then in section 3 the research methodology is explained, to continue with the presentation of the proposals in sections 4 and 5. Each of these sections present an explanation of the algorithm, the different experiments carried out and their results. To finish the work, section 6 presents a discussion and conclusion summarizing all the results obtained and proposing possible future work.

1.1 Objective

The main goal of this research is to improve the selection of binarization schemes of metaheuristic using a reinforcement learning technique, in order to obtain better results in the optimization of combinatorial problems.

1.2 Specific Objectives

- Implement the Multi-armed bandit algorithm.
- Implement a collection of binarization schemes to transform the continuos metaheuristic into a binary domain.
- Integrate the Multi-armed bandit algorithm with the metaheuristic to define the binarization scheme selection mechanism
- Improve the trade-off between exploration and exploitation of the search field.
- Analyze the performance of the proposed solution and compare the results with other binarization techniques.

2 Related works

In this section, the concept of metaheuristic with its different paradigms and variations will be described, to continue with an introduction of different techniques of machine learning (ML) and their possible applications on optimization algorithms.

2.1 Metaheuristics

An optimization problem can be defined with a series of mathematical elements in common: an objective function, which is sought to maximize or minimize; a set of decision variables, which will define the solution, and a set of constraints that limit the possible values of the variables and define the available search space [55]. Based on this, one of the criteria for classifying optimization problems is the domain in which their variables are defined, whether continuous, discrete, or a combination of them. Other criteria can be the presence of constraints, the number of objective functions, or whether they are static or dynamic [10]. In this work we will focus on the discrete domain, and more specifically, binary.

One of the first optimization problems is the well-known Traveling Salesman Problem (TSP), which consists of traveling through a certain number of cities in the shortest possible distance. To solve it, exact algorithms can be used, such as branch and bound [7, 48], which manages to find global optima for small instances, since the number of possible combinations, without knowing the distribution of the cities, is $n!$, so for $n = 100$ it would take longer than the lifetime of the universe, even using the most powerful computers in the world [55]. It is in this situation where more intelligent ways of finding good solutions in a reasonable time must be sought.

Metaheuristics are approximate optimization algorithms, that is, they deliver good solutions in reasonable times without having to go through the search field, and therefore, they do not ensure that each solution delivered is the best (global optimum) for the defined instance of the problem. [50]. For that reason, the main use of metaheuristic algorithms is to solve complex problems, such as NP-Hard. One of their main benefits, and one of the reasons why they have become so popular in the last few decades, is that they can be used to fix a large number of problems without making major changes to [10].

Metaheuristic algorithms can be classified into two groups, according to the number of solutions present at the beginning of their execution.

- **Single-solution based search:** Characterized by starting with a single solution, which is modified to find other solutions in the search space. Here stand out Simulated Annealing [53], Tabu Search [24], among others.
- **Population-based search:** They are characterized by starting with a set of solutions, called the initial population, and new solutions are sought based on the interaction between them. Four kinds of algorithms arise from here: swarm, which are inspired by the social behavior that occurs in the interaction between members of a community, such as particle swarms [43], Ant Colonies[19], Bee Colony[33], Gray Wolf [38], Blue Whale [37], among others; evolutionary techniques, inspired by Darwin's theory of evolution such as Genetic Algorithms [27]; Physics-based techniques, inspired by physical rules such as Black Hole [25]; and Human-related techniques such as Mine Blast Algorithm [46].

2.1.1 Evolutionary algorithms

One of the earliest search methods is evolutionary algorithms (EA), which use natural selection and Darwin's theory of evolution [16] as an abstraction of searching for new and better individuals. They are stochastic algorithms that, unlike the deterministic ones, for the same input, different outputs can be obtained. Within the family of population-based algorithms, evolutionary algorithms are the most popular among them [50]. In general terms, it is characterized by having the structure presented in the figure

From here arises the class of genetic algorithms [17], mainly associated with a binary representation, where some operators must be defined:

- **Selection**: Function that defines how the individuals of the population that will be used for the reproduction stage, will be selected. The most used method is the roulette method, where each individual is assigned a probability based on their fitness.
- **Crossover**: Function that defines how the parent individuals will be combined to generate their offspring. A type of crossover is performed by points, which selects n cutoff points where the solutions are separated and combined, maintaining the value of each variable. Another is arithmetic, which weights each parent variable and adds them together.

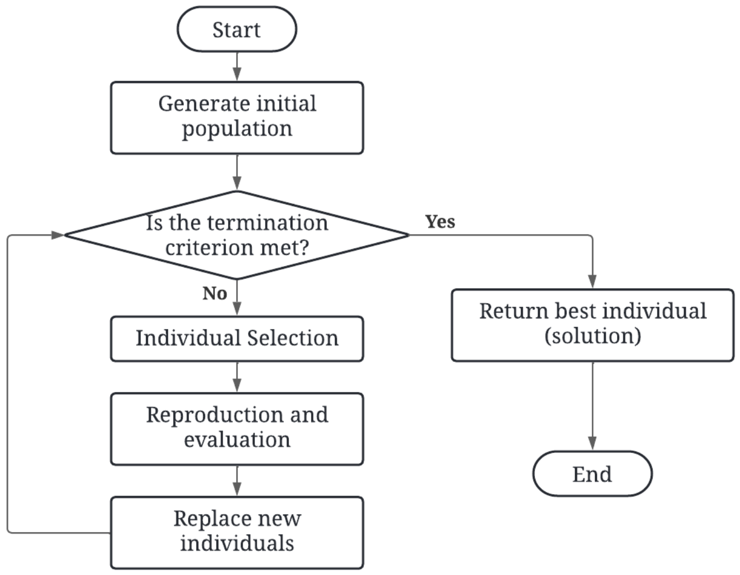

Start
Generate initial population
Is the termination criterion met?
Yes
No
Individual Selection
Reproduction and evaluation
Replace new individuals
Return best individual (solution)
End

Figure 1: EA Flowchart

- **Mutation**: This function defines how a solution is going to modify itself. It is characterized by making small changes in the individual in question. The most common is to select the variable to mutate and add a random number to it.

Depending on the mathematical operation used to select part of the population, to perform the crossover and the mutation, we can differentiate the different proposed algorithms [50]. In this work, the Black Widow Optimizer [26] algorithm will be used, which proposes a third step in the search for better solutions, called cannibalism, which allows filtering the members of the population to performing the other steps only with the best individuals.

2.1.2 Binarization techniques

In the crossover stage of reproduction, there is the challenge that the result of the operation is valid within the constraints of the problem, so there is concern when we want to maintain new binary variables ($0 \leq x \leq 1 | x \in \mathbb{Z}$). The problem arises when using an arithmetic crossover, which randomly weights each pair of parent variables, generating solutions with continuous variables. Crowford et. al. at [12] gather and describe a series of techniques that allow variables to be taken from the continuous domain to the discrete binary. In this thesis we will focus on the two-step technique, which consists of a first transfer phase, where the variables are taken to a continuous domain bounded between 0 and 1, and then move on to a second phase where each variable is binarized worth.

2.1.3 Transfer function

In [54] they present several ways to solve this problem applied to the PSO algorithm, which uses a function that transforms each variable in the solution into a value between 0 and 1 associated with the probability that said variable is equal to 0 or 1. The function is called transfer function, which Mirjalili and Lewis later formalize in [36] presenting two types of functions and different variants of each one of them. The names of each function came from the way in which these functions are graphed.

Name	Function
S1	$T(d_w^j) = \dfrac{1}{1 + e^{-2d_w^j}}$
S2	$T(d_w^j) = \dfrac{1}{1 + e^{-d_w^j}}$
S3	$T(d_w^j) = \dfrac{1}{1 + e^{\frac{-d_w^j}{2}}}$
S4	$T(d_w^j) = \dfrac{1}{1 + e^{\frac{-d_w^j}{3}}}$

Table 1: S-shape transfer functions

Name	**Function**
V1	$T(d_w^j) = \left\lvert erf\left(\frac{\sqrt{\pi}}{2} d_w^j\right)\right\rvert = \left\lvert \frac{\sqrt{2}}{\pi} \int_0^{\frac{\sqrt{\pi}}{2}x} e^{-t^2} dt \right\rvert$
V2	$T(d_w^j) = \left\lvert tanh\left(d_w^j\right)\right\rvert$
V3	$T(d_w^j) = \left\lvert \frac{d_w^j}{\sqrt{1+\left(d_w^i\right)^2}} \right\rvert$
V4	$T(d_w^j) = \left\lvert \frac{2}{\pi} arctan\left(\frac{\pi}{2} d_w^j\right)\right\rvert$

Table 2: V-shape transfer functions

2.1.4 Binarization function

Crawford et. al. at [12] they compile the various ways in which the literature binarizes continuous variables to solve combinatorial problems. In this way, they present 5 binarization functions that take the output of the transfer function $(0 \leq x \leq 1 | x \in \mathbb{R})$ and take it to a binary discrete domain $(0 \leq x \leq 1 | x \in \mathbb{Z})$.

Name	Function
D_1: Standard	$X_{new}^{j} = \begin{cases} 1 & if\ rand \leq T\left(d_w^j\right) \\ 0 & otherwise \end{cases}$
D_2: Complement	$X_{new}^{j} = \begin{cases} Complement\left(X_w^j\right) & if\ rand \leq T\left(d_w^j\right) \\ 0 & otherwise \end{cases}$
D_3: Static	$X_{new}^{j} = \begin{cases} 0 & if\ T\left(d_w^j\right) \leq \alpha \\ X_w^j & if\ \alpha < T\left(d_w^j\right) \leq \frac{1}{2}(1+\alpha) \\ 1 & if\ T\left(d_w^j\right) \geq \frac{1}{2}(1+\alpha) \end{cases}$
D_4: Elitist	$X_{new}^{j} = \begin{cases} X_{Best}^j & if\ rand < T\left(d_w^j\right) \\ 0 & otherwise \end{cases}$
D_5: Elitist Roulette	$X_{new}^{j} = \begin{cases} P[X_w^j = \delta^j] = \frac{f(\delta)}{\sum_{\delta \in X} f(\delta)} & if\ \alpha < T\left(d_w^j\right) \\ P[X_w^j = 0] = 1 & otherwise \end{cases}$

Table 3: Binarization Function

2.2 Exploration and exploitation

The biggest challenge that exists in the design and implementation of a metaheuristic algorithm is the balance that it obtains between its ability to explore and exploit the search space [40]. For this reason it is important to numerically parameterize these characteristics with some formula.

- **Exploration:** It is the ability of an algorithm to find solutions in areas where it has not been searched, far from the solutions already explored.

- **Exploitation:** Contrary to the previous one, it is the ability to intensify the search around a good solution, without escaping from the current search zone.

As you can see, in order to parameterize these characteristics, it is necessary to define a distance parameter between each solution or individual in the population. This distance is known as the diversity measure. In [40], the use of an equation called dimension-wise diversity is considered measurement, presented in equation (2) and used later in the calculation of the percentage of exploration and exploitation. More measures of diversity can be found in [11].

$$Div_j = \frac{1}{n}\sum_{i=1}^{n} |median(x^j) - x_i^j| \tag{1}$$

$$Div = \frac{1}{m}\sum_{j=1}^{m} Div_j \tag{2}$$

where $median(x^j)$ represents the median of dimension j of the entire population x, and x_i represents each individual of the population. n and m represents the size of the population and the dimension of the problem, respectively.

Then, having defined the way to calculate the distance between each individual, in [40] they present us a way to calculate the percentage of exploration and exploitation, present in the Eq. (3) and (4), in such a way that the measure of diversity remained independent, that is, that we can modify this formula without affecting the calculation of the percentage.

$$XPL\% = \left(\frac{Div}{Div_{max}}\right) \times 100 \tag{3}$$

$$XPT\% = \left(\frac{|Div - Div_{max}|}{Div_{max}}\right) \times 100 \tag{4}$$

where $XPL\%$ and $XPT\%$ represents the percentage of exploration and exploration, respectively, and Div_{max} is the maximum value of Div.

2.3 Black Widow Optimization

This metaheuristic is an evolutionary algorithm, which has had great results in applications in the field of engineering [3]. It is inspired by the peculiar reproduction of black widow spiders. This consists of a first stage of mating and then one of cannibalism between the female and the male, and between the new children. The algorithm will be described in detail below.

Figure 2 shows the life cycle or flowchart of the Black Widow (BWO) metaheuristic, as shown in [26].

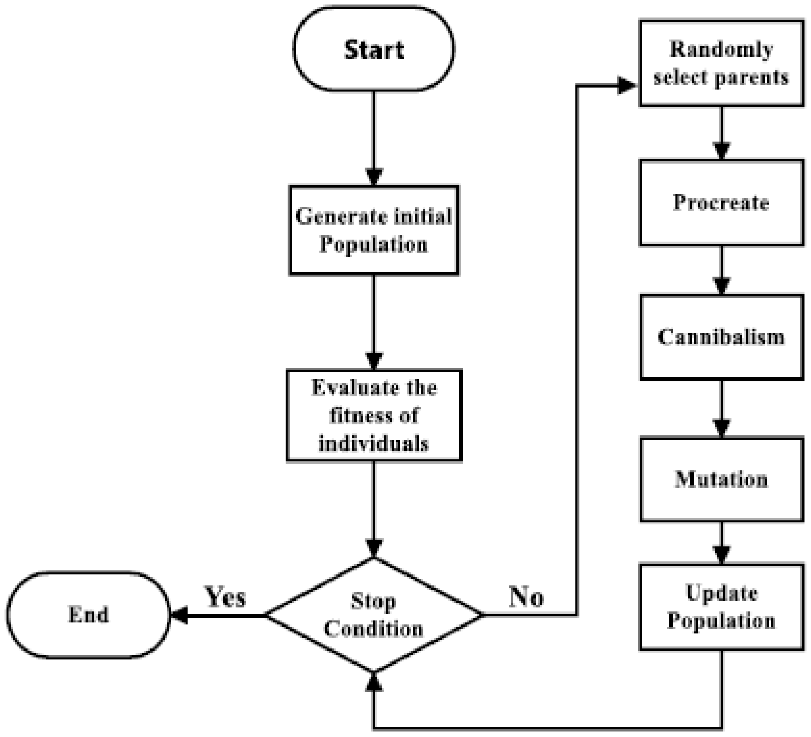


Figure 2: BWO Flowchart [26]

2.3.1 Initial population

For the generation of the initial population, it is necessary to define the values of the variables to form a suitable structure for the solution. Unlike other genetic algorithms, where each variable represents a gene, and each solution a chromosome, in BWO each variable represents a chromosome, and each solution a spider [26]. The defined structure will be called "Widow" and will be considered as a one-dimensional vector

$$Widow = [x_1, x_2, \cdots, x_{Nvar-1}, x_{Nvar}]$$

$$x_i \rightarrow Chromosome$$

To generate the initial population, its size and an array of size $Nvar$ are provided, which contains a tuple that defines the range in which the value of each variable can be generated.

Definition 1 *Let Bounds be a vector of size Nvar and Widow a spider from the initial population:*

$$Bounds = [(lb_1, ub_1), (lb_2, ub_2), \cdots, (lb_{Nvar}, ub_{Nvar})]$$

$$Widow = [x_1, x_2, \cdots, x_{Nvar-1}, x_{Nvar}]$$

such that:

$$lb_i \leq x_i \leq ub_i$$

If the Bounds vector is not specified, the following is assumed by default:

$$-1 \leq x_i \leq 1$$

To fulfill this condition, the value generated to the variable is defined as follows:

$$x_i = (rand) * (ub_i - lb_i) + lb_i$$

where $rand$ is a random value between 0 and 1 uniformly distributed, generating random values between lb and ub.

To know if the solution found is good compared to the others, the fitness of $Widow$ is calculated, which is obtained through an objective function that is defined for the problem.

$$Fitness(Aptitude) = f(widow) = f(x_1, x_2, \cdots, x_{Nvar})$$

2.3.2 Procreate

In the procreation stage, the aim is to obtain a new individual, a candidate to be a better solution, from the chromosomes of its parents. For this process we will define an operator $(\times)$ and a vector α, which will help us find the offspring.

Definition 2 *Let A and X be vectors of size n, we define the operations $\times$ and $\sim$, such that:*

$$A \times X = [a_1 x_1\,, a_2 x_2\,, \cdots, a_n x_n] \tag{5}$$

and

$$\widetilde{A} = [1 - a_1, 1 - a_2, \cdots, 1 - a_n] \tag{6}$$

We also define a vector α of size Nvar, where each value is random number between 0 and 1.

Now we use them for offspring generation, with the following equations.

$$Y_1 = \alpha \times X_1 + \widetilde{\alpha} \times X_2 \tag{7}$$

$$Y_2 = \alpha \times X_2 + \widetilde{\alpha} \times X_1 \tag{8}$$

where X_1 and X_2 are random selected parents, and Y_1 and Y_2, the two new sons. As can be seen, α defines the proportion that the chromosomes of each of the parents will have.

2.3.3 Cannibalism

The cannibalism stage of the metaheuristic consists of 2 phases, explained below.

- **Sexual cannibalism:** The female eats the male. The female and male can be recognized by their fitness value. The one with the best fitness survives.
- **Siblings cannibalism:** Strong spiders eat weak ones. A cannibalism classification (CR) is established, according to which the number of survivors is determined.

2.3.4 Mutation

In the mutation stage, a number of individuals is chosen according to the mutation rate, where each of the chosen solutions randomly exchanges two elements of the matrix, the figure 3 shows the exchange process among the chosen solutions.

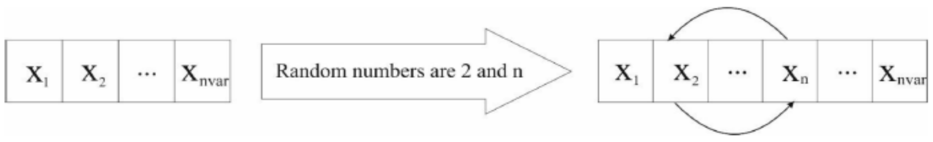


Figure 3: Mutation

2.3.5 Parameters

Some fundamental parameters for the use of the BWO metaheuristic are the following:

Ratio of procreation (PP): Determines how many individuals should participate in the procreation. By controlling the production of several offspring, it provides greater diversification and gives more opportunities to explore the search space more precisely.

Cannibalism rate (CR): Determine how many individuals will survive after the sibling cannibalism. Deletes individuals with worse fitness. This filter makes it possible to explore and exploit only the best individuals.

Mutation rate (PM): The appropriate value can guarantee the balance between exploration and exploitation. Determine how many individuals the mutation will be applied to (Fig. 3) It allows to control the transfer of search agents from the global to the local scenario and drive them towards the best solution.

The adjustment of these parameters could directly affect the performance of the algorithm.

2.4 Sine Cosine Algorithm

This metaheuristic is a physical-based algorithm which proposes to find better solutions using a mathematical model based on the sine and cosine functions, together with some parameters that will adjust the exploratory or exploitative behavior of the algorithm. Like BWO, Sine Cosine Algorithm (SCA) is a population algorithm designed to solve continuous optimization problems [35]. Each individual in the population represents a point on a circle, where the center is the best solution found, also called *destination*. According to the functions defined in the equations (9) - (11) each individual will move towards or away from the optimum.

$$X_{i,j}^{t+1} = X_{i,j}^{t} + r_1 \cdot \sin(r_2) \cdot |r_3 P_j^t - X_{i,j}^t| \tag{9}$$

$$X_{i,j}^{t+1} = X_{i,j}^{t} + r_1 \cdot \cos(r_2) \cdot |r_3 P_j^t - X_{i,j}^t| \tag{10}$$

$$X_{i,j}^{t+1} = \begin{cases} \textit{equation}\,(9) & ,\ r_4 < 0.5 \\ \textit{equation}\,(10) & ,\ r_4 \geq 0.5 \end{cases} \tag{11}$$

In the 3 equations, $X_{i,j}^t$ corresponds to the variable of dimension j of the individual i in the iteration t ($t+1$ in the following iteration), and P_j^t is the j-ith dimension of the best solution found up to the iteration t.

To incorporate the stochastic component into the equations, 4 random parameters are defined. The first one, r_1, indicates the direction in which the individual will move in the next iteration, r_2 indicates the amplitude that the movement will have, r_3 indicates the weight that the best solution will have and r_4 defines what equation will be used in each iteration. The equations are defined below:

$$r_1 = a - t\frac{a}{T} \tag{12}$$

$$r_2 = 2 \cdot \pi \cdot \textit{rand}\,[0,1] \tag{13}$$

$$r_3 = 2 \cdot \textit{rand}\,[0,1] \tag{14}$$

$$r_4 = \textit{rand}\,[0,1] \tag{15}$$

where a is a constant that we can adjust, t corresponds to the current iteration, while T corresponds to the maximum number of iterations that will be carried out in the execution of the algorithm.

2.5 Pendulum Search Algorithm

One of the problems that a metaheuristic can have is known as premature convergence. This problem consists of reaching an optimal value in very early iterations, known as local optimum, which makes it very difficult to find a better solution in subsequent iterations. Solving this problem is a challenge that is often faced with a small perturbation in the individuals of the population, for example the mutation in BWO. SCA uses the behavior of the sine and cosine functions to get closer to the optimum, along with the parameter r_1 that generates a linear downward movement over time [1]. Despite this, it has been seen that this characteristic of the algorithm does not solve premature convergence.

Pendulum Search Algorithm (PSA) tries to solve this problem thanks to the harmonic movement of a pendulum, which, unlike SCA, decides exponentially. This exponential function would enhance the exploration and exploitation balance [2]. It was recently introduced by Nor Azlina and Kamarulzaman [1] to solve continuous optimization problems.

The search agents are initialized randomly and their position is updated using equation 16.

$$X_{i,j}^t = X_{i,j}^t + pend_{i,j}^t \cdot (Best_j^t - X_{i,j}^t) \tag{16}$$

where $X_{i,j}^t$ is the position of the i-th solution in the j-th dimension in t-th iteration, $Best_j^t$ it is the position of the best solution in the $j-th$ dimension in the $t-th$ iteration and $pend_{i,j}^t$ is a parameter which is calculated as follows:

$$pend_{i,j}^t = 2 \cdot e^{(-t/tmax)} \cdot cos(2 \cdot \pi \cdot rand) \tag{17}$$

where t is the current iteration, $tmax$ is the maximum number of iterations and $rand$ is a random number between [0,1].

2.6 Machine learning

In [39] define Machine Learning (ML) as computational methods that use previous experience to improve the performance of some behavior or to make more precise predictions, understanding experience as information from the past that is made available to the learning agent. Some examples of problems that can be solved using ML are classification problems, regression, clustering, ranking, among others.

Depending on the way in which the machine obtains the information, the type of information that is obtained, and the way in which its learning is evaluated, different paradigms arise within ML [39], witch the following stand out.

- **Supervised learning**: learning is done with a first phase of training, where a supervisor gives the machine the input (features) and its respective expected output (labels), for later in the training, with only the input, the machine is capable to generate the corresponding label. It is mainly used for classification and regression problems.[39].
- **Unsupervised learning**: It is also carried out in a training phase and a testing phase, but in the first one the expected output is not delivered, but only with the input the machine is able to find patterns and group the delivered records according to their features.[47].
- **Reinforcement learning**: It is characterized by having its training and testing phase at the same time. Learning consists of the agent performing actions that generate a stimulus from its environment. This signal is known as a reward and the objective is to maximize its value with the following actions to be carried out. The information available to learn grows as learning progresses.[49]

Reinforcement learning algorithms are characterized by having a series of elements in common, which allow a better understanding of the entire learning process. They described below.[49]

- **Agent**: This element refers to the section of the algorithm that learns the desired behavior. It is the agent who performs an action and who receives external reward according to each action performed.
- **Environment**: It refers to what surrounds the agent and with whom he interacts. The state of the environment changes with each action performed by the agent, generating a signal perceived by it.
- **Reward**: Each action that the agent performs in its environment has an associated numerical reward, either positive or negative. This value is the one that the agent must learn to maximize throughout the execution of the algorithm.
- **Policy**: It is the brain of the agent. It is the one who decides what action the agent is going to perform in the next iteration. It makes the

decision based on the rewards it has received throughout the iterations carried out during the execution of the algorithm.

- **Value function**: It is the mathematical model that the policy uses to make the decision of what action to perform in each iteration.

2.7 Multi-armed bandit

This technique is a reinforcement learning algorithm, presented as an idea for the first time in 1952 by Herber Robbins in [44], where the design of statistical experiments where the size of the sample grows as the experiment is developed is explained .

The metaphor of this technique puts us in the situation of a casino, where we have a series of slot machines, of which the probability of success is unknown, nor is the reward that will be obtained in each game. In this way, we must test each of them until we know their behavior and use a strategy that allows us to maximize the gain at the end of the experiment. Another common objective in this kind of experiment is to minimize the difference between what is obtained and the maximum value expected (known as the regret value) [6, 42]. Based on this, the exploration - exploitation dilemma is presented, where we must negotiate between activating the arm that has the best rewards, and trying the others to find possible better rewards [5].

Depending on the value function to be used in the algorithm, the different variations that this technique will have are defined. [49].

Fig. 4 shows a graphical representation of the RL elements applied to MAB.

2.7.1 Epsilon greedy

This method uses a simple average as the value function and defines a variable ε to determine the probability that an arm is randomly selected in an iteration t.

We will define the estimated value of an action a at iteration t as $Q_t(a)$.

$$Q_t(a) \doteq \frac{\sum_{i=1}^{t-1} R_i \cdot \mathbb{1}_{A_i=a}}{\sum_{i=1}^{t-1} \mathbb{1}_{A_i=a}} \tag{18}$$

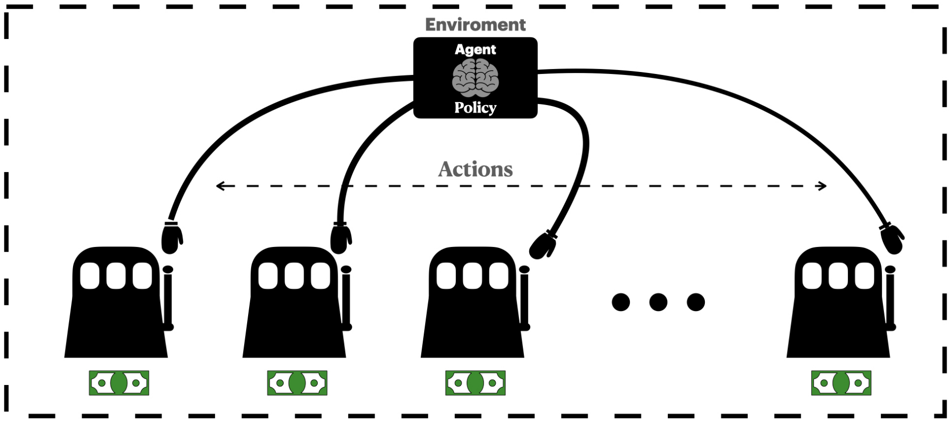


Figure 4: Multi-armed bandit

where R_i is the reward obtained in iteration i, A_i is the arm selected in iteration i and $\mathbb{1}_{predicate}$ defines a variable that will be one only if the *predicate* is true and 0 otherwise. If the denominator of the fraction equals zero, then a default value for $Q_t(a)$ will be selected, such as 0, or another value depending on where we want the optimal value to start. We also define the greedy action of the t iteration as:

$$A_t \doteq argmax_a Q_t(a) \tag{19}$$

where $argmax_a$ corresponds to the action a that maximizes the value of Q_t. The variable ε mentioned above is defined as:

$$\varepsilon = \mathbb{P}\big(A_t \doteq rand(0,k)\big) \tag{20}$$

where k corresponds to the number of arms available in the running instance. This variable allows us to explore the possible reward that the other arms deliver, but it does so in a static way, that is, without using previous information from the learning process.

2.7.2 Upper Confidence Bound

To improve the exploration process of the previous algorithm, this technique (UCB) solves the case in which the average rewards are equal between two arms, it will reward the one that has been triggered a smaller number of times, favoring the exploration of the instantiated arms. It does this by defining A_t as:

$$A_t \doteq argmax_a \left[Q_t(a) + c\sqrt{\frac{\ln t}{N_t(a)}} \right] \tag{21}$$

where $N_t(a)$ corresponds to the number of times arm a has been actuated up to time t (the denominator in Eq. (18)), $\ln t$ is the natural logarithm of t , and the constant $c > 0$ controls the degree of exploration that the algorithm will have. The higher the value, the more importance it will give to the factor that follows it.

As can be seen, this reinforcement learning technique is an optimization problem in itself, which seeks to maximize the reward obtained by activating the arms.

2.7.3 Dynamic Version

Understanding all of the above, the literature shows that there are cases where an arm can start to perform worse. With the models reviewed so far, it would take a long time for the algorithm to stop recommending the arm with the highest average reward [15]. It is also likely to take more iterations than defined in the run of the experiment, meaning that the arm with the best performance will never be found. For this situation, a dynamic version of the algorithm is presented, which allows restarting the learning algorithm when it finds some atypical behavior in some iteration.

This modification consists of performing a test in each iteration that consists of verifying if the given reward generates a standard deviation that is very different from the average deviation. Two values are defined for this.

$$avgDev_a(t) = \sum_{i=1}^{t} (r_{a,i} - \overline{r}_{a,i} + \delta) \tag{22}$$

$$maxDev_a(t) = max_a \{avgDev_a(i), i = 1...t\} \tag{23}$$

where $r_{a,i}$ corresponds to the reward obtained by activating arm a in iteration i, and $\overline{r}_{a,i}$ to the average reward of arm a accumulated up to iteration i. A delta is also defined as a deviation tolerance parameter. To know when an iteration returns an outlier, a λ parameter is defined that will allow us to approve or reject the test in question.

$$maxDev_a(t) - avgDev_a(t) > \lambda \tag{24}$$

If the equation 24 is true, then everything learned by the MAB[56] algorithm will be reset. This may sound like an extreme measure, but the literature shows that it is the fastest way to get back to learning [15].

3 Research Methodology

In the course of this work, several stages were passed, which will be described below.

During the first phase of this research (phase 0), a review of the literature and state of the art on machine learning and reinforcement learning algorithms is carried out, to propose Multi-Armed Bandit as a selector algorithm. Then optimization issues, metaheuristics, combinatorial problems, among others, were investigated to propose a final algorithm.

Moving on to the development phase (phase 1), each software component were implemented separately and independently and then assembled as if it were a puzzle. This allowed us to work with a cleaner code, granting more control when modifying each of the components and making it easier to experiment with different configurations.

In the experimentation phase, a trial and error logic was proposed, for which many adjustments were made based on the results obtained in each experimentation iteration. In the first place, it went through an assembly phase (phase 2), which consisted of grouping the components developed in phase 1 in order to put them to the test with the resolution of the Set Covering Problem. Then, in the evaluation phase, the results are obtained (phase 3), what is obtained is verified and the necessary changes are made in case they are not as expected or do not allow us to draw major conclusions. Figure 5 shows a flowchart that represents the step by step of the methodology used.

In the first iterations of the tweak and test cycle, few experiments were performed for each configuration, but enough to be able to make decisions about it, and make the corresponding changes. As the assembly of components was improved and a concise proposal was reached, the number of experiments increased in order to make more accurate decisions about the algorithm. From this logic, two large phases occurred, which contemplated changes in the metaheuristics and design of the experiments carried out, resulting in two software proposals with many differences between them. These proposals are explained in sections 4 and 5.

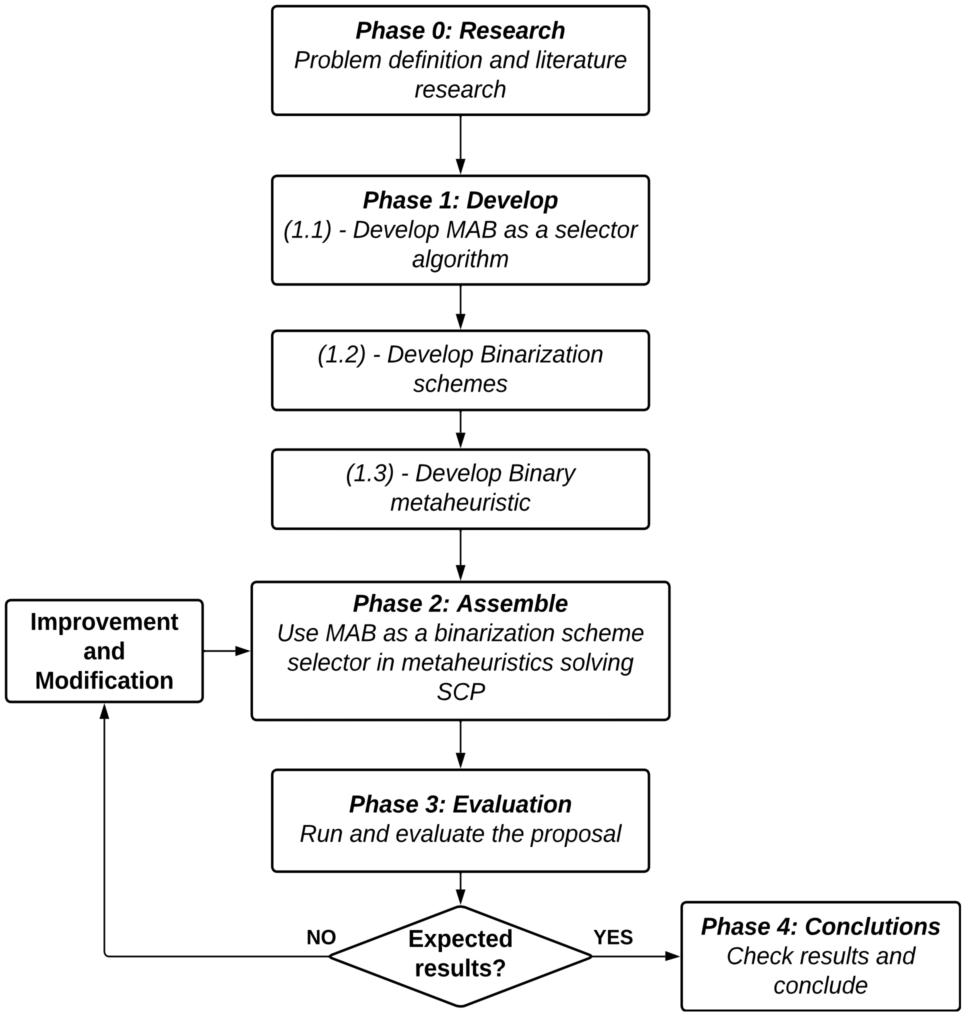

Phase 0: Research
Problem definition and literature research
Phase 1: Develop
(1.1) - Develop MAB as a selector algorithm
(1.2) - Develop Binarization schemes
(1.3) - Develop Binary metaheuristic
Improvement and Modification
Phase 2: Assemble
Use MAB as a binarization scheme selector in metaheuristics solving SCP
Phase 3: Evaluation
Run and evaluate the proposal
NO
Expected results?
YES
Phase 4: Conclutions
Check results and conclude

Figure 5: Research methodology diagram

4 First proposal

As mentioned in the research methodology, in the experimentation phase enough changes arose to present two different proposals. The first of these is explained below.

4.1 Set covering problem

As mentioned in the section 2.1, metaheuristic algorithms are used to solve NP-Hard problems [22], of which the well-known Set Covering Problem (SCP) [8] stands out. This consists of covering a set of areas, locating elements that are capable of covering the area where they are and their neighbors. Each of these elements has an associated cost, therefore all areas must be covered, obtaining the lowest cost. Figure 6 shows a small instance where, by placing the elements in nodes 1 and 4, all the areas are covered.

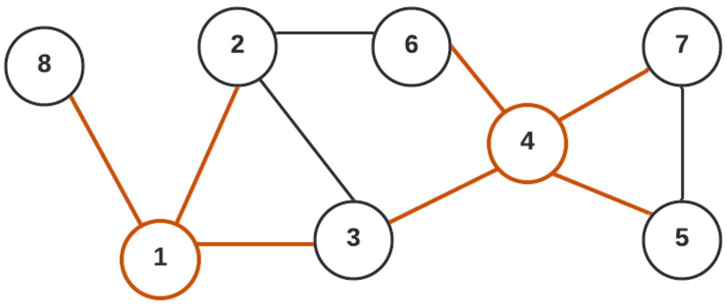


Figure 6: Example SCP instance

SCP can be modeled to solve applied problems such as aircraft-crew scheduling [4], truck routing [8], political districting [23], among others.

4.1.1 Mathematical model

Each instance can be modeled as a matrix of $m \times n$ where m is the number of zones to cover and n the elements located, from now on, constraints and columns respectively. In this way, we will have a binary matrix that will contain a 1 if a column satisfies a constraint. In addition to this, the instance includes a cost vector that indicates the cost value associated with selecting each column. In this way, the objective is to select the columns that exceed all the constraints, and that generate the lowest cost.

Definition 3 *Let $A = (a_{i,j})$ be a binary matrix of M rows ($i \in I = \{1, \ldots, M\}$) and N columns ($j \in J = \{1, \ldots, N\}$) such that:*

$$a_{i,j} = \begin{cases} 1, & if\ row\ i\ can\ be\ covered\ by\ column\ j \\ 0, & otherwise \end{cases} \tag{25}$$

and $C = (c_j)$ a uni-dimensional vector that contains the cost of each column.

Definition 4 *Let $X = (x_j)$ ($j \in J$) a uni-dimensional vector that contain each decision variable.*

$$x_j = \begin{cases} 1, & if\ column\ j\ is\ selected \\ 0, & otherwise \end{cases} \tag{26}$$

In this way, the best combination of selected columns must be sought to minimize the cost function. It is presented as follows.

$$\min_{Z \in \mathbb{R}} \quad Z = \sum_{j=1}^{N} c_j \cdot x_j \tag{27}$$

This must be done in compliance with the restriction that all rows are covered and that the decision variables are binary.

$$\begin{aligned} \text{s.t.} \quad & \sum_{j=1}^{N} a_{i,j} \cdot x_j \geq 1 & \forall\ i \in I \\ & x_j \in \{0, 1\} & \forall\ j \in J \end{aligned} \tag{28}$$

4.1.2 Repair of infeasible solutions

Throughout the process of searching for new solutions, it is likely that these solutions do not meet all the constraints posed by the instance. In that case, a repair process must be resorted to.

For this, the so-called information heuristic is used, which consists of a function that tells us how strong a solution is [51]. In this case, the function is defined as the trade-off between the cost of a column and the number of constraints it fixes. Let e_j be the number of new constraints that are satisfied by adding column j to a solution. With this, our heuristic information h_j is defined as $h_j = c_j/e_j$ [18]. With this value, when fixing a solution, the column j with the lowest value h_j is found and added to the solution.

4.2 Binary Black Widow Optimization

The BWO algorithm, since its publication, has been successful in its applications in production and distribution optimization problems [21], deep convolution neural network [41], among others [34, 28, 29]. Despite its success, its original model presents difficulties in solving combinatorial optimization problems, precisely because of the crossover model it proposes for its procreation stage, described in the equations(7) and (8), section 2.3.2.

4.2.1 Cross-over proposal

Going into this problem in detail, we are presented with 2 cases: when the i variables of each parent are equal (1-1 or 0-0), and when they are different (1-0 or 0- 1). In the first case there is no problem, since regardless of the value $alpha_i$ the result will remain intact, that is, the i-th variable of the new child will be identical to that of its parents. The second case is where the problems arise, since being different, the values α_i and $1 - \alpha_i$ will be assigned to the new children, removing the variables from the discrete domain. Figure 7 shows an example of this problem.

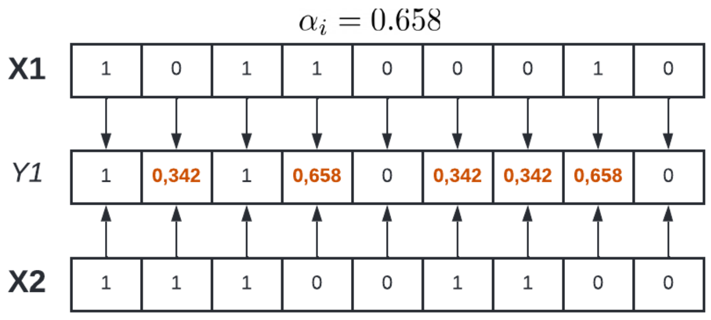


Figure 7: Arithmetic Crossover Example

To solve this problem, 2 proposals are presented, used for feature selection [3], and distribution problems [21]. In [21] it is proposed to modify the operation used for procreation (arithmetic cross-over), and use an order-based crossover approach [52], which consists of selecting some variables from the father and others from the mother to generate the new child, as shown in figure 8. In [3] instead, it is proposed to add an extra step to the generation of new solutions, which is the two-step binarization explained in the section

2.1.2. First, apply the transfer function S2, corresponding to the sigmoid function, and then apply the standard binarization.

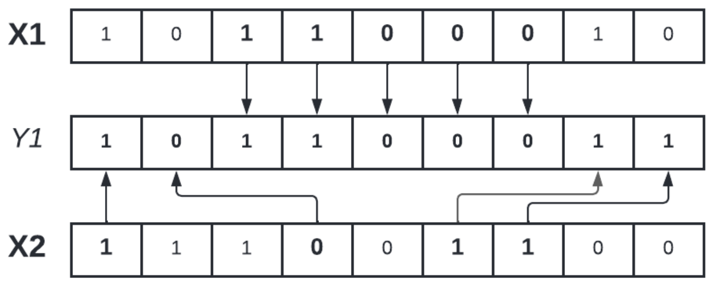


Figure 8: Order-based Crossover Example

Moreover, the problem of generating new solutions in algorithms that solve combinatorial optimization problems is the generation of infeasible solutions, that is, solutions that do not meet the intrinsic constraints of the problem. This is why each generated solution must be reviewed before adding it to the population. When this problem occurs, the repair of said infeasible solution must be carried out.

As can be seen in the example presented in figure 7, regardless of the value of the random variable *alpha*, the resulting value of the operation will never go outside the range [0,1]. For this reason, the transfer functions presented in the section 2.1.2 would not be necessary to be able to binarize with the binarization functions (section 2.1.2).

4.2.2 Mutation proposal

In the original article [26], a parameter is used in the mutation stage to define how much population will be mutated. Taking into account that the cross-over explained in the previous section modifies the variables only in certain situations, and leaves new children equal to the parents in the event that these individuals are exactly the same, the mutation process plays an important role in the search for new solutions. That is why in this thesis another parameter is proposed that tells us the number of variables that will be mutated. These randomly selected variables will be swapped for other variables having the opposite value. For example, if a one is selected, it can only be changed to a variable with a value of zero. Otherwise, the mutation would not produce any change. The figure shows an example of the operation performed.

4.2.3 Population proposal

As explained in the original paper of the algorithm, according to the *pp* parameter, the population of individuals with whom the reproduction will be carried out, called *pop*1, is optimized. With it, the corresponding cross-over is carried out, generating, for each pair of parents, as many children as the dimensions of the problem. This creates serious time optimization difficulties when trying to solve the Set Covering Problem. That is why in this thesis it is proposed to generate *npop* children (population size), for each pair of parents. With this change the times are reduced considerably, taking into account that the smallest instance of OR-Library has 1000 variables.

As for the population resulting from the reproduction and cannibalism stage (*pop*2), it is proposed to store the surviving mother (father) and surviving children, and as for the mutation, it is proposed to use the original population to reproduce *pop*1. In this way, in each iteration the following result will be obtained:

- The best parents of the inner iteration (surviving mother)
- Best new guys (surviving sons)
- A small modification of the best parents from the previous iteration.

Next, in figure 10 the flowchart of the BBWO algorithm to be used will be shown.

4.3 Dynamic Multi-armed Bandit for binarization schemes selection

To improve the proposal presented in [3], the Dynamic Multi-armed Bandit (DMAB) algorithm will be used, which will learn to select different binarization schemes to use in the BBWO metaheuristic, in order to obtain a good balance between exploration and exploitation of the search field.

In [31], the importance of binarization schemes for the search result, solving SCP, is analyzed. The authors conclude that it directly affects and give recommendations of schemes to use in certain instances.

As the selected scheme is so important, in this thesis it is proposed to use a reinforced learning algorithm, specifically Multi-armed Bandit, recognized for its use in recommendation systems [20], to implement a dynamic binarization model in the metaheuristic BBWO, solving SCP.

An important factor for the success of any reinforcement learning algorithm is the modeling of the reward that will be obtained with each action performed. For this investigation, the way in which the reward will be calculated is with its percentage of fitness improvement delivered by the metaheuristic, expressed as follows.

$$r = 100 \cdot \left[\frac{f_{old} - f_{new}}{f_{old}} \right] \tag{29}$$

where f_{old} is the best fitness of the last iteration, and f_{new} is the best fitness of the current iteration.

As mentioned in the 2.7 section, there are many ways to choose which arm to use in each iteration, but based on the literature [56, 45, 9, 15], one that has given good results is the UCB1 call described in the section 2.7.2. Also, as an exercise to compare the performance of UCB1, the random value functions will be developed, which consists of always choosing an arm at random, and the $\varepsilon - greedy$ function explained in the section 2.7.1 . In addition to the UCB1 value function, the PH-test is presented, which allows restarting the learning of the agent when it begins to have atypical behavior. The implementation of this test would accelerate learning in those cases.

In this way, as a summary, the Dynamic Multi-armed bandit model to use is as follows:

- **Environment**: As mentioned in section 4.2.1, the binarization schemes will be complained just by the binarization function, giving us a total of 5 possible ways to binarize our continues variables. For this reason the arms to be selected in each iteration will be just 5.

- **Policy**: The policy will be governed by the value function UCB1, added to the PH-test. The random and $\varepsilon - greedy$ functions will be implemented only for preliminary experiments and future comparisons.

- **Reward**: The reward will be defined by the fitness improvement percentage of each execution of the BBWO algorithm.

Figure 9 shows the Dynamic Multi-armed Bandit (DMAB) model.

4.3.1 DMAB Flowchart

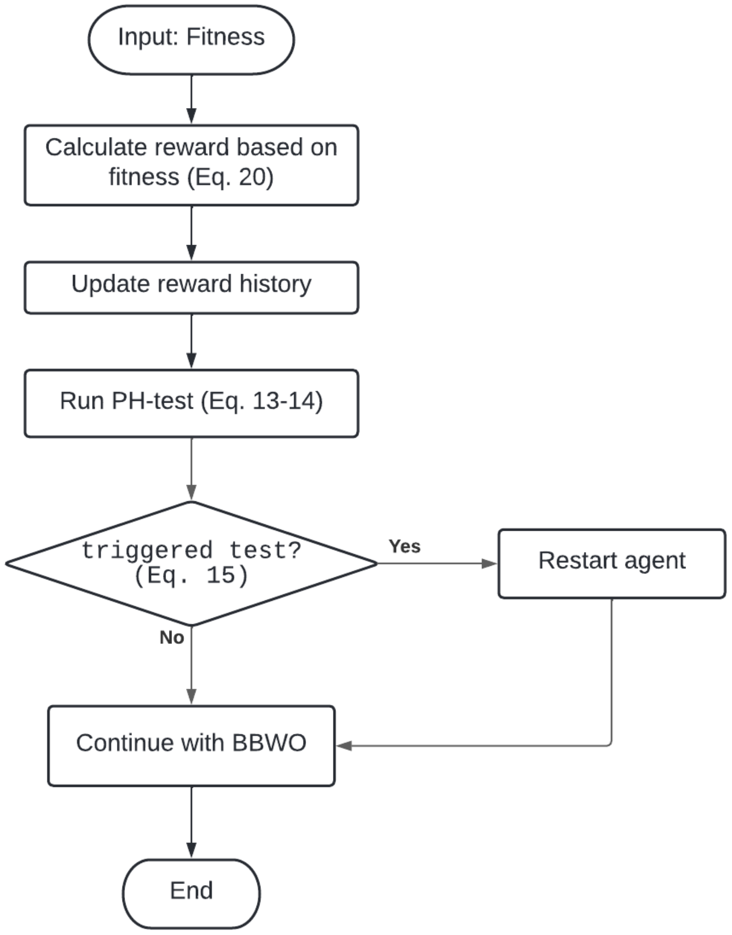


Figure 9: DMAB flowchart

The step-by-step explanation of the algorithm is presented below.

1. Based on the fitness of the best solution found in the execution and the best historical fitness, the reward generated by selecting said scheme was calculated, using the equation (29).
2. The average reward of the selected scheme is updated.
3. Based on the reward obtained, the PH-test is executed using the equations (22) and (23).

4. If the test is triggered (Eq. 24), the agent's training is restarted and the environment is initialized again. Otherwise, the iterations continue.

4.3.2 BBWO+DMAB Flowchart

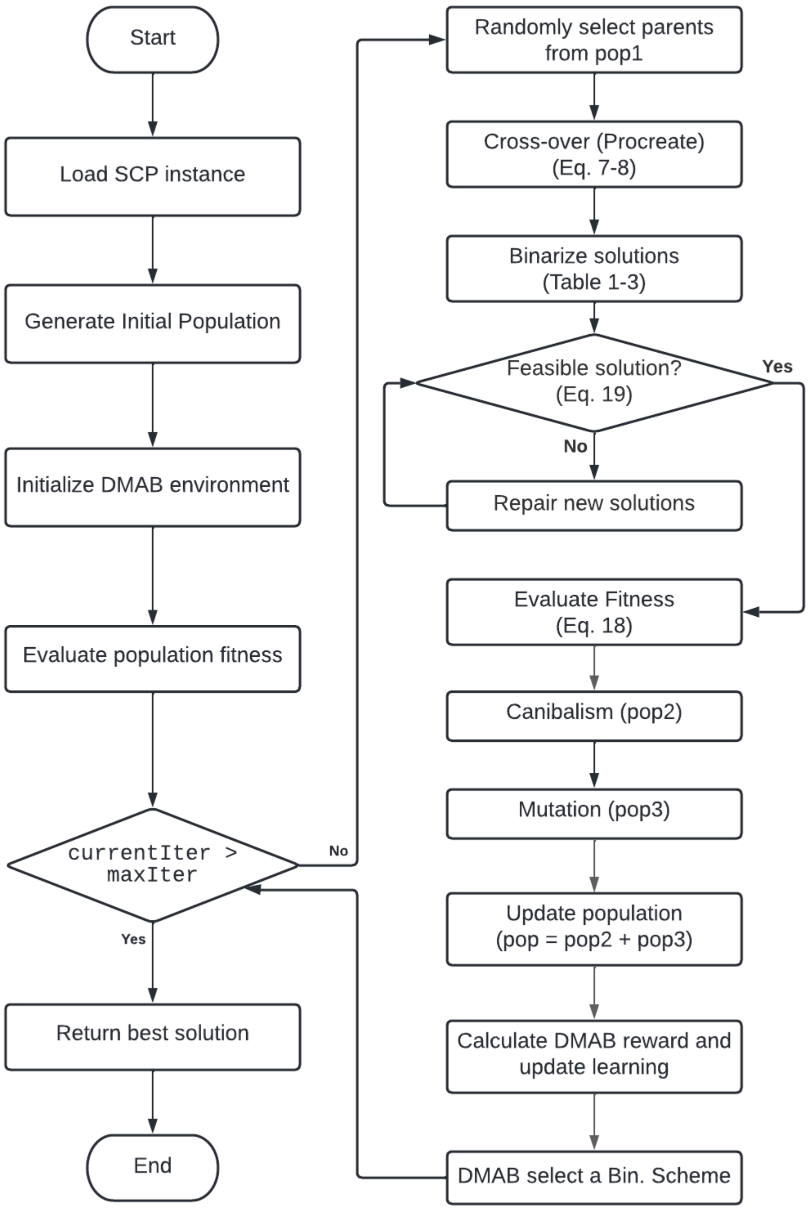

Start
Load SCP instance
Generate Initial Population
Initialize DMAB environment
Evaluate population fitness
currentIter > maxIter
No
Yes
Return best solution
End
Randomly select parents from pop1
Cross-over (Procreate) (Eq. 7-8)
Binarize solutions (Table 1-3)
Feasible solution? (Eq. 19)
Yes
No
Repair new solutions
Evaluate Fitness (Eq. 18)
Canibalism (pop2)
Mutation (pop3)
Update population (pop = pop2 + pop3)
Calculate DMAB reward and update learning
DMAB select a Bin. Scheme

Figure 10: BBWO flowchart

The step-by-step explanation of the algorithm is presented below.

1. The instance to use is loaded, brought from OR-Library
2. The initial population is generated, ensuring that each individual complies with the constraints of the problem.
3. The environment composed of the binarization functions (arms) is initialized. A priori we have 5 of them, but they can vary depending on what is decided in the experiment stage.
4. Each solution is evaluated and the population is sorted based on the fitness of each individual.
5. As long as the iterations are less than the maximum, new solutions will be searched.
6. Based on the procreation ratio (PP), the best individuals are selected to start procreation between each pair.
7. For each pair of parents, *npop* new individuals are generated, using the crossover equations (7-8)
8. Binarization is applied.
9. The feasibility of the new solutions generated is reviewed.
10. For each pair of parents, the one with worse fitness is discarded.
11. Based on the cannibalism ratio (CR), the new offspring with the worst fitness are discarded. Survivor sons and mother are stored in *pop*2.
12. Based on the mutation ratio (PM), the individuals to carry out the mutation are selected from *pop*1. The result is stored in *pop*3.
13. The feasibility of the new solutions generated is reviewed.
14. The population *pop* is updated by adding the 2 resulting populations $pop2 + pop3$. To maintain the initial population size, only the best *npop* individuals will be kept.
15. The fitness of the population is calculated.
16. DMAB, explained in section 4.3.1 is executed
17. DMAB selects the binarization scheme to use, based on the selected value function (Eq. 18-21).
18. At the end of the search for solutions, we return the best one found.

4.4 Proposed pseudo-code

A summary of the parameters to be used will be presented first.:

- Procreation ratio *pp*: Defines with how much population the reproduction will be carried out.
- Cannibalism ratio *cr*: Define how many children will survive after reproduction.
- Mutation rate *pm*: Define how many individuals will be mutated.
- Mutated variables *vm*: Define how many variables will be exchanged in each individual.
- Population size *npop*: This size will be maintained throughout the algorithm.
- itareations $maxIter$: iterations to execute.
- ε: Probability that the agent chooses a random arm, in the policy ε-greedy.
- c: UCB-1 policy scan parameter. Gives more weight to arms with fewer executions.
- λ: Adjust how rigorous the PH-Test will be.

Algorithm 1 Binary Black Widow Optimizer

```
Input: Population $X = \{x_1, x_2, ..., x_{npop}\}$, params $\{pp, cr, pm, maxIter\}$
Output: Updated population $X' = \{x'_1, x'_2, ..., x'_{npop}\}$ and $X_{best}$
 1: Set procreation number $Pn \rightarrow npop * pp$
 2: Set mutation number $Mn \rightarrow npop * pm$
 3: Set survivors number $Sn \rightarrow npop * cr$
 4: Initialize population $pop$
 5: Binarize $pop \rightarrow bpop$
 6: for $t$ to $maxIter$ do
 7:     Sort the population according to their fitness
 8:     Update best solution $X_{best}$
 9:     Select best $Pn$ individuals and set at $pop1$
10:     for $i = 1$ to $Pn$ do
11:         Procreation (Algorithm 2)
12:         Sexual cannibalism: Delete parent with worst fitness from $pop1$
13:         Apply binarization scheme selected by DMAB (Table 3)
14:         Repair solutions if the are not feasible. [Section (4.1.2)]
15:         Siblings cannibalism: Keep best $Sn$ new individuals.
16:         $pop2$ = best parent + best children
17:     end for
18:     for $i = 1$ to $Mn$ do
19:         Mutation (Algorithm 3)
20:     end for
21:     $pop = pop2 + pop3$: Keep best $npop$ individuals from $pop$
22:     Calculate fitness
23:     Update de best solution $X_{best}$
24:     Update DMAB (Algorithm 4)
25: end for
```

Algorithm 2 Black Widow Procreation

```
    Input: $bpop1$ $X = \{x_1, x_2, ..., x_{Pn}\}$
    Output: New individuals.
1: Select two random parents from $bpop1$
2: for $j$ to $npop/2$ do
3:     Generate random value $\alpha$
4:     Generate 2 new individuals with Eq. (7) and (8)
5:     Save new individuals
6: end for
```

Algorithm 3 Black Widow Mutation

```
    Input: $bpop1$ $X = \{x_1, x_2, ..., x_{Pn}\}$, $vm$
    Output: Mutated individuals at $pop3$.
1: Calculate number of variables to swap $vm * dimension \rightarrow vars$
2: Select random individual from $bpop1 \rightarrow bp$
3: Swap $vars$ random variables from $bp$
4: Repair $bp$, if the are not feasible
5: Add mutated individual to $bpop3$
```

4.5 Design of the experiments

In a first stage, it was decided to experiment only with the binary version of Black Widow Optimization, using a fixed binarization function in each execution, to later compare its behavior with those obtained by executing the algorithm together with DMAB, offering dynamic binarization.

The firsts tests were carried out to refine and define the design of the algorithm, its details of operations and configurations. After deciding on the design, a preliminary parameter adjustment was carried out, with few and small experiments, to verify its best configuration. The parameters that remained fixed in these executions were those described in the table 4. The analysis of these values was as follows. First of all, *pp* had to be high, since all the new individuals generated for the next iteration will be obtained from that percentage of the population, either by cross-over or mutation. Second, *cr* was adjusted considering the possible exploration of the algorithm: being very low, the following iterations would search for values very similar to the best ones (exploitation). Lastly, as mentioned above, in advanced iterations the crossover operation stops having as much impact as the individuals in

Algorithm 4 Dynamic Multi-armed Bandit

```
    Input: Environment (Bin. functions) $BS = \{bs_1, bs_2, ..., bs_{40}\}$ and new fitness.
1: Generate reward with Eq. (29) from fitness (BBWO output)
2: Update reward mean history
3: Run PH-test with Eq. (22) and (23)
4: if Eq. (24) is True then
5:     Restart agent learning. Start from first line
6: else
7:     Continue
8: end if
```

the population are all the same or very similar. This is why the values *pm* and *vm* are adjusted to high values in order to be able to search for new solutions when the cross-over stops perturbing the population.

npop	pp	cr	pm	vm	max iter
50	0.8	0.5	0.8	0.1	1000

Table 4: Static parameters

Parallel to these experiments, the algorithm was tested with the dynamic binarization technique, where the table parameters 4 and the instances were kept, but the policy used in the DMAB algorithm was varied. First, the random policy, which selects an arm randomly; second, the ε-greedy policy (section 2.7.1) using $\varepsilon = 0.2$ as the value; and finally UCB1 (section 2.7.2) using $c = 2$ as parameter.

The SCP instances were selected from the OR-Library, which contains a large number of instances with different characteristics and complexities. The table 5 shows a detailed description of the different families of instances they have. The column m corresponds to the number of restrictions that each instance has, n corresponds to the dimension of the problem, that is, the number of variables that the solutions have. Cost range refers to the interval of possible values that the cost of each variable can have. The density percentage corresponds to the amount of 1 that the coverage matrix has, explained in the equation (25). Finally, the "Optimal Solution" column tells us if the optimal solution has already been found for each of the instances.

Instance set	m	n	Cost range	Density(%)	Optimal solution
4	200	1000	[1,100]	2	Known
5	200	2000	[1,100]	2	Known
6	200	1000	[1,100]	5	Known
A	300	3000	[1,100]	2	Known
B	300	3000	[1,100]	5	Known
C	400	4000	[1,100]	2	Known
D	400	4000	[1,100]	5	Known
NRE	500	5000	[1,100]	10	Known
NRF	500	5000	[1,100]	20	Known
NRG	1000	10000	[1,100]	2	Unknown
NRH	1000	10000	[1,100]	5	Unknown

Table 5: Beasley's OR-library benchmark description. [30]

For the execution of these experiments, 3 different machines with different characteristics were used. For this reason, an analysis of the execution times obtained by each configuration could not be carried out. The machines were as follows:

- Apple Macbook Air: M1 processor 7-core CPU, 8-core GPU and 8GB of RAM.
- AWS server: t2.micro EC2 instance (1 core CPU 1GB of RAM).
- DigitalOcean Server: Droplets (1 core CPU 1GB of RAM).

4.6 Preliminary Results

For the BBWO analysis with fixed binarization, running each combination 4 times, the following results were obtained, presented in the table 6.

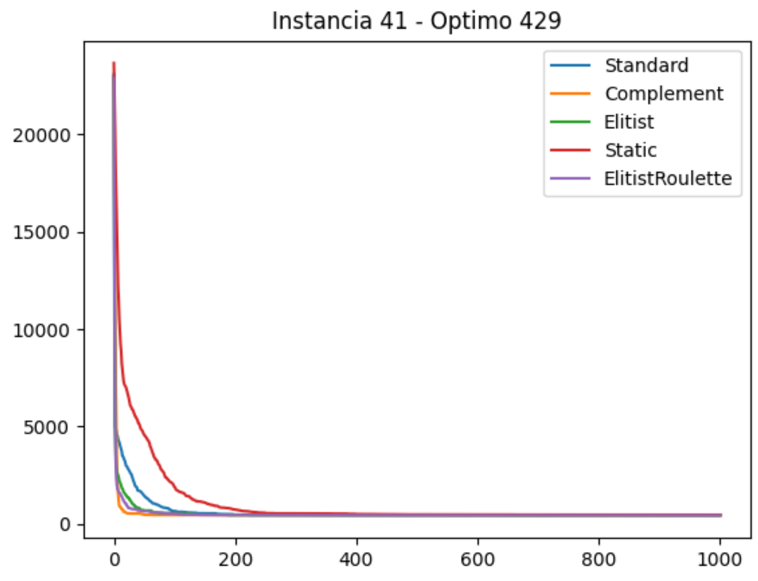
As can be seen, there is a clear better performance in the Elitis and Elitist Roulette functions, over the others, with the exception of the Complement function, which equals in instance 53 and wins in instance 41. In any case, there were few executions to be able to classify a function as the best.

The convergence of each of the instances 4.1 and 5.3 can be seen in figure 11, and instances 6.2 and b.1 can be seen in figure 12. These results are related to those present in the table 6, generating a possible correlation

Instancia	Optimo	BF	Max	Min	Mean
41	429	**Complement**	453.0	444.0	**449.75**
		Elitist	454.0	449.0	452.25
		Elitist Roulette	477.0	448.0	457.75
		Standard	485.0	455.0	470.40
		Static	475.0	466.0	470.25
53	226	**Complement**	254.0	252.0	**253.50**
		Elitist	260.0	246.0	**253.50**
		Elitist Roulette	265.0	248.0	260.25
		Standard	280.0	256.0	266.50
		Static	294.0	256.0	282.75
62	146	Complement	165.0	161.0	163.25
		Elitist	174.0	167.0	171.25
		Elitist Roulette	176.0	155.0	**161.25**
		Standard	186.0	165.0	177.00
		Static	187.0	164.0	175.50
b1	69	Complement	92.0	89.0	90.25
		Elitist	92.0	83.0	86.75
		Elitist Roulette	92.0	79.0	**85.75**
		Standard	131.0	105.0	122.25
		Static	350.0	208.0	280.00

Table 6: BBWO results with fixed binarization

between the speed of convergence and the best solution found. The Static and Standard functions had the most difficulty finding an optimal value.



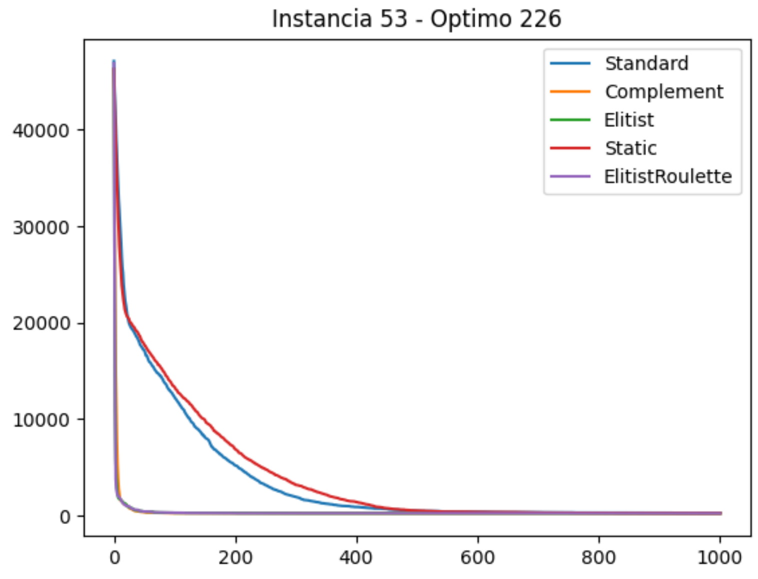
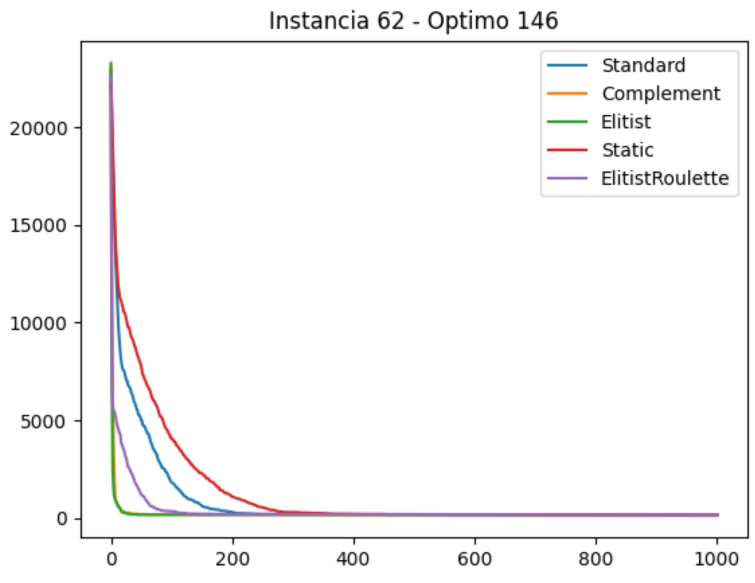


Figure 11: Convergence of 4.1 and 5.3 instances by binarization function



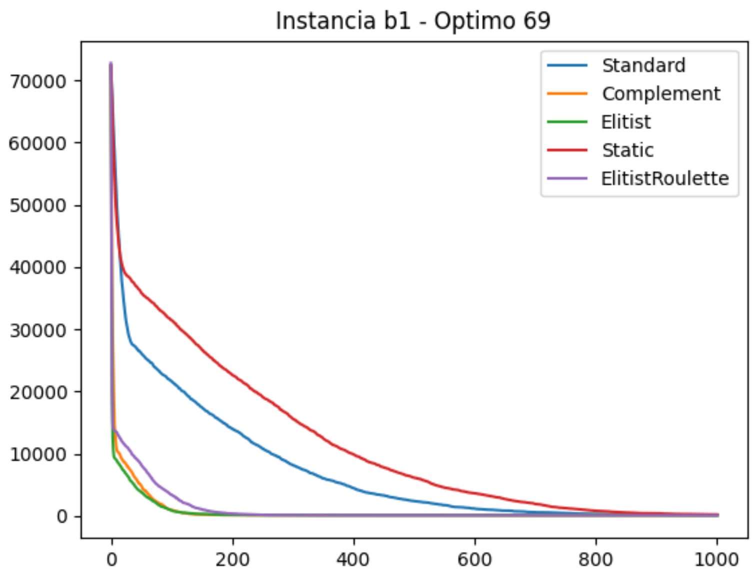


Figure 12: Convergence of 6.2 and b.1 instances by binarization function

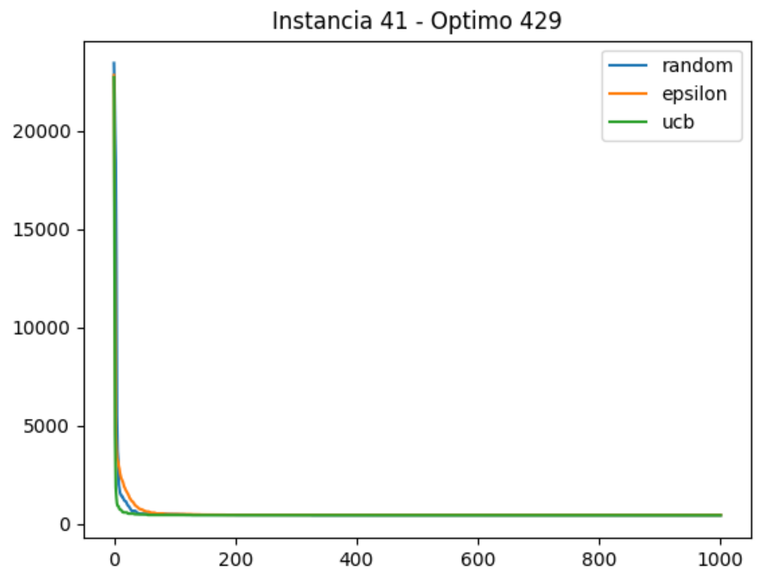
4.6.1 DMAB-BBWO results

For the following results, 5 executions were performed for each policy. Looking at the table 7, we can see that, with the executions carried out, there is not a great relationship between the best results and a specific policy. If we analyze the convergence that exists in each of the executed instances, there is also no constant behavior in any of the policies used (Fig 13 - 14).

Instance	Optimal	Policy	Max	Min	Mean
41	429	ε-greedy	464.0	447.0	454.00
		Random	457.0	448.0	452.80
		UCB-1	453.0	445.0	**450.75**
53	226	ε-greedy	286.0	247.0	260.00
		Random	263.0	242.0	**254.20**
		UCB-1	264.0	252.0	256.75
62	146	**ε-greedy**	169.0	158.0	**163.75**
		Random	175.0	160.0	164.80
		UCB-1	168.0	162.0	164.50
b1	69	**ε-greedy**	95.0	80.0	**88.00**
		Random	93.0	89.0	90.00
		UCB-1	96.0	82.0	90.25

Table 7: BBWO results with fixed binarization

On the other hand, reviewing the figures 15, 16 and 17, we will be able to analyze the behavior of each policy in question. They show the average number of times each binarization function was selected. The Random policy had an expected behavior, giving each arm a similar number of executions. In the case of the ε-greedy policy, it can be seen how it rewards the Elitist and Elitist Roulette functions more, which makes sense with the results obtained in the execution of BBWO with fixed binarization (Table 6). Finally, reviewing the UCB-1 policy, it is interesting to note the same as in the previous policy, adding to the Complement function, but it is curious that they are selected a similar number of times. This is due to the fact that the fitness improvement percentage (reward) is so low in the middle and final iterations, that the exploratory factor of the policy begins to be taken more into account, which favors the functions that have been selected less number of times. This is why in the first iterations Elitist and Elitist Roulette were selected and then they were equalized when the reward obtained decreased.



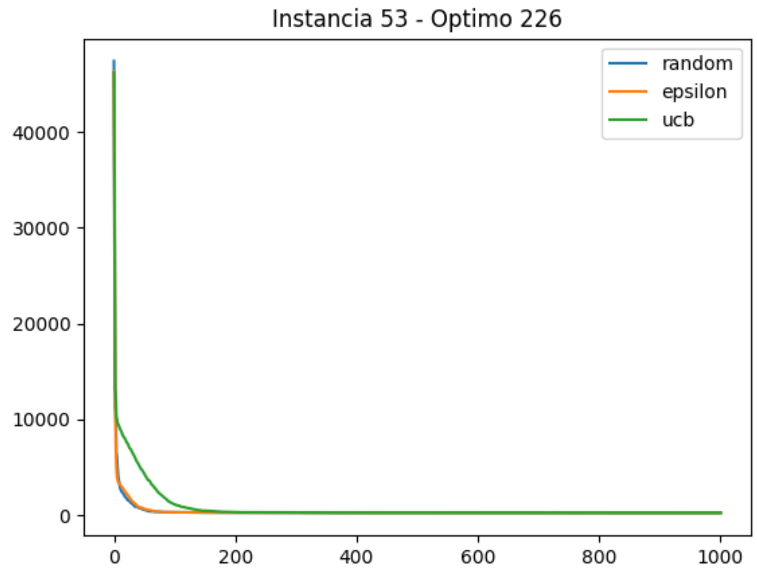
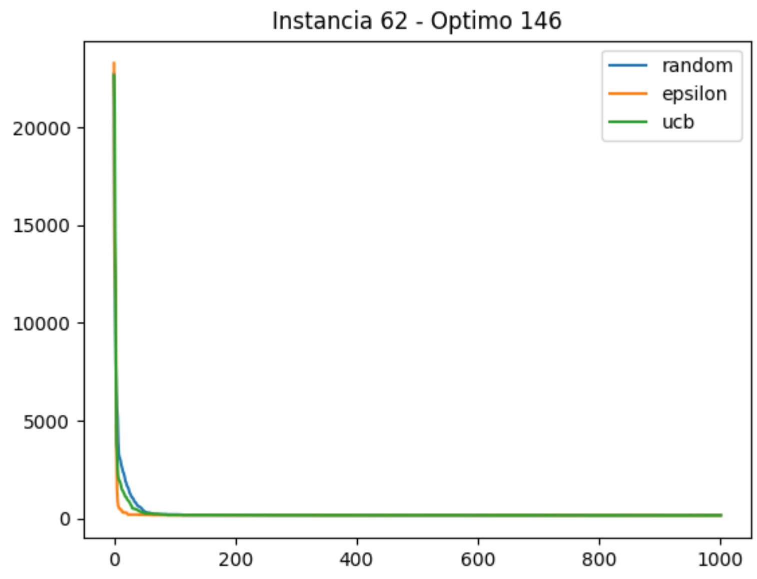


Figure 13: Convergence of 4.1 and 5.3 instances by agent policy



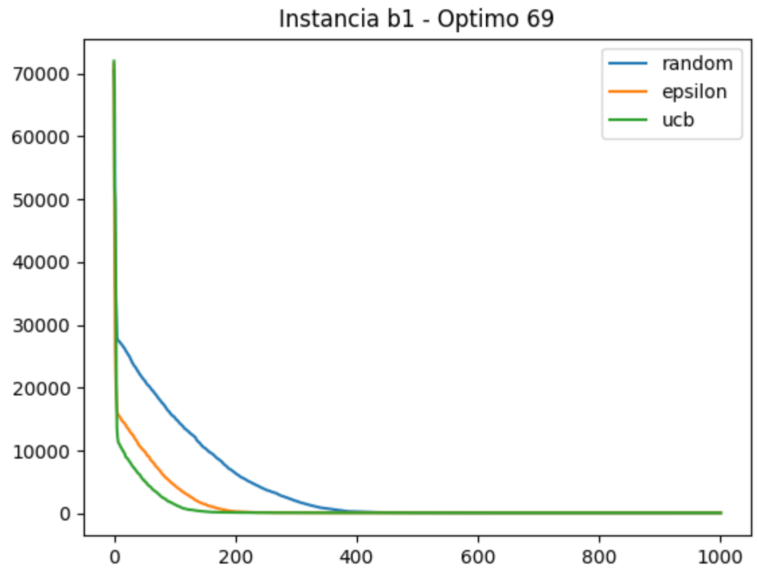


Figure 14: Convergence of 6.2 and b.1 instances by agent policy

It is also curious that the PH-test has not been activated once. This may be due to the λ parameter, or the way it was defined to reward the agent.

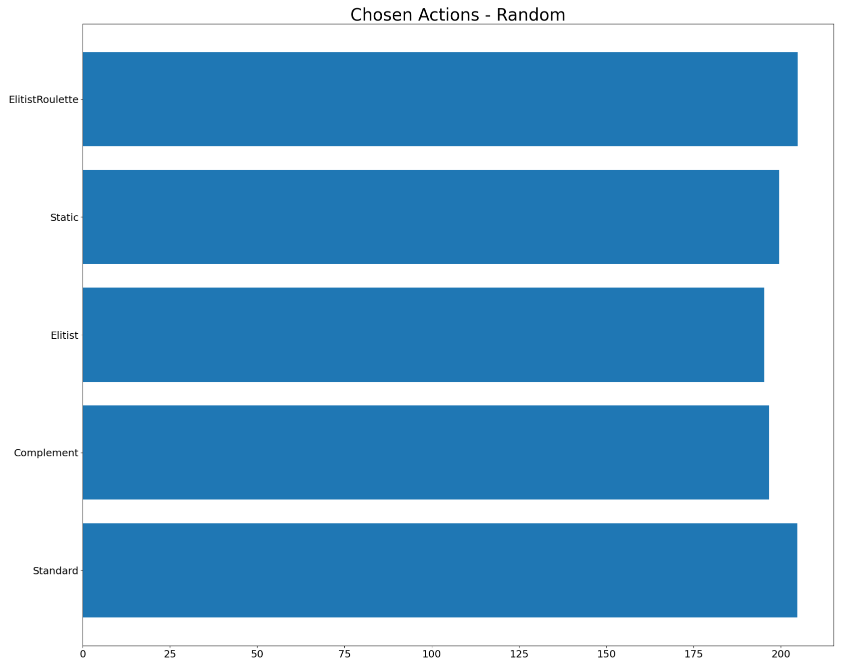


Figure 15: Average number of selections with Random policy

4.7 Discussion

Analyzing all the results obtained, we do not find enough evidence to ensure that using a dynamic binarization technique, such as DMAB, improves the results when solving the Set Covering Problem with BBWO. This may be due to many factors that affect this behavior that we will describe below.

First of all, the BWO metaheuristic, designed to solve continuous problems, presents a cross-over operation that does not allow exploring new solutions when dealing with binary variables and their individuals are the same

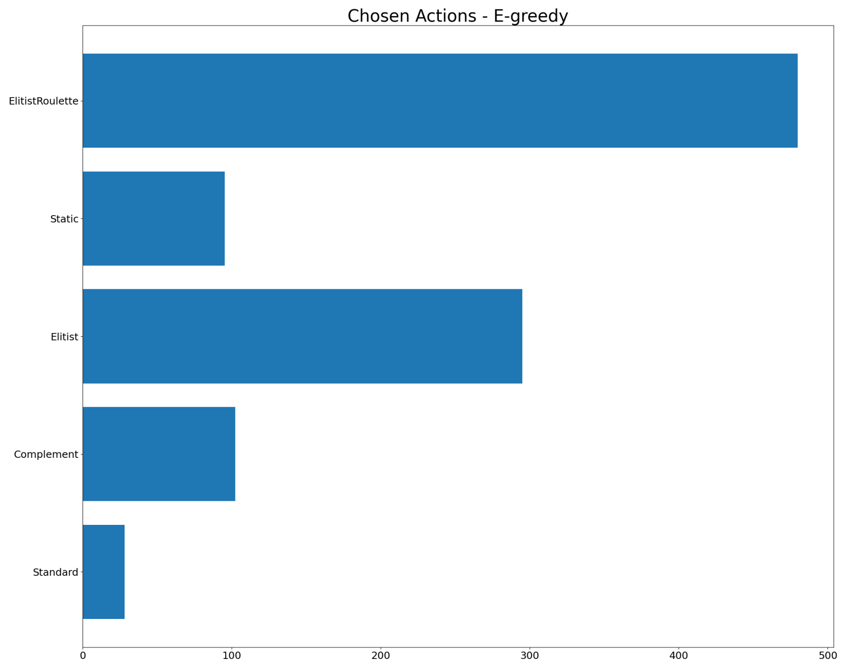


Figure 16: Average number of selections with ε-greedy policy

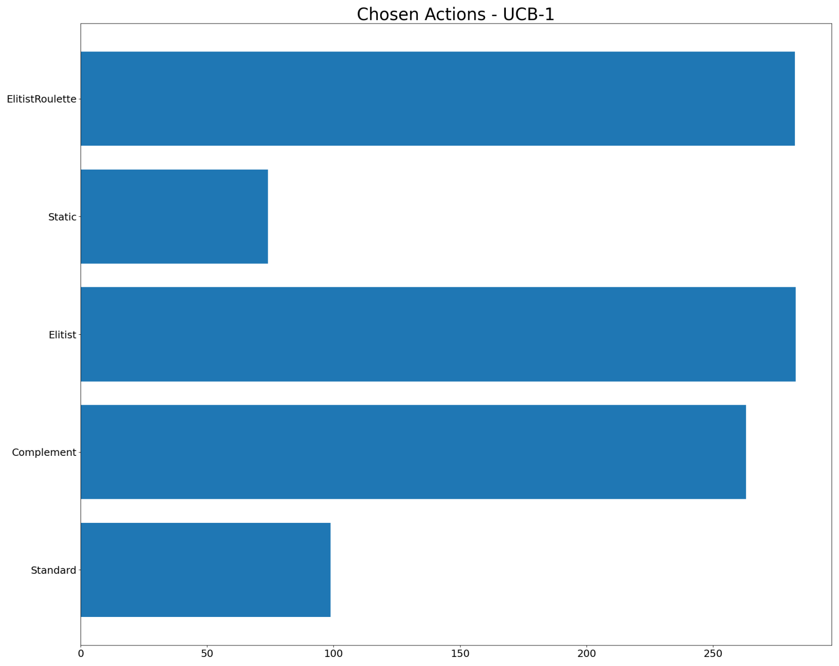


Figure 17: Average number of selections with UCB-1 policy

or very similar. This means that the search for new solutions is impaired over time, considering that when we approach an optimum, the individuals in the population are increasingly similar.

Secondly, when solving the SCP with instances of OR-Library, a problem is generated in the mutation of the algorithm, since, following the original operation, very few variables are modified in relation to the dimension of the problem, ranging between 1000 and 10000. This, added to the behavior of the cross-over, makes it difficult to get out of local optima, or else, a very slow convergence to an optimal value occurs.

Despite these possible problems present in the metaheuristics, the proposal presented in this thesis seems logical and it is believed that with a correct adjustment in the operators or parameters values, which directly affect the exploration and exploitation of the search field, better results can be found.

On the other hand, analyzing the behavior of DMAB, we can see that it did not offer improvements compared to a fixed binarization technique. It did deliver faster convergences, in some cases, but not better solutions. One of the possible problems that could cause this behavior is the way that rewarding the agent was defined.

There is no doubt that the experiments carried out and presented until now are not enough to draw great conclusions about the behavior of the different binarization functions, nor of the metaheuristic itself. For this reason, it was decided to make a second proposal that simplifies some features of the algorithm, to focus efforts on the performance of MAB as a selector of binarization schemes. With this second proposal, it will seek to obtain more accurate conclusions about the behavior of this reinforced learning algorithm and achieve the objective of this thesis.

5 Second proposal

This second proposal was designed based on the problems that arose during the development of the previous proposal and after analyzing the results obtained. The differences between the proposals include changes in the metaheuristics and in the design of the experiments. The problem to solve will continue to be SCP.

5.1 Binary Pendulum Search Algorithm

In this proposal it was decided to use the PSA metaheuristic, which is characterized by having a single mathematical operation that modifies the solutions of the population, simplifying the search process for new solutions and allowing us to focus our efforts on the work of MAB as a selection algorithm of binarization schemes.

Like BWO, this metaheuristic was also designed to solve continuous problems, so we need to go through the two-step binarization process. In this case, the values that the variables can obtain after the mathematical operations proposed in the metaheuristics do vary in a range greater than [0,1], so in this case we must use the transfer functions before the binarization functions. In total we would have 40 different combinations to be able to binarize the continuous values of the variables (figure 18).

5.2 MAB-BPSA

As mentioned in the 4.7 section, one of the reasons for modifying the metaheuristics was its complexity in solving SCP. By using the instance dimension to define the number of new children generated, resolving OR-Library instances leaves the population too large, increasing execution times and, in some cases, premature convergence.

Also, when verifying that the DMAB PH-Test was not activated on any execution, it was decided to leave it out of the experiments in order to capture the behavior of MAB on its own: the less variables you work with, the more accurate the conclusions obtained will be.

In this way, as a summary, the Multi-armed bandit model to use is as follows:

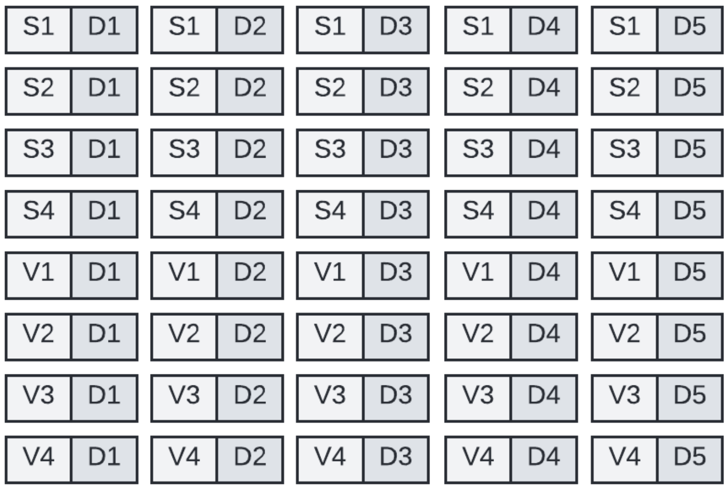

S1 D1 S1 D2 S1 D3 S1 D4 S1 D5
S2 D1 S2 D2 S2 D3 S2 D4 S2 D5
S3 D1 S3 D2 S3 D3 S3 D4 S3 D5
S4 D1 S4 D2 S4 D3 S4 D4 S4 D5
V1 D1 V1 D2 V1 D3 V1 D4 V1 D5
V2 D1 V2 D2 V2 D3 V2 D4 V2 D5
V3 D1 V3 D2 V3 D3 V3 D4 V3 D5
V4 D1 V4 D2 V4 D3 V4 D4 V4 D5

Figure 18: 40 binarization schemes

- **Environment**: As mentioned in section 5.1, in this case the binarization schemes will be complained by the combination of transfer function and binarization function, giving us a total of 40 possible ways to binarize our continues variables (figure 18).
- **Policy**: The policy will be governed by the value function UCB-1.
- **Reward**: The reward will be defined by the fitness improvement percentage of each execution of the BPSA algorithm.

The pseudo-code of the final algorithm is presented in 5. As can be seen, the logic of the algorithm is simplified, reducing the number of steps and operations to perform.

Algorithm 5 MAB-BPSA

```
    Input: The population $X = \{X_1, X_2, ..., X_i\}$
    Output: The updated population $X' = \{X'_1, X'_2, ..., X'_i\}$ and $X_{Best}$
 1: Initialize random population X
 2: Evaluate the fitness of each individual in the population X
 3: Identify the best individual in the population ($X_{Best}$)
 4: for $iteration\,(t)$ do
 5:     for $solution\,(i)$ do
 6:         for $dimension\,(j)$ do
 7:             Update $pend^t_{i.j}$ by Eq. (17)
 8:             Update the position of $X^t_{i,j}$ using Eq. (16)
 9:         end for
10:         Apply binarization scheme selected by MAB (Tables 1 - 3 )
11:         Repair solutions if the are not feasible. (Section (4.1.2))
12:     end for
13:     Evaluate the fitness of each individual in the population X
14:     Generate MAB reward with Eq. (29) from fitness
15:     Update UCB-1 value to select the next binarization scheme
16:     Update $X_{Best}$
17: end for
18: Return the updated population $X'$ where $X_{Best}$ is the best result
```

5.3 Design of experiments

As mentioned in [31], one of the best binarization schemes is the combination of the V4 transfer function and the Elitist binarization rule. This is why we will compare the result of the BPSA execution, configured with V4-Elitist in all its iterations, versus the MAB-BPSA proposal, binarizing in each iteration with the technique provided by the MAB algorithm. Unlike the previous proposal (DMAB-BBWO), only the value function that offers the best results (of the three presented) according to the literature will be used: UCB-1 [14].

Regarding the development, first of all, the instances of the SCP problem were captured from OR-library, from which instances 41, 51, 61, a1, b1, c1, nre1 and nrf1 were used. In this way we have a good representation of the different families of instances. More details of each instance in table 5.

Second, the BPSA and MAB algorithms, along with the different bina-

Context	**Param**	**Value**
PSA	Population	40
	Iters	500
MAB	c	$\sqrt{2}$
Fixed bin.	Transfer func.	V4
	Binary func.	Elitist

Table 8: Parameter configuration.

rization techniques, were developed in the Python v3.9 with the NumPy library to optimize matrix calculations.

Finally, the experiments were carried out on a Macbook Air computer with an Apple M1 processor, 7-core CPU and 8GB of RAM. Each instance was executed 31 times independently, enough quantity to have a confidence idea of the behavior of the algorithm in each one of the instances.

Regarding the configuration of parameters, a population size of 40 individuals and 500 iterations were used for both tested approaches, 31 independent runs were performed and the value for the value c of Eq. 21 is $\sqrt{2}$, based on the suggestion of [14]. The parameters are summarized in table 8.

5.4 Experimental results

The results obtained are shown in Table 9. This table has ten columns, where the first one represents the instance of the Set Covering Problem evaluated, the second one represents the global optimum of instances and the next four columns are repeated for each algorithm executed. The first of these shows the best fitness obtained among the thirty-one independent runs, the second of these shows the average across the thirty-one independent runs, the third one of these shows the standard deviation of the thirty-one independent runs, and the fourth one shows the Relative Percentage Deviation (RPD) between global optimum and the best fitness obtained among the thirty-one independent runs. RPD is defined as follows:

$$RPD = \frac{Z - Z_{opt}}{Z_{opt}} \times 100 \tag{30}$$

where Z corresponds to the best value found and Z_{opt} the global optimal value that is expected to be reached.

		V4-ELIT				MAB			
Inst.	Opt	Best	Avg-fit	Std-dev-fit	RPD	Best	Avg-fit	Std-dev-fit	RPD
41	429	**433**	**433**	**0**	**0.932**	**433**	**433**	**0**	**0.932**
51	253	267	267	0	5.534	**257**	**266.452**	**2.173**	**1.581**
61	138	141	142.774	1.91	2.174	**141**	**141**	**0**	**2.174**
a1	253	257	257.065	0.25	1.581	**257**	**257**	**0**	**1.581**
b1	69	69	69.161	0.374	0	**69**	**69.097**	**0.301**	**0**
c1	227	**230**	**232.387**	**1.202**	**1.322**	231	232.645	0.798	1.762
d1	60	60	60.355	0.661	0	**60**	**60.871**	**0.619**	**0**
nre1	29	**29**	**29**	**0**	**0**	**29**	**29**	**0**	**0**
nrf1	14	**14**	**14**	**0**	**0**	**14**	**14**	**0**	**0**
Avg.	163.556	166.667	167.194	0.489	1.283	**165.667**	**167.007**	**0.432**	**0.892**

Table 9: Comparison of fitness between fixed and dynamic binarization.

The following criteria were used to determine the best algorithm:

* **Best fitness obtained and RPD**: This allows us to see what our best result was and how far it is from the global optimum.
* **Standard Deviation**: A low standard deviation indicates the results obtained with the thirty-one independent runs were close.
* **Average between thirty-one independent**: An average close to the optimal value indicates the results obtained with the thirty-one independent runs the algorithm performed well.

With this in mind, our proposed BPSA with MAB as binarization schemes selector won in 5 out of 9 instances, tied in 3 out of 9 instances, and only lost in one instance.

From figure 19 to 22 shows the convergence plots of the best execution of each algorithm run. The X-axis shows the iterations and the Y-axis shows the best fitness obtained during the process. The left plot show the complete range of values, and the right side a zoom-in plot in order to see the behavior in more detail.

In these two figures, we can see the BPSA with MAB has a slower convergence compared to BPSA with a fixed binarization scheme.

Thus, we can demonstrate that using MAB as a selector of binarization schemes helps to balance the exploration and exploitation of BPSA and to

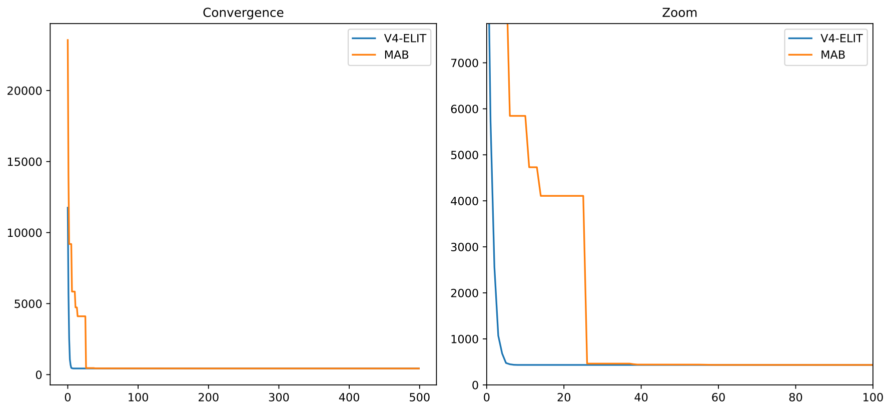


Figure 19: Fitness convergence and Zoom for instance 41

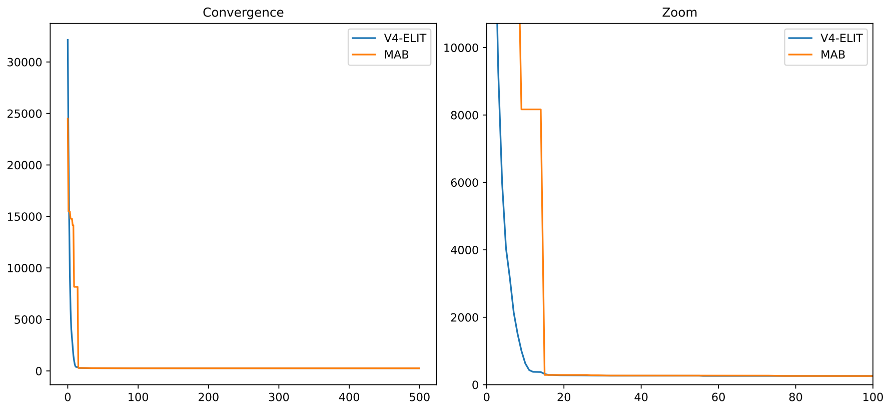


Figure 20: Fitness convergence and Zoom for instance 51

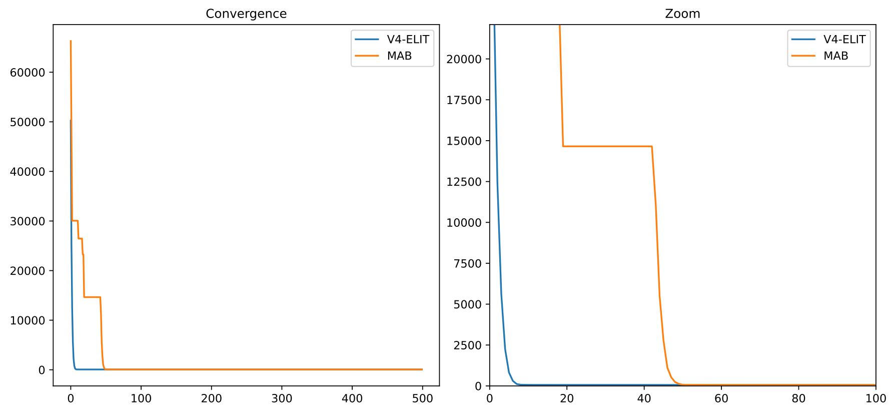


Figure 21: Fitness convergence and Zoom for instance d1

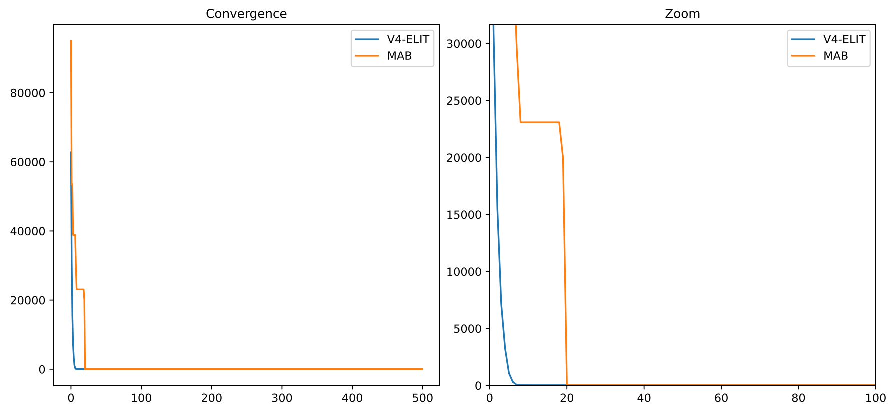


Figure 22: Fitness convergence and Zoom for instance nre1

	V4-ELIT		MAB	
Instance	Avg-time (s)	Std-dev-time (s)	Avg-time (s)	Std-dev-time (s)
41	**69.429**	4.15	71.727	9.763
51	88.634	4.35	89.497	13.175
61	42.355	3.303	**39.683**	5.878
a1	204.238	5.852	**193.114**	17.502
b1	**113.087**	4.817	123.087	17.13
c1	408.542	10.897	**383.78**	41.616
d1	218.491	6.603	**199.662**	20.153
nre1	**257.875**	7.011	277.048	34.237
nrf1	**148.318**	5.306	174.743	21.352
average	**172.330**	5.810	172.482	20.090

Table 10: Comparison of time between fixed and dynamic binarization.

find better solutions.

On the other hand, in figure 24 we can see 3 horizontal bar graphs showing the number of times each actions was selected. The one on the left shows how they had been selected in the first 50 iterations of the run, then at 200 iterations, and on the right at 500 iterations. In this way we can analyze the behavior that MAB algorithm had throughout the execution. From figure 24 which shows the average selection of instance 51, it stands out that in the first iterations, where the algorithm has a greater exploratory component, the 4 actions that are most selected have as a binarization rule (D_2 at table 3) the complement function, while in advanced iterations the actions with the Elitist or Elitist Roulete function are selected more (D_4 and D_5 at table 3). This result makes sense with what is described in [31]. A similar behavior can be observed in figures 23, 25 and 26.

As mentioned above, the use of metaheuristics lies in their efficiency in delivering good results in reasonable times. The Table 10 shows the execution times of each proposal, where the first one represents the instance of the Set Covering Problem evaluated and the next two columns are repeated for each algorithm executed. The first-one of these shows the average time in seconds across the thirty-one independent runs, and the second one of these shows the standard deviation of the thirty-one independent runs.

As can be seen in this table, both proposals have very similar implementation times. This indicates that there is not a large computational increase when incorporating a machine learning technique such as MAB.

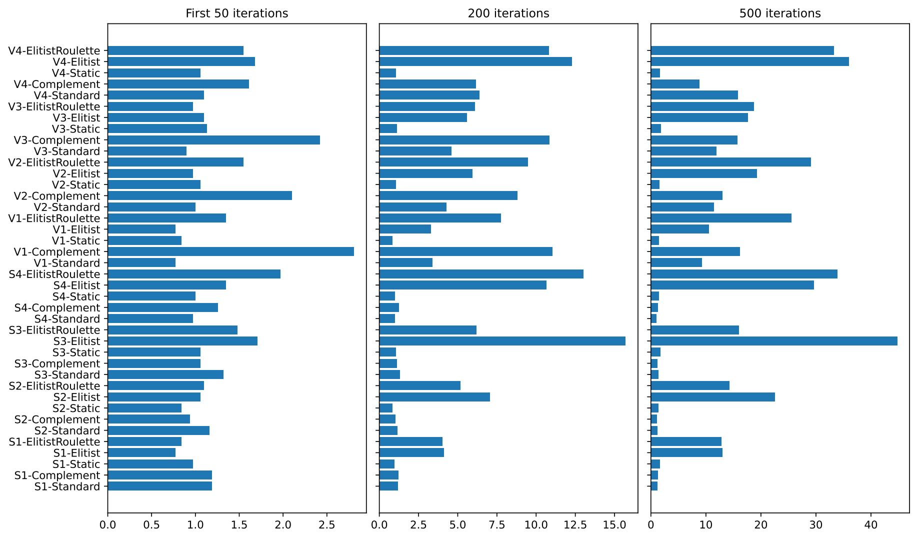


Figure 23: Average number of selections on instance 41

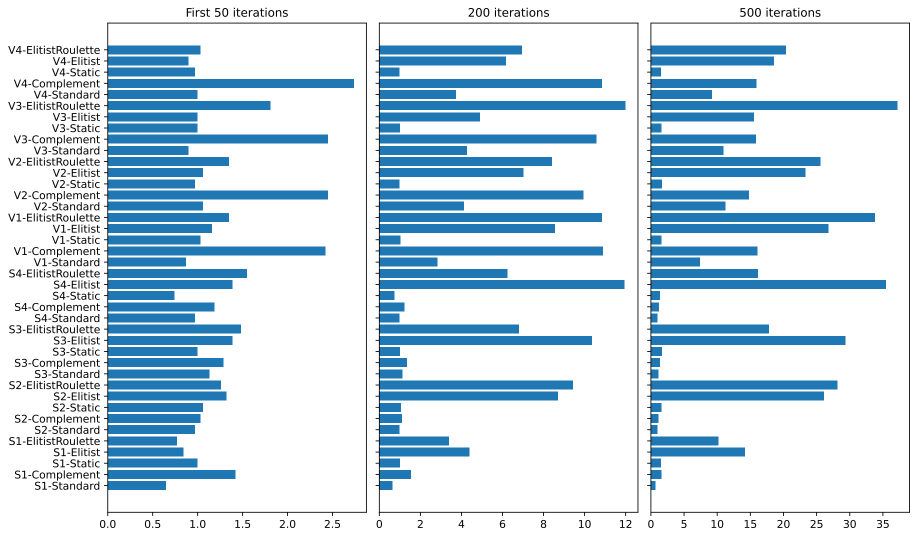


Figure 24: Average number of selections on instance 51

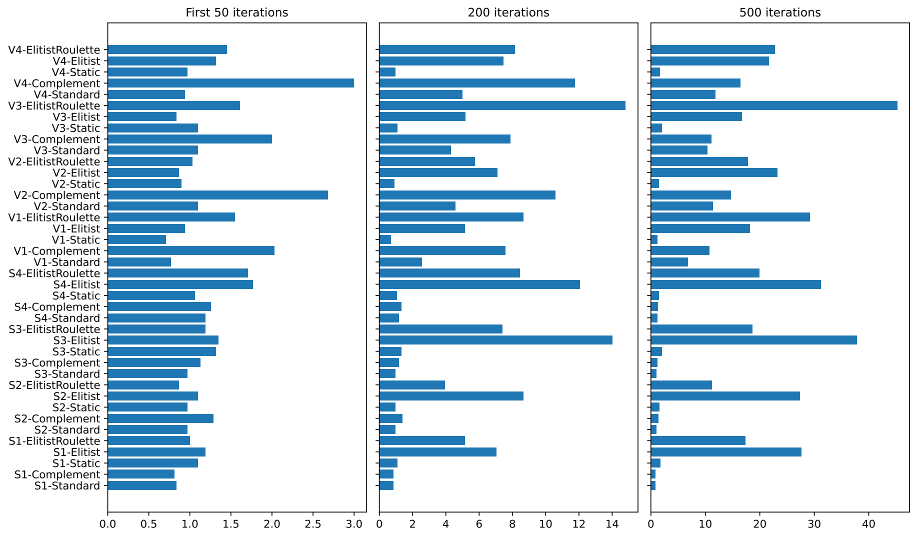


Figure 25: Average number of selections on instance c1

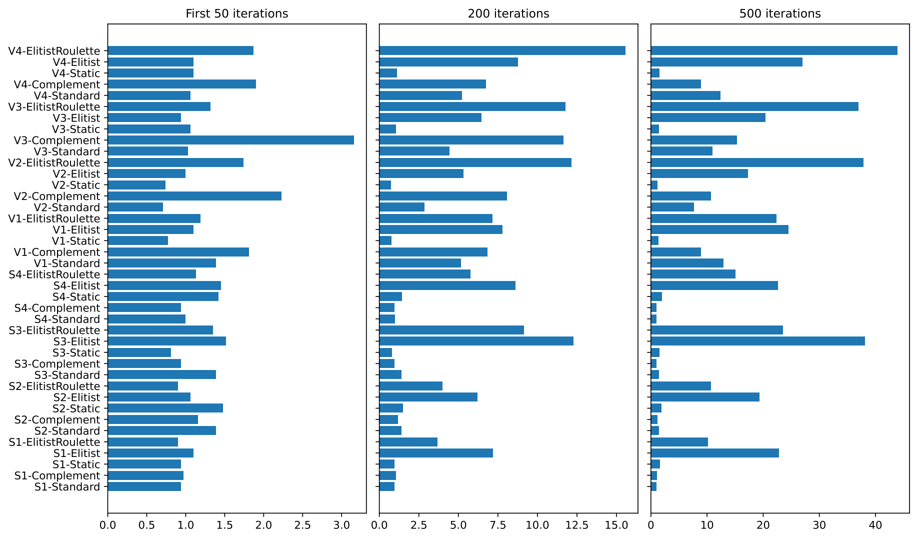


Figure 26: Average number of selections on instance d1

6 Conclusion

This document has presented a brief analysis of the state of the art related to the area of optimization and machine learning. Along with the explanation of the proposed solution. In addition, the proposal of the BWO and PSA algorithm in its binary version is presented in detail, with its due justification, in order to solve the Set Covering Problem.

The ease of accessing large computational capacities at reduced costs has enabled the use of machine learning techniques such as Multi Armed Bandit. The research on hybrid algorithms between metaheuristics and machine learning with the aim of improving the search process is increasing every year and the present work is an example of that. In particular, Multi Armed Bandit was successfully incorporated into Pendulum Search Algorithm where it was used to dynamically and intelligently select binarization schemes.

Preliminary results indicate that BWO had trouble solving SCP and that MAB did not offer any help in their goal of getting better results. On the other hand, results indicate that our second proposal (MAB-BPSA) performs better when compared to Pendulum Search Algorithm using a fixed binarization scheme (V4-Elitist). Better results were obtained by improving the balance of diversification and intensification in the Pendulum Serach Algorithm search process. During the diversification process, Multi Armed Bandit determined that the most exploratory binarization schemes are those that include the Complement binarization function. In contrast, for the intensification process, Multi Armed Bandit determined that the most exploitative binarization schemes are those that include the Elitist or Elitist Roulette binarization function.

Regarding computation times, the results indicate that there is no great increase in computation times when comparing V4-Elitist and MAB-BPSA. This dismisses that incorporating machine learning techniques to metaheuristic algorithms increases the computational time.

With the results described, it can be said that the main objective of this thesis was met, but there are still many issues to be addressed in order to obtain good results in other metaheuristic proposals and combinatorial problems. The current situation in which this research finds itself opens the way for several future investigations, mainly due to the simplification of the first proposal. First of all, the correct configuration and implementation of

DMAB can offer a wide range of experimentation, applying it to different metaheuristics (SCA, PSA, among others), which theoretically could offer good results.

Secondly, another possible issue to be addressed in the future is to improve the design of BWO. New operations can be proposed that reduce execution times. The changes proposed in this thesis were a great advance in this work, but it was not possible to arrive at a clean algorithm that is capable of highlighting some characteristic by solving combinatorial problems.

The third place, when observing the results of the second proposal, where it was observed how the complement binarization function is used in exploration stages, while Elitist and Elitista Roullete in exploitation stages, it would be interesting to analyze the diversity measures described in the 2.2 section to be able to empirically measure the exploration and exploitation capacity of each binarization scheme.

Finally, a characteristic shared by both proposals was the way of rewarding the MAB agent. A possible problem presented by the improvement percentage (Eq. 29) is that in the first iterations the reward is much higher than in advanced iterations, precisely where it is more difficult to improve in fitness. A possible future investigation would be the normalization of said reward function, in order to avoid biases generated in the first iterations.

As a last detail to highlight, PSA is a metaheuristic that made the experiments much easier, because it did not have parameters to configure and because of its simplicity in the movement executions of the individuals, without harming the results obtained. This allows it to be easily used for future investigations as the default metaheuristic, and thus concentrate on the focus of said investigation, without worrying about having to adjust parameters or configure the metaheuristics.

References

[1] Nor Azlina Ab. Aziz and Kamarulzaman Ab. Aziz. Pendulum search algorithm: An optimization algorithm based on simple harmonic motion and its application for a vaccine distribution problem. *Algorithms*, 15(6):214, 2022.

[2] Tasiransurini Ab Rahman, Zuwairie Ibrahim, Nor Azlina Ab. Aziz, Shunyi Zhao, and Nor Hidayati Abdul Aziz. Single-agent finite impulse response optimizer for numerical optimization problems. *IEEE Access*, 6:9358–9374, 2018.

[3] Ahmed Al-Saedi. Binary black widow optimization algorithm for feature selection problems. 2021.

[4] JP Arabeyre, J Fearnley, FC Steiger, and W Teather. The airline crew scheduling problem: A survey. *Transportation Science*, 3(2):140–163, 1969.

[5] Jean-Yves Audibert, Rémi Munos, and Csaba Szepesvári. Exploration–exploitation tradeoff using variance estimates in multi-armed bandits. *Theoretical Computer Science*, 410(19):1876–1902, 2009.

[6] P. Auer, N. Cesa-Bianchi, and P. Fischer. Finite-time analysis of the multiarmed bandit problem. *Machine learning*, 47(2):235–256, 2002.

[7] Egon Balas and Paolo Toth. Branch and bound methods for the traveling salesman problem. 1983.

[8] Michel L Balinski and Richard E Quandt. On an integer program for a delivery problem. *Operations research*, 12(2):300–304, 1964.

[9] Jany Belluz, Marco Gaudesi, Giovanni Squillero, and Alberto Tonda. Operator selection using improved dynamic multi-armed bandit. In *Proceedings of the 2015 Annual Conference on Genetic and Evolutionary Computation*, pages 1311–1317, 2015.

[10] Ilhem Boussaïd, Julien Lepagnot, and Patrick Siarry. A survey on optimization metaheuristics. *Information Sciences*, 237:82 – 117, 2013. Prediction, Control and Diagnosis using Advanced Neural Computations.

[11] Seung-Seok Choi, Sung-Hyuk Cha, and Charles C Tappert. A survey of binary similarity and distance measures. *Journal of systemics, cybernetics and informatics*, 8(1):43–48, 2010.

[12] Broderick Crawford, Ricardo Soto, Gino Astorga, José García, Carlos Castro, and Fernando Paredes. Putting continuous metaheuristics to work in binary search spaces. *Complexity*, 2017, 2017.

[13] Broderick Crawford, Ricardo Soto, Rodrigo Cuesta, and Fernando Paredes. Application of the artificial bee colony algorithm for solving the set covering problem. *The Scientific World Journal*, 2014:Article ID 189164, 2014.

[14] Luis DaCosta, Alvaro Fialho, Marc Schoenauer, and Michèle Sebag. Adaptive operator selection with dynamic multi-armed bandits. In *Proceedings of the 10th annual conference on Genetic and evolutionary computation*, pages 913–920, 2008.

[15] Luis DaCosta, Álvaro Fialho, Marc Schoenauer, and Michèle Sebag. Adaptive operator selection with dynamic multi-armed bandits. *GECCO '08*, 2008.

[16] Charles Darwin. On the origin of species, 1859, 2016.

[17] Lawrence Davis. Handbook of genetic algorithms. 1991.

[18] Marco Dorigo, Mauro Birattari, and Thomas Stutzle. Ant colony optimization. *IEEE computational intelligence magazine*, 1(4):28–39, 2006.

[19] Marco Dorigo, Vittorio Maniezzo, and Alberto Colorni. Ant system: optimization by a colony of cooperating agents. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, 26(1):29–41, 1996.

[20] Gangan Elena, Kudus Milos, and Ilyushin Eugene. Survey of multi-armed bandit algorithms applied to recommendation systems. *International Journal of Open Information Technologies*, 9(4):12–27, 2021.

[21] Yaping Fu, Yushuang Hou, Zhenghua Chen, Xujin Pu, Kaizhou Gao, and Ali Sadollah. Modelling and scheduling integration of distributed production and distribution problems via black widow optimization. *Swarm and Evolutionary Computation*, 68:101015, 2022.

[22] Michael R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman amp; Co., USA, 1979.

[23] Robert S Garfinkel and George L Nemhauser. The set-partitioning problem: set covering with equality constraints. *Operations Research*, 17(5):848–856, 1969.

[24] Fred Glover. Tabu search: A tutorial. *Interfaces*, 20(4):74–94, 1990.

[25] Abdolreza Hatamlou. Black hole: A new heuristic optimization approach for data clustering. *Information sciences*, 222:175–184, 2013.

[26] V. Hayyolalam and A. A. Pourhaji Kazem. Black Widow Optimization Algorithm: A novel meta-heuristic approach for solving engineering optimization problems. *Engineering Applications of Artificial Intelligence*, 87:1–28, 10 2019.

[27] John H Holland and Judith S Reitman. Cognitive systems based on adaptive algorithms. *Acm Sigart Bulletin*, (63):49–49, 1977.

[28] Essam H Houssein, Bahaa El-din Helmy, Diego Oliva, Ahmed A Elngar, and Hassan Shaban. A novel black widow optimization algorithm for multilevel thresholding image segmentation. *Expert Systems with Applications*, 167:114159, 2021.

[29] Gang Hu, Bo Du, Xiaofeng Wang, and Guo Wei. An enhanced black widow optimization algorithm for feature selection. *Knowledge-Based Systems*, 235:107638, 2022.

[30] Jose M Lanza-Gutierrez, NC Caballe, Broderick Crawford, Ricardo Soto, Juan A Gomez-Pulido, and Fernando Paredes. Exploring further advantages in an alternative formulation for the set covering problem. *Mathematical Problems in Engineering*, 2020:Article ID: 5473501, 2020.

[31] Jose M Lanza-Gutierrez, Broderick Crawford, Ricardo Soto, Natalia Berrios, Juan A Gomez-Pulido, and Fernando Paredes. Analyzing the effects of binarization techniques when solving the set covering problem through swarm optimization. *Expert Systems with Applications*, 70:67–82, 2017.

[32] José Lemus-Romani, Marcelo Becerra-Rozas, Broderick Crawford, Ricardo Soto, Felipe Cisternas-Caneo, Emanuel Vega, Mauricio Castillo,

Diego Tapia, Gino Astorga, Wenceslao Palma, Carlos Castro, and José García. A novel learning-based binarization scheme selector for swarm algorithms solving combinatorial problems. *Mathematics*, 9(22):2887, Nov 2021.

[33] Xueyan Lu and Yongquan Zhou. A novel global convergence algorithm: bee collecting pollen algorithm. In *International conference on intelligent computing*, pages 518–525. Springer, 2008.

[34] Sargol Memar, Amin Mahdavi-Meymand, and Wojciech Sulisz. Prediction of seasonal maximum wave height for unevenly spaced time series by black widow optimization algorithm. *Marine Structures*, 78:103005, 2021.

[35] Seyedali Mirjalili. Sca: a sine cosine algorithm for solving optimization problems. *Knowledge-based systems*, 96:120–133, 2016.

[36] Seyedali Mirjalili and Andrew Lewis. S-shaped versus v-shaped transfer functions for binary particle swarm optimization. *Swarm and Evolutionary Computation*, 9:1–14, 2013.

[37] Seyedali Mirjalili and Andrew Lewis. The whale optimization algorithm. *Advances in engineering software*, 95:51–67, 2016.

[38] Seyedali Mirjalili, Seyed Mohammad Mirjalili, and Andrew Lewis. Grey wolf optimizer. *Advances in engineering software*, 69:46–61, 2014.

[39] Mehryar Mohri, Afshin Rostamizadeh, and Ameet Talwalkar. *Foundations of machine learning.* The MIT Press, Cambridge, MA, second edition, 2018.

[40] Bernardo Morales-Castañeda, Daniel Zaldivar, Erik Cuevas, Fernando Fausto, and Alma Rodríguez. A better balance in metaheuristic algorithms: Does it exist? *Swarm and Evolutionary Computation*, 54:100671, 2020.

[41] P Mukilan and Wogderess Semunigus. Human object detection: An enhanced black widow optimization algorithm with deep convolution neural network. *Neural Computing and Applications*, 33(22):15831–15842, 2021.

[42] Vishakha Patil, Ganesh Ghalme, Vineet Nair, and Yadati Narahari. Achieving fairness in the stochastic multi-armed bandit problem. In *AAAI*, pages 5379–5386, 2020.

[43] Riccardo Poli, James Kennedy, and Tim Blackwell. Particle swarm optimization. *Swarm intelligence*, 1(1):33–57, 2007.

[44] Herbert Robbins and Herbert Robbins. Some aspects of the sequential design of experiments. *Bulletin of the American Mathematical Society*, 1952.

[45] Eduardo Rodriguez-Tello, Valentina Narvaez-Teran, and Frederic Lardeux. Dynamic multi-armed bandit algorithm for the cyclic bandwidth sum problem. *IEEE Access*, 7:40258–40270, 2019.

[46] Ali Sadollah, Ardeshir Bahreininejad, Hadi Eskandar, and Mohd Hamdi. Mine blast algorithm: A new population based algorithm for solving constrained engineering optimization problems. *Applied Soft Computing*, 13(5):2592–2612, 2013.

[47] Heda Song, Isaac Triguero, and Ender Özcan. A review on the self and dual interactions between machine learning and optimisation. *Progress in Artificial Intelligence*, 8(2):143–165, 2019.

[48] Kenneth Sörensen and Fred Glover. Metaheuristics. *Encyclopedia of operations research and management science*, 62:960–970, 2013.

[49] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction.* The MIT Press, second edition, 2018.

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

[51] Chun-Hung Tzeng. Heuristic information. In *A Theory of Heuristic Information in Game-Tree Search*, pages 51–63. Springer, 1988.

[52] Anant J Umbarkar and Pranali D Sheth. Crossover operators in genetic algorithms: a review. *ICTACT journal on soft computing*, 6(1), 2015.

[53] Peter JM Van Laarhoven and Emile HL Aarts. Simulated annealing. In *Simulated annealing: Theory and applications*, pages 7–15. Springer, 1987.

[54] Ling Wang, Xiuting Wang, Jingqi Fu, and Lanlan Zhen. A novel probability binary particle swarm optimization algorithm and its application. *J. Softw.*, 3(9):28–35, 2008.

[55] Xin-She Yang. *Engineering Optimization: An Introduction with Metaheuristic Applications.* 2010.

[56] Álvaro Fialho, Luis Da Costa, Marc Schoenauer, and Michèle Sebag. Dynamic multi-armed bandits and extreme value-based rewards for adaptive operator selection in evolutionary algorithms. *LION*, 2009.