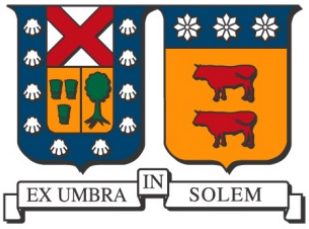
UNIVERSIDAD TÉCNICA FEDERICO SANTA MARÍA
DEPARTAMENTO DE INFORMÁTICA
CIUDAD - CHILE



“DISEÑO DE UNA ESTRATEGIA PARA OBTENER CONOCIMIENTO EN UN SINTONIZADOR DE PARÁMETROS”

DANIEL ALEJANDRO TORO VEGA

MEMORIA PARA OPTAR AL TÍTULO DE
INGENIERO CIVIL EN INFORMÁTICA

Profesor Guía: Nicolás Rojas
Profesor Correferente: José Luis Martí

Noviembre - 2022

DEDICATORIA

Dedico este trabajado a mi familia y amigos, sin ellos nunca hubiese llegado hasta aquí.

AGRADECIMIENTOS

Quiero agradecer en primer lugar a mis padres Luis y Elizabeth por el apoyo incondicional que me han dado. Además, quiero agradecer a mis amigos y compañeros de la universidad, quienes en muchas ocasiones me inspiraron para seguir adelante. Finalmente, quiero agradecer a mi profesor guía Nicolás Rojas por el apoyo y toda la paciencia para este trabajo.

RESUMEN

Resumen— La toma de decisiones es un aspecto cotidiano tanto para las personas como para las grandes corporaciones. Una gran cantidad de decisiones pueden ser modeladas como un problema de optimización. Los problemas de optimización son problemas en donde se debe decidir cómo utilizar los recursos disponibles, de tal manera de lograr un objetivo. Para resolver problemas de optimización es frecuente el uso de metaheurísticas, algoritmos que son capaces de obtener soluciones de buena calidad en un tiempo razonable. Sin embargo, el uso de metaheurísticas suele introducir el problema de determinar los valores adecuados para sus parámetros. Este problema se le conoce como el problema de seteo de parámetros (PSP). Para abordar este problema se han propuesto métodos de sintonización y control de parámetros. En esta memoria se trabajó con el sintonizador de parámetros Evolutionary Calibrator (EVOCA) y se propuso una nueva versión. Esta nueva versión utiliza un modelo de clasificación, Naive Bayes Classifier (NBC) para determinar si el operador de mutación se debe utilizar en otro parámetro. A esta nueva versión se le llamó EVOCA+NBC. Se obtuvo evidencia experimental donde EVOCA+NBC obtiene mejores resultados para las configuraciones en términos de calidad y tiempo de ejecución. Además, se vio que es posible obtener configuraciones equiparables a EVOCA en menos evaluaciones.

Palabras Clave— Problema de seteo de parametros; Evolutionary Calibrator; Naive Bayes Classifier; Metaheurística

ABSTRACT

***Abstract*—** Decision making is a common aspect for people and even big corporations. A wide range of decisions can be modeled as an optimization problem. Optimization problems are problems where one must decide how to use available resources to achieve a goal. To solve this kind of problems it is common to use metaheuristics, algorithms that can produce good quality solutions in reasonable time. However, the use of metaheuristics introduces the problem to determine the adequate values of its parameters. This problem is known as the Parameter setting problem (PSP). To tackle this problem parameter control and tuning methods have been proposed. In this work the parameter tuner Evolutionary Calibrator was used and a new version was proposed. This new version utilizes a classification model,

Naive Bayes Classifier (NBC), to determine if the mutation operator should be used on another parameter. The new version was named EVOCA+NBC. Experimental evidence was found that shows that EVOCA+NBC can obtain better results, in terms of solution quality and execution time. Additionally, it was seen that it is possible to obtain comparable configurations to EVOCA in less evaluations.

Keywords— Parameter setting problem; Evolutionary Calibrator; Naive Bayes Classifier; Metaheuristic

GLOSARIO

ACC: Accuracy
ACO: Ant Colony Optimization
AK: Ant Knapsack
ANOVA: Analisis of variance
APA: Adaptive Pursuit Algorithm
CART: Classification and Regression Trees
DT: Decision Tree
EVOCA: Evolutionary Calibrator
FN: False negative
FP: False positive
GR: Gain Ratio
I-race: Iterated F-Race
ID3: Iterative Dichotomiser 3
K-fold: K-fold cross validation
KNN: K-nearest neighbors
NBC: Naive bayes Classifier
PSO: article swarm optimization
PSP: Problema de seteo de parámetros
ParamILS: Parameter iterated local search
RB: Red Bayesiana
RIP: Root Identification Problem
RT:Regression Trees
SMAC: Sequential Model-based Algorithm Configuration
SVM: Support Vector Machines
TN: True negative
TP: True positive

ÍNDICE DE CONTENIDOS

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

INTRODUCCIÓN

La toma de decisiones es un aspecto cotidiano tanto para las personas como para las grandes corporaciones. La investigación de operaciones es un subcampo de las matemáticas que intenta proporcionar una base objetiva y cuantitativa para la toma de decisiones. Uno de los problemas que se trata en la investigación de operaciones son los problemas de optimización. Los problemas de optimización, en términos generales, son problemas en donde se tiene un objetivo determinado que debe ser llevado a cabo, dado un número restricciones y recursos. Por ejemplo, el problema del vendedor viajero consiste en que un vendedor desea visitar un conjunto de ciudades, exactamente una vez, partiendo y terminado su recorrido desde su ciudad de origen, de tal manera que recorra la menor distancia posible. En este caso, el objetivo es recorrer la menor distancia posible y las restricciones del recorrido son partir y terminar en el mismo punto y visitar todas las ciudades a lo más una vez.

Existen diversos enfoques para resolver problemas de optimización como por medio de métodos exactos o metaheurísticas. Los métodos exactos garantizan (en caso de existir) obtener la solución óptima al problema, luego de terminar su ejecución. Sin embargo, en aplicaciones del mundo real han surgido problemas computacionalmente desafiantes de resolver, lo cual provoca que el tiempo requerido por los métodos exactos aumente de manera exponencial en el peor de los casos [Blum y Roli, 2003]. A diferencia de los métodos exactos, las metaheurísticas no garantizan encontrar la solución óptima al problema, sin embargo, pueden obtener soluciones de alta calidad en un tiempo razonable. Es por esto, que las metaheurísticas son la alternativa principal para resolver problemas de optimización complejos, razón por la cual han recibido un creciente interés como objeto de estudio [Huang *et al.*, 2019].

El uso de metaheurísticas para resolver problemas de optimización suele introducir el problema adicional de la selección de valores para los parámetros que utilizan, lo cuales están fuertemente relacionados a su rendimiento [Montero *et al.*, 2014]. Este problema se conoce como el problema de seteo de parámetros. Para abordar este problema se han propuesto métodos de control de parámetros y métodos de sintonización de parámetros. La sintonización de parámetros puede ser descrita como el proceso de buscar valores adecuados para los parámetros es un paso anterior a la ejecución del algoritmo. Estos parámetros se mantienen fijos durante la ejecución del algoritmo. Por otro lado, los métodos de control de parámetros definen y modifican los valores de los parámetros del algoritmo durante su ejecución.

Debido a los crecientes avances en la tecnología y el aumento de su disponibilidad, una masiva cantidad de datos son generados en diversas áreas como las redes sociales, la medicina, educación, entre otros. Dado esto, se han propuesto diversas técnicas de obtención de conocimiento para poder analizar y dar uso a estos crecientes volúmenes de información. En esta memoria se explora el uso de una técnica de obtención de conocimiento en un sintonizador de parámetros.

En el capítulo 1, en primer lugar se introducen conceptos relevantes relacionados a resolver

problemas de optimización, tales como metaheurística, intensificación, diversificación, entre otros. Además, se presentan los objetivos específicos de este trabajo, los cuales apuntan a diseñar una estrategia de obtención de conocimiento para un sintonizador de parámetros con el objetivo de mejorar su desempeño, en términos de la calidad de las configuraciones obtenidas o del tiempo de ejecución.

En el capítulo 2, se presenta el marco conceptual relevante para el entendimiento del trabajo realizado. En primer lugar se introduce el problema de seteo de parámetros, para luego describir distintos sintonizadores de parámetros presentes en la literatura, tales como EVOCA, ParamILS, entre otros. Luego se introducen conceptos relacionados a los problemas de clasificación. Después, se presentan distintos tipos de clasificadores presentes en la literatura, además de su aplicación. Finalmente, se presentan casos en donde se ha utilizado distintas técnicas de obtención de conocimiento en el problema de seteo de parámetros.

En el capítulo 3, se presenta la propuesta de solución. Dentro del marco del problema del seteo de parámetros se optó por trabajar en una mejora para un sintonizador de parámetros, puntualmente EVOCA. EVOCA es un sintonizador de parámetros, el cual es en sí mismo un algoritmo evolutivo. EVOCA trabaja cada posible configuración como un individuo de una población, a la cual en cada iteración le aplica un operador de cruce y mutación. En este capítulo, se presenta un modelo de clasificación que será incorporado en el operador de mutación de EVOCA. Además, se presentan los detalles de implementación y el clasificador utilizado, Naive Bayes Classifier. Este clasificador utiliza el teorema de Bayes para determinar la probabilidad de pertenencia a cada clase, donde la clase que tiene la mayor probabilidad de pertenencia es la que se asigna a la nueva observación. A esta nueva versión de EVOCA se le dio el nombre de EVOCA+NBC.

En el capítulo 4, se valida la solución propuesta. Para lo cual, se plantea el escenario de sintonización y se muestran los resultados obtenidos. Se observó que con respecto a la calidad de las soluciones, EVOCA+NBC obtiene una mejor solución en el 63 % de las instancias y un mejor rendimiento promedio en aproximadamente el 49 % de las instancias. Por otro lado, con respecto al tiempo de ejecución de las soluciones, se obtiene un mejor resultado en el 56 % de las instancias y un mejor tiempo promedio en el 56 %. Además, se analiza la convergencia de las poblaciones, se evalúan y comparan poblaciones intermedias obtenidas por la propuesta y se analizan la importancia de las variables del modelo. De esto se ve que α y β son las variables más relevantes para el modelo. Finalmente, se realiza una discusión sobre los resultados obtenidos.

Para terminar, en el capítulo 5, se concluye sobre el trabajo realizado. Se concluye que utilizando un modelo de clasificación en EVOCA, se pudo obtener soluciones de mejor calidad. Además, se concluye que es posible obtener resultados de calidad comparable a los de EVOCA por sí solo en un menor número de evaluaciones. También, se menciona el aporte del trabajo realizado y se presentan sugerencias sobre el trabajo futuro.

CAPÍTULO 1
DEFINICIÓN DEL PROBLEMA

En la práctica, en cualquier actividad que se desarrolla los recursos son escasos, es por esto que se vuelve fundamental el uso apropiado de estos. El modelamiento de este tipo de situaciones se les conoce como problemas de optimización. Los problemas de optimización están presentes en varias disciplinas como en la ingeniería, servicios agrícolas, sistemas de manufacturación, economía, entre otros [Baghel *et al.*, 2012].

Algunos modelos de problemas de optimización sólo tienen sentido si las variables del problema son discretas, por ejemplo, el número de personas asignadas a una tarea en específico. Por otro lado, hay modelos en donde las variables tienen sentido que sean continuas, por ejemplo, el largo de una viga de metal. A los modelos con variables continuas se les conoce como problemas de optimización continua, mientras que a los modelos con variables discretas se les conoce como problemas de optimización discreta. Convencionalmente, a los problemas de optimización en el espacio discreto se les llama problemas de optimización combinatoria [Schrijver, 2005]. Formalmente, un problema de optimización combinatoria P se define como la tupla P=(S,f) [Blum y Roli, 2003], el cual es definido por los siguientes elementos:

- Un conjunto de variables X = $\{x_1, ..., x_n\}$
- El dominio de cada variable $D_1, ..., D_n$
- Restricciones a las que está sujeta cada variable
- Una función objetivo f que debe ser minimizada (o maximizada), en donde $f : D_1 \times \cdots \times D_n \rightarrow \mathbb{R}^+$

El espacio de búsqueda S, se define como el conjunto de los puntos o soluciones del problema de optimización que cumplen con las restricciones del problema. Una solución se refiere a una instanciación completa de las variables del problema, esto es, cada variable del problema se le asigna un valor, de tal manera que se respetan las restricciones del problema. De la misma forma, una instanciación incompleta se refiere a que solo a una parte de las variables del problema se les ha asignado un valor.

La manera más sencilla de resolver un problema de optimización es evaluar todas las posibles soluciones. Este enfoque recibe el nombre de búsqueda completa [Yang *et al.*, 2014]. Sin embargo, en problemas de optimización más complejos, es decir, donde hay un gran número de variables con una amplia gama de valores posibles para ellas, esta no es una opción factible, dado que el tamaño del espacio de búsqueda es muy grande y por ende hay una gran cantidad de soluciones posibles [Neumaier *et al.*, 2005]. Es por esto que comúnmente se emplean metaheuristicas o algoritmos de búsqueda local. El término metaheurística

se emplea para describir heurísticas de alto nivel que han sido propuestas para resolver un amplio rango de problemas de optimización. La motivación del uso de metaheurísticas se debe a que estas son capaces de obtener soluciones óptimas o cercanas a las óptimas, para problemas de optimización complejos, en tiempos razonables [Dokeroglu *et al.*, 2019].

De manera general, las metaheurísticas se pueden clasificar en dos categorías principales: las trayectoriales y las poblacionales. Las metaheurísticas que trabajan con tan solo una solución candidata a la vez se les conoce como trayectoriales. Por otro lado, a las que trabajan con un conjunto de soluciones candidatas a la vez se les conoce como poblacionales. Dependiendo de cómo trabaja con las soluciones se distinguen otras dos clases. Si el punto de partida es una posible solución y se trabaja sobre esta, se le conoce como reparadora. Al contrario, si se comienza con una instanciación incompleta o vacía para luego completarla, se le conoce como constructiva.

De manera adicional, dependiendo del comportamiento de las metaheurísticas, estas pueden ser clasificadas de la siguiente forma [Agrawal *et al.*, 2021]:

- Evolution based algorithms: Estos algoritmos están inspirados en la evolución natural. En estos algoritmos se inicia desde una población inicial, la cual tiene una descendencia. La descendencia sufre mutaciones para adaptarse y estas son traspasadas a sus nuevos descendientes. Un ejemplo de este tipo de algoritmo son los Genetic Algorithm [Holland, 1992], los cuales están inspirados en la teoría de Darwin.
- Swarm intelligence-based algorithms: Estos algoritmos están inspirador en el comportamiento social de insectos, animales, peces, aves, entre otros. Algunos ejemplos de estos algoritmos son Particle Swarm Optimization [Kennedy y Eberhart, 1995] y Ant Colony Optimization [Dorigo *et al.*, 1996].
- Physics based algorithms: Estos algoritmos están inspirados en las leyes físicas del universo. Dentro de este tipo se encuentran algoritmos como Simulated annealing [Kirkpatrick *et al.*, 1983] y Harmony search [Geem *et al.*, 2001].
- Human behaviour related algorithms: Estos son algoritmos que están inspirados en el comportamiento característico de los humanos. Un ejemplo de este tipo de metaheurísticas es League Championship algorithm [Kashan, 2009].

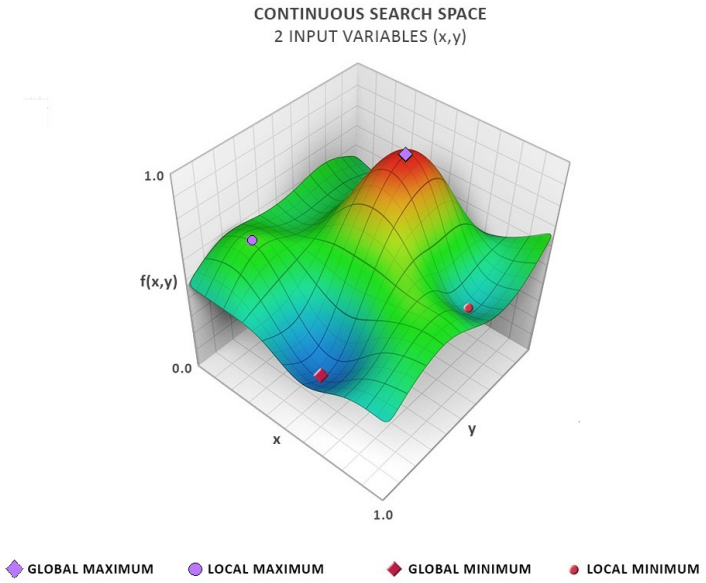


Figura 1: Espacio de búsqueda de dobles variables.
Fuente: Völcker, T. (2022). Continous Search Space. Tim Vöelcker. https://timvoelcker.de.

Para entender mejor los conceptos anteriores, es útil representar el espacio de búsqueda como un plano 3D, como se muestra en la figura 1, en donde el valor de la función objetivo f(x,y) está en el eje Z y las variables del problema están los ejes X e Y. Una posible solución en el espacio de búsqueda puede entenderse como un punto en la superficie. En el caso de las metaheurísticas reparadoras se comienza desde un punto de la superficie, por ejemplo, (1,2). Por otro lado, una metaheurística constructiva podría partir en el punto (1,y), en donde el valor y es determinado por la metaheurística. Las metaheurísticas poblacionales trabajan con un conjunto de puntos de la superficie a la vez, mientras que las trayectoriales solo con uno a la vez.

Otros conceptos importantes referentes a las metaheurísticas o algoritmos de búsqueda local son los de movimiento, vecindario, óptimo local y óptimo global. El movimiento es la operación que se ejecuta para modificar una instanciación completa, por ejemplo, sumar una cantidad constante α al valor de x y mantener fijo el valor de y. El vecindario son todos los puntos que se pueden alcanzar utilizando el operador de movimiento, por ejemplo, utilizando el movimiento que se describió anteriormente, si el punto de partida es (2,3) el vecindario son todos los puntos (2+α, 3). Finalmente, el óptimo global puede entenderse como el punto más alto de la superficie, cuando se trata de un problema de maximización, el cual está representado como un cuadrado púrpura en la figura, puesto que este tiene el mayor valor de la función objetivo en todo el espacio de búsqueda. Dado que las metaheurísticas no revisan todas las soluciones posibles comúnmente estas encuentran óptimos locales. Estos son puntos en donde el valor de la función es mayor comparativamente con sus alrededores en un subconjunto del espacio de búsqueda, en la figura se muestra como un círculo púrpura. A veces el objetivo es encontrar los puntos más bajos dentro del espacio de búsqueda,

cuando se trata de un problema de minimización, cambiando el objetivo a encontrar el mínimo global y mínimo local. Estos puntos están representados en la figura como un cuadrado rojo y punto rojo respectivamente.

Para terminar es necesario definir dos de las características más comunes y fundamentales de las metaheurísticas la intensificación y la diversificación. Estas características son consideradas como la piedra angular para resolver exitosamente problemas de optimización [Hussain *et al.*, 2019]. Diversificación se refiere al mecanismo que utiliza para explorar el espacio de búsqueda y visitar nuevas regiones de este, mientras que intensificación es el mecanismo para enfocarse en regiones atractivas del espacio de búsqueda, las cuales se identifican utilizando la experiencia acumulada en la búsqueda. Es importante el balance de estas para identificar rápidamente regiones de interés en el espacio de búsqueda y no perder mucho tiempo en las regiones poco prometedoras del espacio o las que ya han sido visitadas anteriormente. A pesar de lo anterior, no existe una guía clara para encontrar este balance ambas y esencialmente cada metaheurística tiene un balance distinto.

El balance entre intensificación y diversificación está comúnmente ligado a los valores de los parámetros de cada metaheurística. Dado que las instancias de un problema de optimización pueden ser muy diversas, no es factible utilizar la misma configuración de parámetros para instancias del mismo problema con diferentes características. El problema de encontrar los valores adecuados para estos parámetros, de tal manera que la metaheurística tenga un buen desempeño en la instancia que se está trabajando, se le conoce como el problema de seteo de parámetros. Para abordar este problema se han propuesto diversos enfoques, dentro de los cuales se encuentran los métodos de control de parámetros y los métodos de sintonización de parámetros. Estos conceptos se abordan en más detalle en el siguiente capítulo.

1.1. Objetivos

Objetivo General

Diseñar una estrategia de obtención de conocimiento para un sintonizador de parámetros con el objetivo de mejorar su desempeño, en términos de la calidad de las configuraciones obtenidas o del tiempo de ejecución.

Objetivos Específicos

- Estudiar sintonizadores de parámetros existentes en la literatura.
- Estudiar algoritmos de clasificación existentes en la literatura.

- Implementar una estrategia de obtención de conocimiento en el sintonizador escogido.
- Definir un escenario de sintonización de parámetros y evaluar la estrategia diseñada.
- Comparar el desempeño del sintonizador con otros sintonizadores de la literatura.

CAPÍTULO 2
MARCO CONCEPTUAL

El uso de las metaheurísticas para resolver problemas de optimización suele introducir un problema adicional, la selección de valores para los parámetros que utilizan, los cuales están fuertemente relacionados con su rendimiento [Montero *et al.*, 2014]. Algunos de estos parámetros son, por ejemplo, el tamaño de la población inicial de un Genetic Algorithm, la temperatura inicial de un algoritmo de Simulated Annealing o el tamaño de la lista tabú en un algoritmo de Tabu Search. Las principales dificultades relacionadas con este problema son: (1) Es una tarea que consume mucho tiempo, (2) los mejores valores para los parámetros dependen del problema con el que se está trabajando[Wolpert y Macready, 1997], (3) el valor de los parámetros suele variar a medida que se ejecuta el algoritmo[Back, 1992] y (4) la interacción entre los distintos parámetros es compleja [Rojas *et al.*, 2002] [Corriveau *et al.*, 2016]. Además, cabe señalar que dado la naturaleza estocástica de las metaheurísticas, ejecuciones con el mismo conjunto de parámetros pueden tener diferentes resultados al cambiar la semilla que este utiliza. [1]

El aumento en el uso y en la accesibilidad de la tecnología, ha provocado que continuamente se generen datos en escalas cada vez más grandes. Enormes conjuntos de datos son recolectados y estudiados en diversos campos como las redes sociales, comercio, seguridad, entre otros [Qiu *et al.*, 2016]. Múltiples técnicas de extracción de conocimiento han sido desarrolladas para obtener información oculta, a simple vista, de estos grandes volúmenes de datos. Es por esto que nace la motivación de explorar el uso de un método de obtención de conocimiento para el apoyo en la búsqueda de los parámetros adecuados para las metaheurísticas.

En este capítulo se presenta inicialmente el problema de seteo de parámetros, para luego presentar algunos métodos de sintonización de parámetros. Luego serán introducidos conceptos referentes a un problema de clasificación que serán empleados en el resto de la memoria. Después se presentan métricas comúnmente utilizadas para evaluar modelos de clasificación, además de detallar distintos modelos de clasificación presentes en la literatura. Finalmente, se explora cómo se han implementado estrategias de obtención de conocimiento para el problema de seteo de parámetros.

2.1. Problema del Seteo de Parámetros

El problema de encontrar el mejor conjunto de parámetros (o configuración) para que un algoritmo tenga el mejor desempeño empírico en un conjunto de instancias de un problema de optimización se le conoce como el problema del seteo de parámetros (PSP). Formalmente

[1] Una semilla es un valor que condiciona las secuencias aleatorias del algoritmo. Un algoritmo con un componente aleatorio si se ejecuta varias veces con la misma semilla arroja el mismo resultado.

se define como:

Dado:

- Un algoritmo A con parámetros $p_1, ..., p_k$
- Un espacio C de configuraciones, en donde cada configuración $c \in C$ asigna un valor v_k a cada parámetros p_k
- Un conjunto de instancias del problema I
- Una métrica m que mide el rendimiento de A en un conjunto de instancias I para una configuración c

El objetivo del PSP es encontrar $c^* \in C$ el cual tenga el rendimiento óptimo de A en I con respecto a la métrica m.

Los parámetros que usan estas metaheurísticas pueden ser de dos tipos:

- Parámetros categóricos: Estos son procedimientos o funciones del algoritmo que pueden ser implementados de diferentes formas. Por ejemplo, el tipo de método de selección para algoritmos evolutivos.
- Parámetros numéricos: Estos son valores enteros o reales. Por ejemplo, el tamaño de la población para un algoritmo evolutivo.

Para combatir las problemáticas antes descritas, las metodologías que se han utilizado se pueden clasificar como sintonización de parámetros o control de parámetros. La sintonización de parámetros puede ser descrita como el proceso de buscar valores adecuados para los parámetros es un paso anterior a la ejecución del algoritmo. Estos parámetros se mantienen fijos durante la ejecución del algoritmo. Por otro lado, los métodos de control de parámetros definen y modifican los valores de los parámetros del algoritmo durante su ejecución. **En esta memoria se trabaja con métodos de sintonización.**

2.2. Métodos de Sintonización de parámetros

Los métodos de sintonización de parámetros son estrategias diseñadas para buscar de manera automática la mejor configuración de parámetros para la metaheurística que se desee utilizar. Estos métodos pueden ser clasificados como:

- Hand-made Tuning: En este caso el diseñador de la metaheurística estudia el rendimiento del algoritmo en un conjunto de instancias del problema, utilizando valores de los parámetros escogidos bajo su propio criterio. Dependiendo de los resultados que obtenga puede decir cambiarlos o no. Con esto, se definen los valores considerando la sugerencia de los autores.
- Tuning by analogy: A diferencia del caso anterior, en vez de seguir un criterio propio la idea es seguir las pautas que han sido reconocidas por otros autores a raíz de su propia experimentación.
- By statistical analysis: En este caso, se utilizan métodos estadísticos para detectar relaciones entre el valor de los parámetros y el rendimiento del algoritmo. La finalidad es determinar modelos que predigan el rendimiento de un algoritmo utilizando análisis estadísticos para detectar diferencias significativas en los resultados del algoritmo.
- Search based tuning: Este tipo se basa en usar estrategias de búsqueda para resolver el problema de *seteo* de parámetros, tales como *tabu search*, algoritmos evolutivos, entre otros.

Antes de utilizar algún método de sintonización es necesario definir un escenario de sintonización, puesto que se debe precisar bajo qué condiciones los resultados de la sintonización son válidos, además del objetivo que se busca alcanzar. Un escenario de sintonización está compuesto por: (1) un conjunto de instancias de entrenamiento/training que se han escogido para sintonizar los parámetros del algoritmo objetivo; (2) un conjunto de instancias de prueba/testing para evaluar el desempeño del algoritmo objetivo con los parámetros que entrega el sintonizador; (3) la métrica (m) del desempeño del algoritmo , la cual puede enfocarse en el tiempo de ejecución del algoritmo objetivo, la calidad de las soluciones que arroja, entre otros; (4) la cantidad de recursos disponibles, el cual puede ser expresado en cantidad total de evaluaciones o el tiempo de ejecución.

Los escenarios de sintonización son útiles para comparar distintos sintonizadores o bien variaciones de diseño del mismo sintonizador. Por ejemplo, durante su ejecución, al comparar dos posibles calibraciones, se puede decidir descartar la de peor rendimiento, perdiendo el tiempo que se utiliza en evaluarla. Sin embargo, otro sintonizador, ante el mismo escenario decide mantener la peor configuración y darle otro uso más adelante, ocupando más recursos almacenando configuraciones que en el caso anterior. Es difícil determinar a priori cual de estas dos alternativas tendrá un mejor rendimiento y bajo qué condiciones. Para esto, es necesario una evaluación de ambas en un mismo escenario de sintonización.

2.2.1. Evolutionary Calibrator

Evolutionary Calibrator (EVOCA) es un método de sintonización de parámetros, el cual es en sí mismo un algoritmo evolutivo. En EVOCA cada individuo de la población representa

una posible configuración de parámetros para el algoritmo objetivo. Un diagrama de flujo de su comportamiento se muestra en la figura 4. A continuación se explicará en términos generales su funcionamiento para después detallar los distintos componentes de EVOCA. Como se muestra la figura, en primer lugar se genera una población inicial. El tamaño de la población se calcula dependiendo del número de parámetros que se desea sintonizar y el tamaño de sus dominios. Para obtener un conjunto inicial de valores bien distribuidos para cada parámetro se utiliza Latin Hypercube design. Latin Hypercube design es un método para obtener muestras aleatorias de un conjunto de datos. Este método consiste en dividir el dominio de las variables en intervalos iguales y elegir un valor dentro de cada intervalo. En la figura 2 se muestra un ejemplo para dos variables x e y, en donde se desea tomar 25 muestras. En esta figura se ha dividido la región en 25 partes iguales. El muestreo se termina cuando se escogen al azar 25 puntos, uno en cada subdivisión. En la figura se ve como se han escogido hasta el momento 5 muestras.

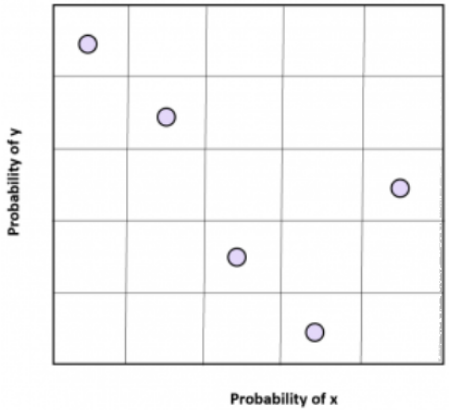


Figura 2: Ejemplo Latin Hypercube para dos variables.
Fuente: Zach.(2022). Two-Dimensional Latin Hypercube Sampling. Statology. https://www.statology.org/.

Posteriormente se inicia el ciclo principal del algoritmo, el cual se termina al completar una cantidad determinada de evaluaciones o bien al transcurrir un intervalo de tiempo indicado por el usuario. Una vez obtenida la población inicial se aplica el operador de cruzamiento, Wheel Crossover, sobre la población. El operador de cruzamiento Wheel Crossover busca incorporar la información de la población actual en un nuevo individuo. Para esto toma características de distintos individuos de la población y las combina en uno nuevo. Por ejemplo, para sintonizar un algoritmo objetivo con dos parámetros (a,b) y se tiene una población de tamaño tres, entonces se tienen tres configuraciones: $P_1 = (a_1, b_1)$, $P_2 = (a_2, b_2)$, $P_3 = (a_3, b_3)$. Con esta población, se crea una ruleta como se muestra en la figura 3 de forma simbólica, en donde la probabilidad de elección de cada alternativa depende de la calidad de cada configuración. De la figura se desprende que P_1 es la mejor configuración de la población, dado que cubre un mayor área. Para fabricar el nuevo individuo de la población este hereda parámetros de las configuraciones en la población, lo cual se hace manera aleatoria n veces (utilizando la distribución de probabilidad que entrega la ruleta), donde n es el número de

parámetros a sintonizar. En el caso del ejemplo, $n = 2$ por lo que se escogen dos individuos al azar. Si se escoge primero P_1 y luego P_3 la configuración del nuevo individuo sera (a_1, b_3).

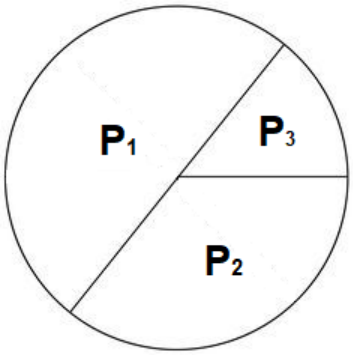


Figura 3: Ruleta para wheel crossover
Fuente: Elaboración propia.

El nuevo individuo resultante, individuo cruzado, se evalúa y reemplaza al peor individuo de la población. La evaluación de las configuraciones es de manera empírica, esto es, se ejecuta el algoritmo objetivo con los parámetros del individuo cruzado y se mide qué tan cercano es el resultado del algoritmo objetivo con respecto al óptimo conocido de la instancia con la que se ejecuta. Dado la estocasticidad del proceso, se evalúa un conjunto de instancias con semillas distintas para tener un resultado más representativo. El número de pares (semilla,instancia) que se evalúan es un hiperparámetro de EVOCA. Terminada la evaluación del individuo cruzado se le aplica el operador de mutación Hill Climbing para obtener el individuo mutado[2]. EVOCA solamente revisa el vecindario del punto inicial hasta encontrar una configuración mejor. Para determinar el vecindario, en primer lugar se selecciona un parámetro de la configuración de forma aleatoria. Al reemplazar el valor del parámetro seleccionado por otro valor aleatorio dentro del rango de su dominio se obtiene el vecindario. Finalmente, el individuo mutado es evaluado y es comparado con el individuo cruzado, en el caso de ser mejor reemplaza al segundo peor individuo de la población y comienza otro ciclo del algoritmo aplicando el operador de cruzamiento. En el caso contrario vuelve a aplicar el operador de mutación. Esto se repite hasta que el número de vecinos que han sido visitados sea mayor al tamaño del dominio del parámetro mutado. El tamaño del dominio de los parámetros se calcula en función de los límites de su dominio, los cuales son entregados por el usuario, según la siguiente ecuación: tamaño del dominio $=$ límite superior $-$ límite inferior $+ 1$

2.2.2. Parameter iterated local search

Parameter iterated local search (ParamILS) funciona como un algoritmo de búsqueda local iterativa. En el algoritmo 1 se muestra la estructura de ParamILS. Este empieza desde una

[2]El operador de mutación Hill Climbing es una metaheurística trayectorial que se utiliza con el fin de mejorar la configuración resultante del operador de cruzamiento.

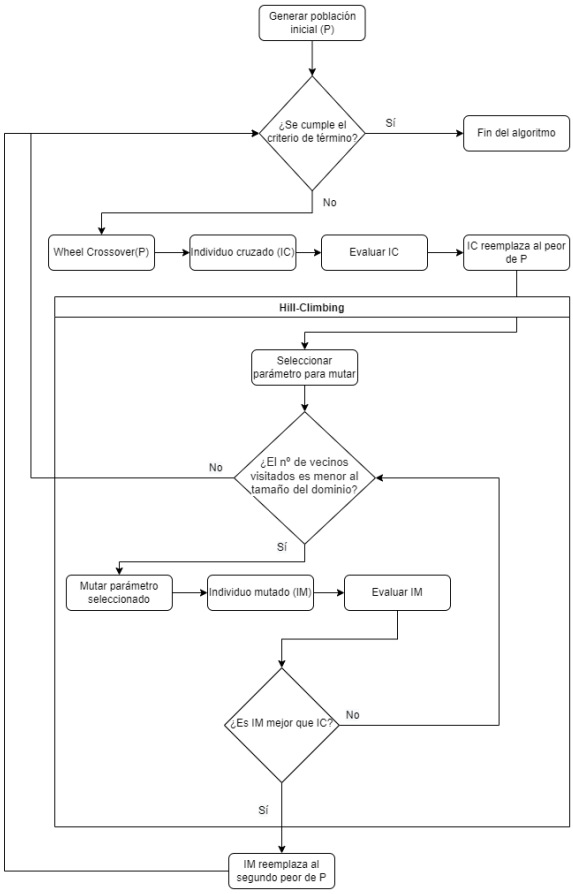


Figura 4: Diagrama de flujo de EVOCA.
Fuente: Elaboración propia .

calibración entregada por el usuario, para luego intentar R veces una mejor de forma aleatoria. La calibración resultante se le aplica el operador *IterativeFirstImprovement* (línea 8). Este operador busca de manera aleatoria en el vecindario de la calibración por una de mejor calidad. El vecindario de la calibración es creado al cambiar de forma aleatoria un parámetro de la calibración.

En cada iteración, desde la línea 9 hasta la 21, ParamILS ejecuta s perturbaciones aleatorias a la calibración y la mejor calibración resultante es luego mejorada por el operador *IterativeFirstImprovement*. El rendimiento de esta calibración se compara con la mejor calibración encontrada hasta el momento. Se utiliza el concepto de dominación para comprar calibraciones. Una calibración c domina a la calibración c', si y sólo si, el rendimiento promedio del algoritmo objetivo, en las instancias de entrenamiento, utilizando c en n semillas es mejor que el rendimiento promedio de c' utilizando n' semillas, donde n>n'. Al terminar esta comparación, existe una probabilidad de iniciar la búsqueda de nuevo $p_{restart}$, el cual permite al algoritmo escapar de los óptimos locales.

Algorithm 1 ParamILS

```
1:  c_0 ← default calibration
2:  for i ← 1 to R do
3:      c ← random calibration
4:      if better(c,c_0) then
5:          c_0 ← c
6:      end if
7:  end for
8:  c_ils ← IterativeFirstImprovement(c_0)
9:  while maximum budget is not reach do
10:     c ← c_ils
11:     for i ← 1 to s do
12:         c ← random calibration
13:     end for
14:     c ← IterativeFirstImprovement(c)
15:     if better(c,c_ils) then
16:         c_ils ← c
17:     end if
18:     if p_restart then
19:         c_ils ← random calibration
20:     end if
21: end while
22: return c
```

2.2.3. Sequential Model-based Algorithm Configuration

En [Hutter *et al.*, 2010] se presenta Sequential Model-based Algorithm Configuration (SMAC), el cual en términos simples construye un modelo para predecir el rendimiento del algoritmo objetivo. El modelo es utilizado posteriormente para seleccionar configuraciones de parámetros prometedores para el algoritmo. Finalmente, para determinar la mejor configuración del conjunto, SMAC utiliza un mecanismo de intensificación. Aquí, (1) se comparan pares de configuraciones, (2) se determina cuántas evaluaciones se llevarán a cabo para cada configuración (con el objetivo de) y (3) definir cuándo configuración es mejor que la best-so-far.

SMAC utiliza un conjunto de modelos de Regression Trees (RT), más formalmente llamado Random Forest (RF). RF es considerado un método de ensamblado (técnica que combina predicciones de un conjunto de modelos para obtener una predicción más precisa que un modelo por sí solo). Dado que la estructura de los árboles cambia dependiendo de los datos utilizados, RF busca reducir la varianza entre los modelos, utilizando sub-muestras de datos repetidos de la muestra original de datos para construir los distintos árboles.

El modelo de SMAC busca predecir el rendimiento del algoritmo dado un conjunto de pa-

rámetros que se utilizan, pudiendo considerar ciertas características de las instancias en la que se ejecuta el algoritmo. Una vez construido el modelo, configuraciones candidatas se evalúan en el, después se calcula σ y μ promedio de los RT que conforman el RF. Estos estadísticos se utilizan para calcular un Expected Improvement (EI), el cual es una medida de qué tan atractivas son estas configuraciones. Se selecciona un subconjunto de configuraciones con el mayor EI y se hace una búsqueda local con cada una de ellas maximizando EI. Una vez terminadas las búsquedas locales, las configuraciones prometedoras se ponen a prueba con un mecanismo de intensificación. Este mecanismo compara pares de configuraciones de forma empírica y determina cuales son mejores considerando un presupuesto de tiempo definido. Finalmente, una vez **completado** un batch de configuraciones, la información sobre las configuraciones evaluadas se utiliza para actualizar el modelo.

2.2.4. Iterated F-Race

Algorithm 2 I-race

$Set_1 \leftarrow$ User candidate set()
$I = 2 + round(\log_2 p)$
$i \leftarrow 1$
while $budget_{used} \leq maximun\_budget$ **do**
    $budget_i = (maximun\_budget - budget_{used}) \backslash (I - i + 1)$
    $N_i = \lfloor budget_i \backslash \mu_i \rfloor$
    $Set_{new} \leftarrow Sample(\Theta, \Theta_{elite})$
    $Set_i \leftarrow Set_{new} \cup Set_{elite}$
    $Set_{elite} \leftarrow Race(Set_i, budget_i)$
    $budget_{used} = budget_{used} + budget_i$
    $i = i + 1$
end while
return Set_{elite}

Iterated F-Race (I-Race) es una versión iterativa del algoritmo F-race. I-race consiste en tres pasos: (1) muestrear nuevas configuraciones de acuerdo a una distribución normal d-variada para una cantidad d de parámetros[3], (2) seleccionar las mejores configuraciones de la muestra anterior utilizando F-race y (3) actualizar la distribución para sesgar la muestra de nuevas configuraciones candidatas hacia las mejores configuraciones. Estos tres pasos se repiten hasta alcanzar el criterio de término.

En cada iteración F-race es utilizado para determinar la calibración con el mejor rendimiento estadístico. F-race utiliza Friedman Two-ways analysis of variance (ANOVA) para comparar configuraciones de parámetros, dado esto no se hacen suposiciones de las distribuciones del rendimiento de las configuraciones.

[3]Cada parámetro tiene su propia distribución normal dada por $\mu = (\mu_1, \ldots, \mu_d)$ y $\sigma = (\sigma_1, \ldots, \sigma_d)$

En el algoritmo 2 muestra la estructura de I-race. I-race calcula el número de iteraciones que va a ejecutar mediante la fórmula $I = 2 + round(\log_2 p)$, en donde p es el número de parámetros (línea 2). Por otra parte, la cantidad de recursos que se utilizan dentro de cada iteración se calcula como $budget_i = (maximun\_budget - budget_{used})/(I - i + 1)$ (línea 5), en donde $budget_{used}$ son los recursos que se han utilizado hasta el momento y *maximun_budget* la cantidad máxima de recursos que se utilizan, el cual es definido por el usuario.

En cada iteración, $N_i = \lfloor budget_i/\mu_i \rfloor$ configuraciones candidatas son muestreadas, donde μ_i es un factor que se incrementa con el número de iteraciones (línea 6), el cual no está relacionado con la media μ. En la primera iteración, el conjunto de calibraciones candidatas es muestreado al azar de forma uniforme o es definido por el usuario al inicio. Las ejecuciones de F-race pueden terminar si a lo más quedan $N_{min} = 2 + round(\log_2 p)$ configuraciones candidatas en el conjunto. Una vez que termina F-race las mejores N_{elite} calibraciones son elegidas, se les asigna un peso de acuerdo al resultado de la carrera y son usadas para actualizar la distribución actual.

2.3. Conceptos previos

El primer concepto a definir es el de problema de clasificación. En un problema de clasificación se tiene un conjunto de datos no etiquetados, es decir, no se sabe si pertenecen a un subconjunto determinado (clase) y se desea predecir la clase a la que pertenecerán dada sus características (atributos o variables). Por ejemplo, en la figura 5 se muestran algunos datos de personas que tienen una tarjeta de crédito de un banco y el banco quiere estimar con esta información si un futuro cliente pagará o no la deuda de la tarjeta de crédito, para así minimizar sus pérdidas. En este caso hay dos clases (pago o no pago), donde se indica con un 1 si no pudo pagar y un 0 en el caso contrario. Los atributos de la persona son todas las características que se sepan de ella, en este caso tales como su sexo (SEX), edad (AGE), su cupo máximo de crédito (LIMIT_BAL), educación (EDUCATION), estado civil (MARRIAGE), entre otros. Los atributos pueden ser categóricos, es decir, que tienen dos o más valores, pero que no existe un orden establecido, por ejemplo, el estado civil. Por otro lado, los atributos pueden ser numéricos, esto es, que variables que tienen dos o más valores posibles, pero en este caso sí existe un orden, por ejemplo, la edad.

Un modelo de clasificación se construye en base a la información (observaciones) que se tiene disponible del fenómeno que se quiere modelar. En el ejemplo anterior, el banco tiene información de todas las personas que les ha dado una línea de crédito y si les ha pagado o no una vez llegada la fecha acordada. Dado que la información etiquetada es escasa, los datos disponibles se suelen dividir en datos de entrenamiento que son utilizados para crear el modelo y datos de testeo para medir el rendimiento del modelo. Al modelo se le entregan los datos de testeo, de los cuales se conoce su clase real, sin embargo para el modelo es una observación nueva y debe predecir su clase.

Un problema que tienen los datos del mundo real es que estos incluyen datos irrelevantes o

LIMIT_BAL	SEX	AGE	EDUCATION	MARRIAGE	default.payment.next.month
20000.0	2	24	2	1	1
120000.0	2	26	2	2	1
90000.0	2	34	2	2	0
50000.0	2	37	2	1	0
50000.0	1	57	2	1	0

Figura 5: Fragmento de dataset: Default Payments of Credit Card Clients in Taiwan from 2005. Fuente: Elaboración propia.

carentes de sentido. A datos de estas características se les llama “ruidosos”, dado que afectan significativamente de forma negativa las tareas de obtención de conocimiento como lo es un problema de clasificación [Gupta y Gupta, 2019]. El ruido puede estar presente tanto en los atributos (attribute noise) o en la clase (class noise).

Class noise El ruido en la clase hace referencia a cuando a la observación se le asigna la clase equivocada. Esta situación puede darse por un simple error o cuando existen dos observaciones con los mismos atributos pero con distintas clases.

Attribute noise El ruido en los atributos se refiere a cuando uno o más atributos tienen valores erróneos. Esta puede deberse a que el valor tenga un error, por ejemplo una edad negativa, el valor del atributo no se conoce y finalmente el atributo puede no aportar información útil.

Una vez que se tiene el modelo entrenado es necesario someterlo a prueba para evaluar su desempeño. Para esto se utilizan los datos de testeo, los cuales se le entregan al modelo sin su clase y la idea es que este haga una predicción sobre esta. Se puede dar el caso que el modelo tiene un buen desempeño en predecir la clase de los datos que se utilizaron para entrenarlo, pero que con datos de testeo tenga un muy mal desempeño, es decir, el modelo tiene baja capacidad para generalizar (overfitting). A continuación se presentarán diversas métricas existentes para evaluar modelos.

2.4. Métricas de Evaluación del Modelo

En general, las métricas de evaluación pueden ser descritas como una herramienta para medir el desempeño de un modelo clasificador. Distintas métricas evalúan distintas características del clasificador [Hossin y Sulaiman, 2015]

	actual	
	+	−
predicted+	*TP*	*FP*
predicted−	*FN*	*TN*

Figura 6: Matriz de confusión.
Fuente: [Brodersen *et al.*, 2010].

En el caso de clasificación binaria, esto es, solo hay dos clases una de ellas se le suele llamar la clase positiva (ocurre el fenómeno) y a la otra la clase negativa (no ocurre el fenómeno). En base a esto se define los true positive (tp) y los true negative (tn), la cual es la cantidad de observaciones que fueron clasificadas correctamente como positivas y como negativas respectivamente. Por otro lado, están los false positive (fp) y los false negative (fn), esto es, la cantidad de observaciones que fueron clasificadas de forma equivocada como positivas y como negativas respectivamente. Visualmente, estos conceptos se resumen en la matriz de confusión, como se muestra en la figura 6.

2.4.1. Accuracy

Accuracy (acc) mide la razón de predicciones correctas del total de evaluaciones.

$$acc = \frac{tp + tn}{tp + tn + fp + fn}$$

2.4.2. Error Rate

Error rate es la razón de las predicciones incorrectas del total de evaluaciones.

$$err = \frac{fp + fn}{tp + tn + fp + fn}$$

2.4.3. Precision

Precisión (p) es usada para medir las predicciones positivas que fueron clasificadas correctamente del total de predicciones positivas.

$$p = \frac{tp}{tp + fp}$$

2.4.4. Recall

Recall (r) es usada para medir la razón de predicciones positivas que están correctamente clasificadas.

$$r = \frac{tp}{tp + tn}$$

2.4.5. F-Measure

F-Measure (FM) o F-score es una medida que balancea de igual forma r y p y calcula la media armónica. [Sokolova *et al.*, 2006].

$$FM = \frac{2 * p * r}{p + r}$$

2.5. K-fold cross validation

Un enfoque alternativo para evaluar un modelo es a través de k-fold cross validation (K-Fold). En la figura 7 se muestra un ejemplo de 4-Fold. El conjunto de datos disponibles se divide en k subconjuntos, $k = 4$ en este caso. En cada iteración se toma uno de estos subconjuntos y se usa como datos de prueba y el resto como datos de entrenamiento. Se evalúa el modelo con alguna métrica y se repite el proceso k veces. Finalmente, se toma el promedio de estas métricas y se tiene el desempeño del modelo.

Un caso particular es el Leave One Out cross-validation, en el cual se hacen $K = n$ divisiones, en donde n es el número de observaciones.

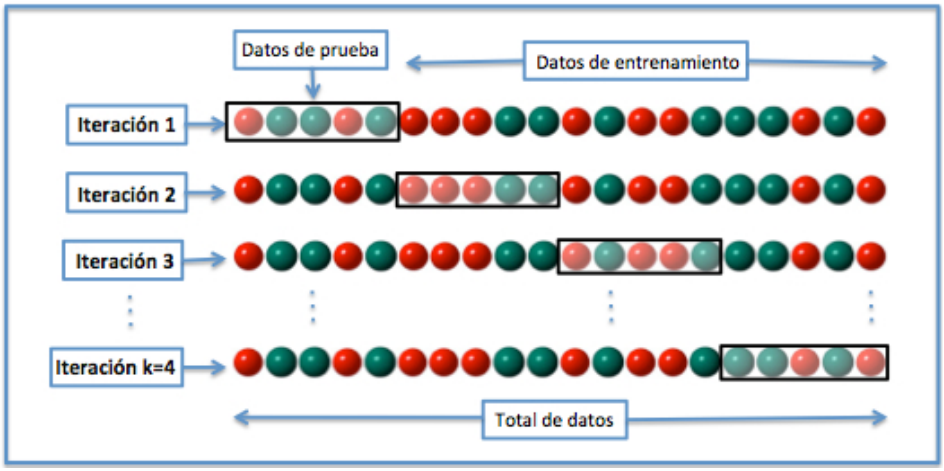


Figura 7: Ejemplo de 4-fold cross-validation. .
Fuente: Refaeilzadeh, P.(2022). Validación cruzada de K iteraciones con K=4. Wikipedia. https://es.wikipedia.org.

2.6. Modelos de clasificación

A continuación se presentan distintos modelos de clasificación que se encuentran en la literatura, además de un ejemplo de su aplicación.

2.6.1. K-nearest neighbors

K-nearest neighbors (KNN) es un clasificador que usa la noción de distancia para clasificar nuevas observaciones. Se encuentran las k observaciones más cercanas y la nueva observación se clasifica con la clase de de la mayoría de estos k ”vecinos”. En la figura 8 se muestra un ejemplo con 1 vecino. En este caso el nuevo dato en cuadro amarillo su vecino mas cercano es una estrella y por lo tanto su clase sera estrella.

La distancia euclidiana es la más utilizada, sin embargo también hay otros métodos de calcular la distancia como la distancia de Manhattan [Zhang, 2016].

Un aspecto importante de este algoritmo es la elección de k, en donde algunos autores sugieren la raíz cuadrada del número de datos de entrenamiento.

En [Amra y Maghari, 2017] se utilizó KNN para predecir el rendimiento de estudiantes (A,B,C,D,F). Se utilizó un conjunto de 500 datos con 8 atributos, como por ejemplo género del estudiante y la profesión de sus padres. Los datos fueron divididos en 70 % para entrenar y 30 % para testear, con esto el modelo obtuvo un 63.45 de precisión. Se destaca la simpleza del algoritmo y su mayor limitante es el uso de memoria, puesto que debe guardar en memoria todos los datos de entrenamiento para comparar su distancia con la nueva evaluación.

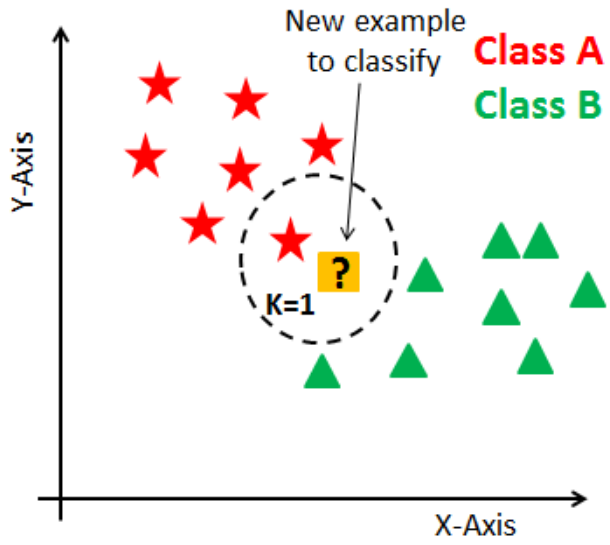


Figura 8: Ejemplo de 1-nn.
Fuente: Leuschke, A.(2022). KNN Classification using Scikit-learn. Morioh.
https://morioh.com.

2.6.2. Naive bayes Classifier

El Naive bayes Classifier (NBC) es una de las herramientas más antiguas en machine learning [Berrar, 2018]. El teorema de Bayes puede ser usado para derivar la probabilidad a posteriori de una hipótesis dado un número de observaciones, esto es:

$$P(hipotess \mid datos) = \frac{P(datos \mid hipotesis) * P(hipotesis)}{P(datos)}$$

Donde P(datos| hipótesis) es la verosimilitud de los datos (si la hipótesis es verdadera, entonces ¿cuál es la probabilidad de observar estos datos?), P(hipótesis) es la probabilidad a priori de la hipótesis y P(datos) es la probabilidad de observar los datos. Para el caso de NBC se utiliza un modelo estadístico Bayesiano para obtener una probabilidad de que un dato pertenezca a una clase dado las observaciones anteriores.

Uno de los problemas que tiene NBC es que si el número de datos es pequeño es difícil estimar la probabilidad de que ocurra cada uno de los atributos de las clases que se están analizando. Por otro lado, se trabaja con el supuesto que los atributos de una clase son independientes, lo cual es un supuesto muy fuerte. Sin embargo, a pesar de esto último ha mostrado buen desempeño aun cuando los atributos no son independientes [Berrar, 2018].

En [Mukherjee y Sharma, 2012] se utilizó NBC para clasificar eventos que podrían ser mali-

ciosos dentro de un computador. Las posibles clases de eventos son: DoS (denial of service), U2R (user to root), R2L (remote to local) y Probe (information gathering). En este trabajo se analizó el rendimiento de este algoritmo usando diferentes métodos de selección de variables: Correlation-based Feature Selection (CFS), Information Gain (IG) and Gain Ratio (GR). Además, se utilizó un conjunto de 62.986 datos y se utilizó 10-fold cross validation, obteniéndose así un 97.78 de precisión. Finalmente, menciona que para que tenga un buen resultado NBC es necesario de datos sin mucho ruido.

2.6.3. Support Vector Machine

Support Vector Machines (SVM) es un tipo específico de algoritmos de machine learning, el cual es muy utilizado en muchos problemas de aprendizaje estadístico como clasificación de texto, filtros de spam, reconocimiento de caras y objetos, entre otros [Bhavsar y Panchal, 2012]. La noción geométrica de las SVM para clasificación es un algoritmo que busca la separación óptima de una superficie, a través de un hiperplano, el cual separa de manera equidistante a dos clases. En la figura 9 se muestra de forma gráfica esta situación en un plano 2D.

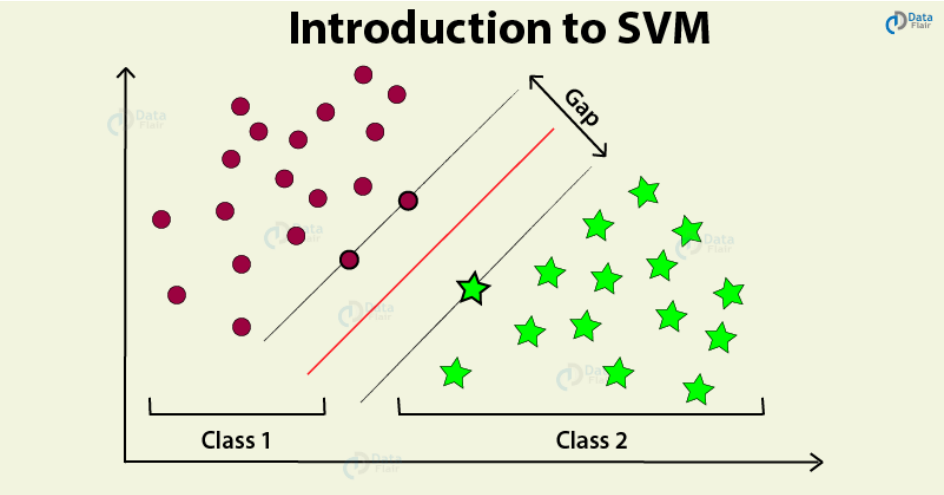


Figura 9: Interpretación geométrica de una SVM para clasificación.
Fuente: Data Flair https://data-flair.training.

En un principio las SVM fueron utilizadas para el caso en las clases fueran linealmente separables. Sin embargo, con el uso de kernels es posible utilizarlos en casos donde las clases no son linealmente separables. Los kernels son funciones que mapean los datos en un espacio distinto donde son linealmente separables. En la figura 10 se muestra que en un espacio 2D las clases no son linealmente separables, sin embargo al ser mapeadas en un espacio 3D existe un plano que separa ambas clases.

El kernel a utilizar depende del problema, el cual puede ser lineal, polinomial, Radial Basis Function (RBF), sigmoidal, entre otros. La mayor limitación de las SVM es la elección del

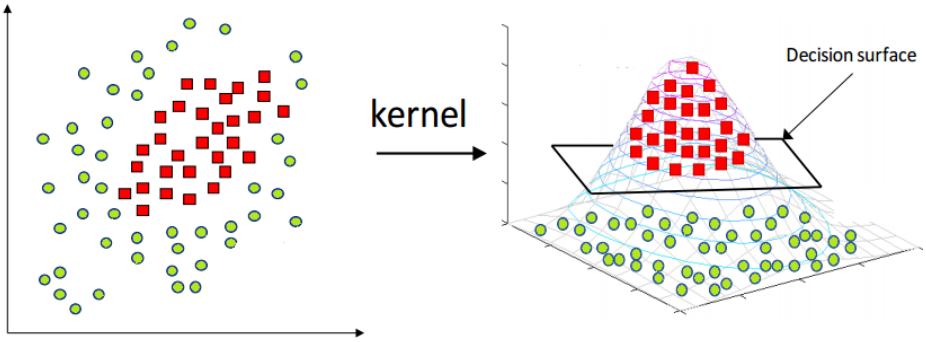


Figura 10: Ejemplo de uso de kernels.
Fuente: Sharma S.(2022). Kernel Trick. Medium. https://medium.com.

kernel puesto que este tiene un gran impacto en el rendimiento del algoritmo. Además en [Cervantes *et al.*, 2020] se muestra que un dataset desbalanceado tiene un gran efecto negativo en el desempeño.

En [Furey *et al.*, 2000] se utilizó para clasificar tejido y tipos de células utilizando un kernel de producto punto. Se utilizó un conjunto de datos de 31 observaciones de 97.802 atributos cada una, además se utilizó hold one out cross validation. Con respecto a los resultados, se encuentra que tiene un rendimiento similar a un perceptrón (red neuronal de una capa). Esto se atribuye a los pocos datos disponibles y se estima que a medida que se tengan data sets más completos y la complejidad aumente la SVM tendrá un mejor rendimiento.

2.6.4. Decision Trees

Un Decision Tree (DT) es un modelo de machine learning, el cual a través de una serie de preguntas acerca de los atributos de un dato busca llegar a una conclusión. En el caso de clasificación, la conclusión es la clase a la que pertenece una observación. En la figura 11 se muestran las partes de un DT. Existen tres tipos de nodos: nodo raíz, nodo interno o nodo de decisión y nodos hoja. Los nodos de decisión son nodos en donde se hace una pregunta sobre el valor de algún atributo de los datos. Desde los nodos de decisión salen ramas, las cuales marcan el camino a seguir dependiendo de la respuesta de la pregunta que se hizo. El nodo raíz es un nodo especial de decisión que marca el principio del DT. Finalmente, los nodos hojas son nodos terminales, es decir, desde ellos no salen ramas y tienen la predicción de la clase. Para clasificar una nueva observación se parte desde el nodo raíz y dependiendo de los valores de los atributos se siguen las ramas hasta llegar a una hoja.

En la figura 12 se muestra un ejemplo de DT. El funcionamiento general de un DT consiste en depositar todo el conjunto de datos de entrenamiento en la raíz, en la figura 12 se ve que en el nodo raíz hay 112 datos. Después, se selecciona uno de los atributos utilizando alguna métrica, el ancho del pétalo en este caso, y se reparten los datos en nuevos nodos

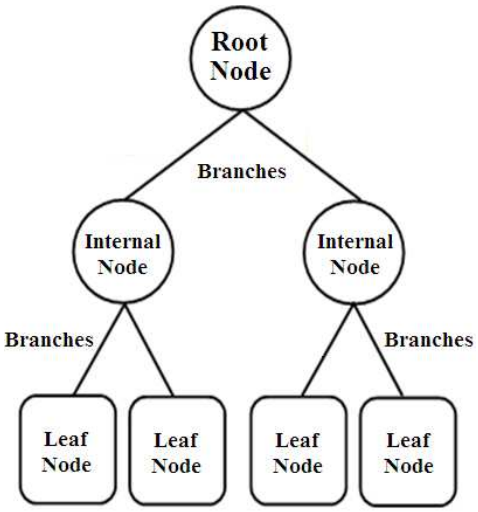


Figura 11: Partes de un DT.
Fuente: Silva J.(2022). Components of a decision tree. ResearchGate.
https://www.researchgate.net.

de decisión, dependiendo del valor del atributo seleccionado de cada dato. Formalmente, a esta acción se le denomina particionar los datos. En la figura, se observan los datos del nodo raíz de los cuales 37 tienen un ancho del pétalo menor o igual a 0.8[cm] y 75 mayor a 0.8[cm] . Posteriormente, para cada uno de los nuevos nodos, se ve si todos los datos del nodo son de la misma clase. De ser así se considera como una hoja (marcadas con verde en el ejemplo), en el caso contrario se vuelven a particionar los datos. Esto se repite hasta solo tener hojas o algún otro criterio dado por el algoritmo de construcción que se use.

Dependiendo del algoritmo utilizado para construir el DT los criterios para elegir los atributos para la partición cambian. A continuación se profundizan algunas de ellas:

Iterative Dichotomiser 3

Iterative Dichotomiser 3 (ID3) utiliza un criterio para seleccionar los atributos de los nodos de decisión considerando dos métricas: information gain (IG) y entropía. La entropía es la medida de impureza del nodo e IG es la diferencia de entropía entre el nodo con los datos antes de particionar (nodo padre) y la suma de entropía de los nodos con los datos repartidos por el atributo elegido (nodos hijos). El objetivo es maximizar IG. Cabe señalar que ID3 toma la mejor decisión a nivel local, lo cual no asegura encontrar el mejor DT posible. En la figura 13 se muestra un DT construido con el dataset mencionado en la sección 2.3.

Las ventajas que tiene ID3 es que es de construcción rápida, los árboles son cortos en términos de largo, solo necesita testear suficientes atributos hasta que todos los datos estén clasificados, crea reglas de decisión fáciles de entender y todo el conjunto de datos es explorado

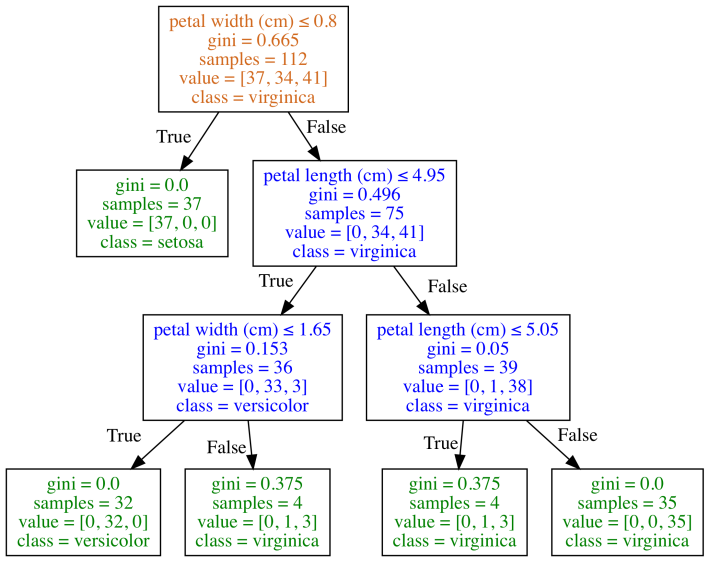


Figura 12: Ejemplo de DT para clasificar tipos de flores.
Fuente: Galarnyk M.(2022). Decision Trees for Classification. Towards Data Science. https://towardsdatascience.com.

para construir el árbol. En contraste, las desventajas que exhibe es que no puede manejar atributos numéricos o valores faltantes, solo un atributo es testeado a la vez para evaluar datos de prueba y los datos de prueba pueden sufrir de overfitting [Singh y Gupta, 2014].

En [Fitrani *et al.*, 2019] se utilizó ID3 para analizar la minutación de casos de tribunales. Se utilizaron 998 casos con 8 atributos del Tribunal de Distrito de Bangil, de los cuales 65 % fueron utilizados para entrenar y 35 % para testear. Se obtuvo un accuracy promedio de 93,41.

Classification and Regression Trees

A diferencia de ID3, Classification and Regression Trees (CART) utiliza el índice de Gini para seleccionar el atributo para particionar los datos. Además, realiza una poda del árbol, es decir, elimina una parte de este, con el fin de disminuir el overfitting. En la figura 14 se muestra un DT construido con el dataset mencionado en la sección 2.3.

Dentro de las ventajas de CART se encuentra que puede manejar fácilmente variables numéricas como categóricas, manejar outliers y el algoritmo en sí mismo es capaz de identificar las variables más significativas y eliminar las no significativas. Por otra parte, las desventajas de CART es que las particiones son de una variable y es sensible a los datos de entrenamiento, esto es, puede darse que al eliminar un conjunto de observaciones el árbol resultante puede ser completamente distinto [Singh y Gupta, 2014].

En [Zulfikar *et al.*, 2021] se utilizó para clasificar la factibilidad que aun alumno participe en

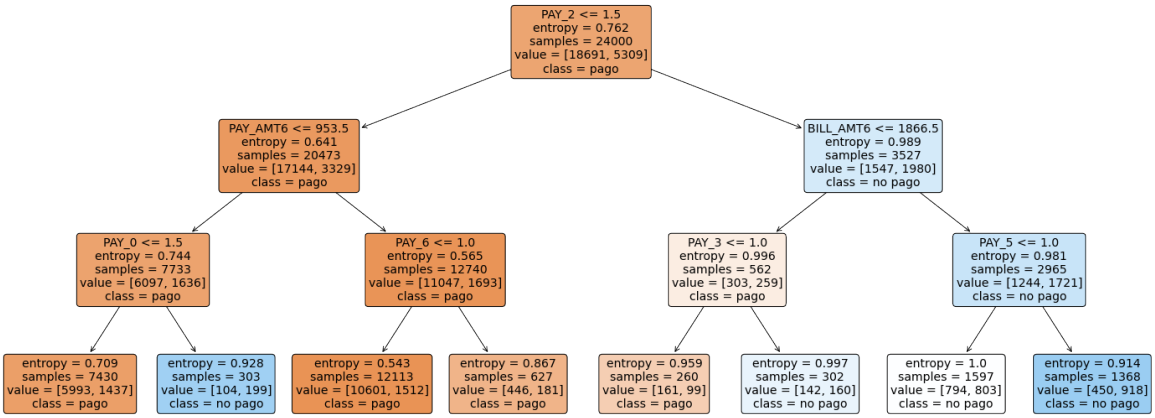


Figura 13: Ejemplo de DT construido con ID3
Fuente: Elaboración propia.

un programa de intercambio. Se utilizó un conjunto de 171 datos de 7 variables. Se probó el algoritmo con distintas proporciones de datos de entrenamiento y testeo 70/30, 50/50 y 40/60 (entrenamiento/testeo), obteniendo el mejor resultado con 50/50 con un recall del 95 %.

C4.5

C4.5 es la evolución de ID3, la cual elimina algunas de las desventajas de ID3. A diferencia de ID3 se utiliza gain ratio (GR), la cual es una versión normalizada de IG para seleccionar el atributo para particionar.

Las ventajas que tiene C4.5 son la siguientes: Puede manejar atributos discretos y continuos, soporta el uso de datos con atributos faltantes, una vez terminado el árbol intenta remover ramas que no ayudan y las reemplaza por hojas. Por el contrario, las desventajas que tiene son las siguientes: construye ramas vacías y es susceptible al ruido y outliers.

En [Budiman *et al.*, 2017] se utilizó C4.5 para predecir el tiempo de graduación de un alumno (fast time, on time, delay time). Se utilizaron 279 datos con 11 atributos. Se comparó el rendimiento del modelo con distintas proporciones de datos de entrenamiento y testeo, en incrementos de 10 % (90/10, 80/20, ...,10/90), obteniendo así el mejor desempeño para con 90 % datos de entrenamiento y 10 % datos de testeo llegando a una precisión del 78,58.

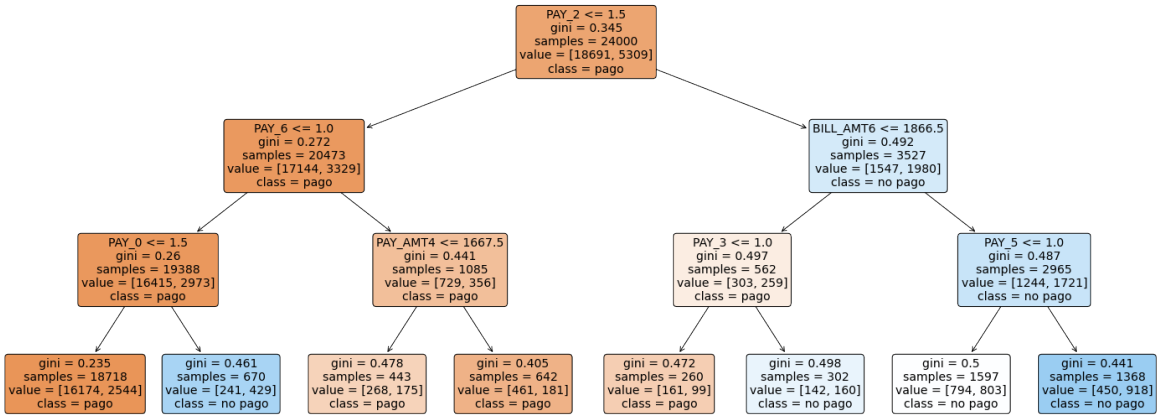


Figura 14: Ejemplo de DT construido con CART
Fuente: Elaboración propia.

2.7. Uso de técnicas de obtención de conocimiento

En la literatura existen diversos enfoques de cómo utilizar técnicas de obtención de conocimiento para el problema de seteo de parámetros. A continuación se detallan algunas de ellas:

En [Pavón *et al.*, 2009] se utiliza una Red Bayesiana (RB) para sugerir posibles configuraciones de parámetros. Una RB es un modelo gráfico que muestra las variables (nodos) de un conjunto de datos, su probabilidad condicional o independencia entre las mismas. Las conexiones (arcos) entre los nodos representan las relaciones causales entre los nodos, sin embargo no representan necesariamente una relación directa de causa y efecto. En la figura 15 se muestra una red simplificada para diagnosticar pacientes. La dirección de los arcos suele indicar la causalidad y los nodos son las facetas del paciente que se pueden relacionar a su condición. Por ejemplo, “Smoking” indica que el paciente fuma, además se ve que “Smoking” esta conectado a “LungCancer” y “Bronchitis”, de lo cual se infiere que el fumar aumenta las probabilidades de desarrollar bronquitis o cáncer al pulmón.

Los autores utilizan una colección de RBs para definir valores de parámetros para un algoritmo genético en el Root Identification Problem (RIP) dependiendo de las características de la instancia con la cual se trabajará. En la figura 16 se muestra el procedimiento para crear esta colección de RBs. Dado un problema P cada instancia del problema P_i tiene sus propias características k_{ji}. Cada una de estas instancias P_i se ejecuta el algoritmo A que se desea sintonizar y un determinado conjunto de parámetros X que utiliza el algoritmo. Una vez terminada la ejecución del algoritmo se tiene un conjunto de parámetros que evalúa su rendimiento X^+. Finalmente, para cada instancia P_i se genera una RB con la información obtenida de su rendimiento X^+ con cada una de las configuraciones utilizadas X. Esta colección se almacena como un par (Problema,Modelo), en donde el problema son las características

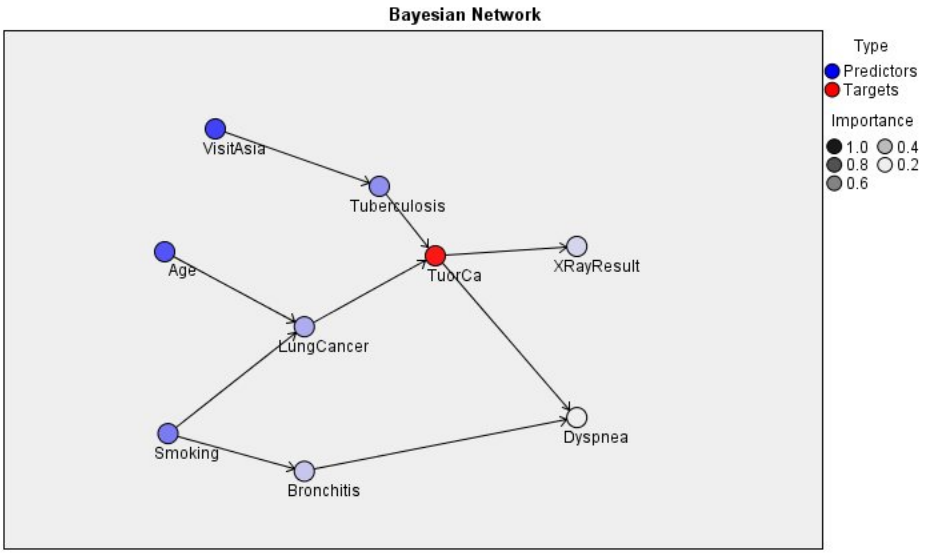


Figura 15: Ejemplo de red Bayesiana.
Fuente: Lauritzen S.(2022). Ejemplo de red Asia de Lauritzen y Spegelhalte. IBM, https://www.ibm.com.

de la instancia y el modelo la RB asociada a esa instancia.

Para instancias nuevas que no se encuentren en esta colección se escoge la que es más parecida. Esto se hace con la noción de distancia euclidiana entre el vector de las características de la instancia nueva y los vectores de las características en la colección. El modelo asociado a la instancia más cercana se utiliza para sugerir la configuración de parámetros.

En [Thierens, 2005] se utiliza Adaptive Pursuit Algorithm (APA) para determinar la probabilidades de aplicar operadores de algoritmos genéticos. Los algoritmos genéticos usualmente aplican sus operadores con una probabilidad fija. Sin embargo, no existe una regla general para determinar el valor óptimo de estas probabilidades. Además, el valor óptimo de probabilidad depende del estado actual de la búsqueda del algoritmo. Para determinar las probabilidades de los operadores, APA define un vector de operadores y un vector de recompensas asociadas a cada operador, el cual depende del estado actual del algoritmo objetivo. El objetivo de APA es determinar las probabilidades de aplicar cada operador de tal manera de maximizar las recompensas durante la ejecución del algoritmo. Finalmente, el vector de probabilidades resultante es utilizado para determinar la probabilidad de aplicar los operadores del algoritmo objetivo, dependiendo del estado que se encuentra.

En [Lessmann *et al.*, 2011] se utiliza un modelo de regresión para determinar el valor adecuado para los parámetros de un algoritmo de Particle swarm optimization (PSO). Se elaboran modelos de regresión independientes para predecir las tasas de aprendizaje (c_1, c_2) y el factor de la velocidad máxima. Estos modelos utilizan la velocidad, posición actual y la mejor posición de cada partícula como variables. Durante la ejecución del algoritmo se recopila la

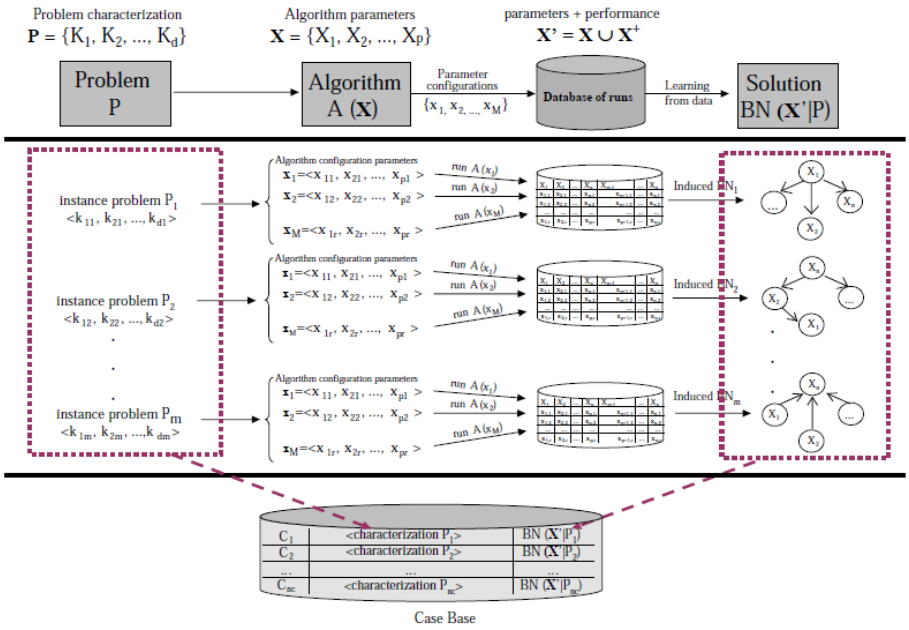


Figura 16: Creación de los modelos
Fuente: [Pavón *et al.*, 2009].

información de las partículas que han mejorado con respecto a la iteración anterior. Una vez que se recopilan suficientes observaciones para los modelos, estos se utilizan para determinar las tasas de aprendizaje y el factor de la velocidad máxima en las iteraciones restantes del algoritmo, actualizándose con la nueva información recopilada al final de cada iteración.

En [Zennaki y Ech-Cherif, 2010] se utiliza un modelo de Support Vector Machine (SVM) para predecir la calidad de las soluciones en una metaheurística de Tabu search. Para esto, en primer lugar se forman pares (solución, instancia) para ser ejecutados y obtener la calidad empírica de la solución. Luego, dependiendo de la calidad de las soluciones, estas se dividen en clases. Por ejemplo, a las soluciones que alcanzan el óptimo o están muy cercanas del óptimo se les etiqueta como “óptimas”. Por otro lado, a las soluciones muy lejas al óptimo se les clasifica como “mala”. Una vez que se obtienen suficientes datos se entra el modelo, donde este utiliza la información de la solución para predecir su clase. El modelo entrenado se utiliza para guiar a la metaheurística durante su ejecución. Durante la ejecución del algoritmo, al terminar una iteración de Tabu Search, se utiliza el modelo para evaluar la solución. Si la solución actual es clasificada como “buena”, se aplica un operador de intensificación. En cambio, si la solución actual es clasificada como “mala”, se aplica un operador de diversificación. Finalmente, si la solución actual se clasifica como “óptima”, se finaliza la ejecución del algoritmo.

En [Zhang *et al.*, 2007] se utiliza un algoritmo de clustering, K-means algorithm, para determinar de manera dinámica la probabilidad de cruce (p_x) y de mutación (p_m) de un Genetic Algorithm. Cada generación del algoritmo se divide en grupos (clusters) utilizando K-means.

Dependiendo del tamaño del cluster donde se encuentra el mejor y el peor individuo de la población, se modifican los valores de p_x y p_m. Por ejemplo, si el mejor y el peor individuo se encuentran en clusters pequeños, se incrementa p_x y p_m. Por otro lado, si el mejor y el peor individuo se encuentran en clusters grandes, se incrementa p_x y se disminuye p_m.

CAPÍTULO 3
PROPUESTA DE SOLUCIÓN

Dentro del marco del problema del seteo de parámetros se optó por trabajar en una mejora para un sintonizador de parámetros. El sintonizador que se seleccionó para la propuesta fue EVOCA. La elección de EVOCA por sobre otro sintonizador se debe a que se ve la existencia de una oportunidad de mejora en su diseño. El operador de mutación de EVOCA al explorar el vecindario del individuo cruzado, utiliza recursos para evaluar configuraciones que frecuentemente no son incluidas en la población. Es por esto, que nace la motivación de utilizar la información de las configuraciones descartadas para mejorar el uso de recursos dentro de EVOCA. Además, EVOCA fue diseñado para ser utilizado por usuarios que no necesariamente están interiorizados con el algoritmo objetivo o el problema de optimización con el que se esté trabajando, lo cual lo destaca para su elección en comparación con otros sintonizadores que tienen más barreras de entrada para su correcto uso. Para la mejora se propone incorporar un modelo de clasificación en EVOCA, con la finalidad de mejorar su desempeño, en términos de la calidad de las configuraciones obtenidas o del tiempo de ejecución.

A continuación se plantea el problema de clasificación que deben abordar las técnicas de obtención de conocimiento, junto con la explicación de cómo se utiliza una técnica de obtención de conocimiento en el sintonizador. Finalmente, se entregan los detalles de implementación para el modelo que se utilizará, además de cómo se evaluará este modelo.

3.1. Modelo de clasificación

Como se mencionó en el capítulo anterior, el operador de mutación de EVOCA, en primer lugar determina qué parámetro mutar, como se muestra en la figura 17. Luego de mutar el parámetro seleccionado, este nuevo individuo, individuo mutado, se evalúa y se compara con el individuo cruzado. En el caso de ser mejor que el individuo cruzado se añade a la población y se pasa a la siguiente iteración. En el caso contrario, se vuelve a mutar el parámetro seleccionado y se evalúa este nuevo individuo mutado. Esto se repite hasta intentar con un número de vecinos mayor al tamaño del dominio del parámetro mutado. Si ninguno de los individuos mutados es mejor que el individuo cruzado, entonces no se añade ninguno a la población y se pasa a la siguiente iteración.

Nuestra propuesta considera responder el siguiente problema de clasificación:”Dada la información recopilada sobre las configuraciones mutadas que han sido descartadas en iteraciones anteriores ¿Vale la pena evaluar y mutar la configuración candidata?”. De manera más formal, sea F(x) la función de clasificación que determina si x pertenece a la clase positiva (se acepta la configuración luego de la mutación) o la clase negativa (se descarta la configuración luego de la mutación). El modelo de clasificación busca aprender sobre F(x), en donde x es un individuo de la población de EVOCA antes de ser mutado. Luego de selec-

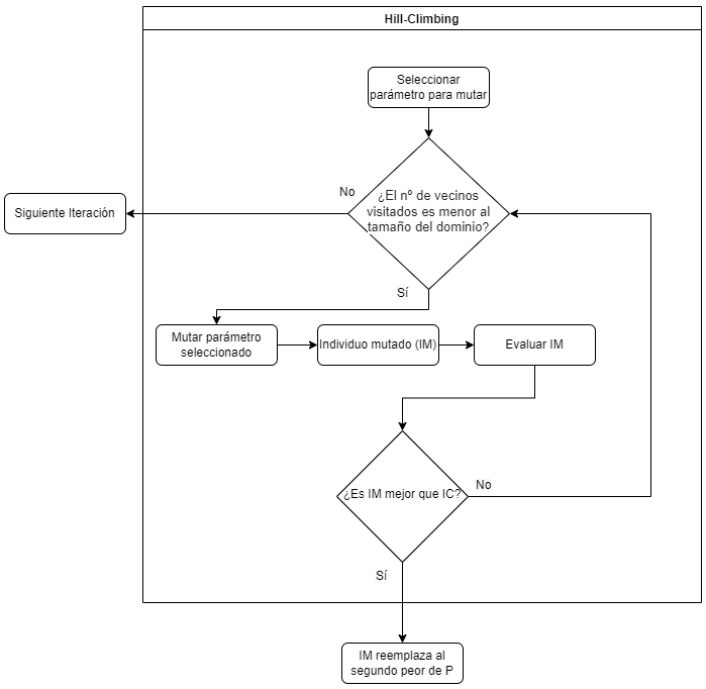


Figura 17: Operador de mutación de EVOCA
Fuente: Elaboración propia.

cionar un parámetro para mutar, se utiliza el modelo para determinar si, dado un individuo cruzado, una mutación del parámetro seleccionado, es prometedora o no, como se muestra en la figura 18. En el caso que el individuo es clasificado con la clase positiva se le aplica el operador de mutación, puesto según el modelo reúne las características deseables para que la mutación de este sea aceptada en la población. En el caso contrario, se intenta mutar otro parámetro de la configuración, sin repetir, hasta intentar con todas las alternativas posibles. En el caso que se determine que ningún parámetro es prometedor para ser mutado, se pasa a la siguiente iteración.

Se optó por incorporar un modelo en EVOCA, el cual aprende de la información que otorgan las configuraciones descartadas durante la ejecución de EVOCA, es decir, por un modelo online [4]. Esto se debe a que de esta manera el modelo se puede ajustar de forma dinámica al algoritmo que se está sintonizando y las instancias que se utilizan. Así se mantiene el espíritu inicial de diseño de EVOCA, que sea una herramienta accesible para experimentadores no tan interiorizados con el funcionamiento del algoritmo [Riff y Montero, 2013]. En caso contrario, se requiere la información una gran cantidad de ejecuciones previas de un algoritmo objetivo en un gran espectro de instancias, lo cual va más allá del alcance de esta memoria.

Finalmente, para evaluar el modelo se utilizarán las métricas más comunes como Accuracy, precision, Recall y F-measure(FM). Accuracy es utilizada, dado que es la más comúnmente

[4]Un modelo online, a diferencia de uno offline, se construye durante la ejecución del algoritmo y no antes de la ejecución como sería para el caso de un modelo offline

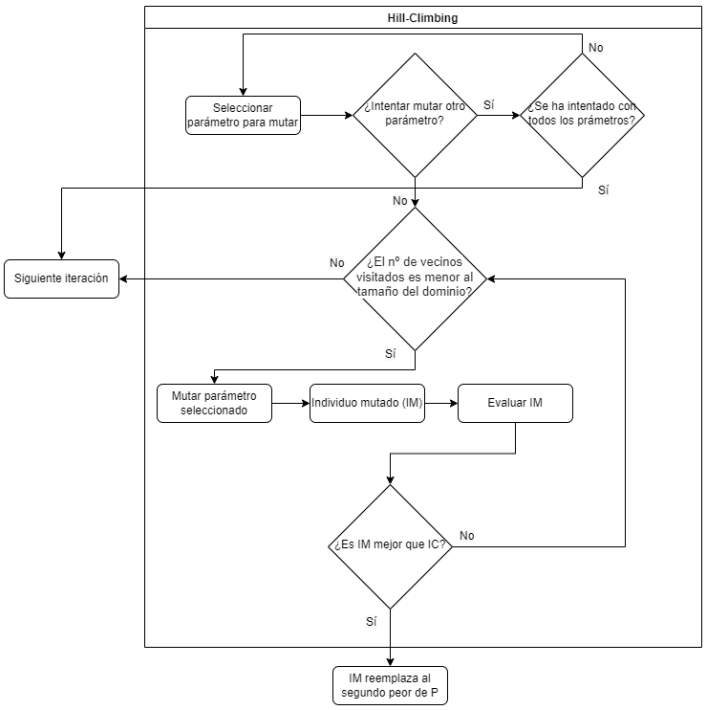


Figura 18: Operador de mutación propuesto
Fuente: Elaboración propia.

utilizada en problemas de clasificación [Hossin y Sulaiman, 2015]. Por otro lado, Precision y Recall son de interés en un problema de clasificación binario para tener una medida de la calidad de las predicciones para cada clase que no necesariamente será la misma, dado que esto está sujeto al balance que hay entre las clases [Prati *et al.*, 2015].

Un ejemplo de por qué es beneficioso utilizar métricas como Precision y Recall, se muestra a continuación. Si se tiene un grupo de 5 personas, como se muestra en la figura 19, en el cual solo una de ellas está enferma. Se le asigna la clase positiva (1) a la persona si está enferma y la clase negativa (0) si está sana. Si un modelo hace una predicción como se muestra en la figura, al evaluar solamente el Accuracy esta sería del 80 %, sin embargo este no refleja que la totalidad de personas enfermas fue clasificada de manera incorrecta. Por otro lado, al evaluar su Precision y Recall ambas son 0 %, dilucidando la falencia del modelo.

Finalmente, se utiliza FM la cual condensa en una sola métrica los resultados de precision y Recall, dado que esta es la media armónica entre ambas. Esto hace más fácil comparar el rendimiento entre modelos, puesto que los valores de precision y Recall están interrelacionados. Una alza de la precision puede venir a expensas de una baja de Recall y viceversa, lo cual complejiza el análisis del desempeño de los modelos utilizando solamente estas métricas.

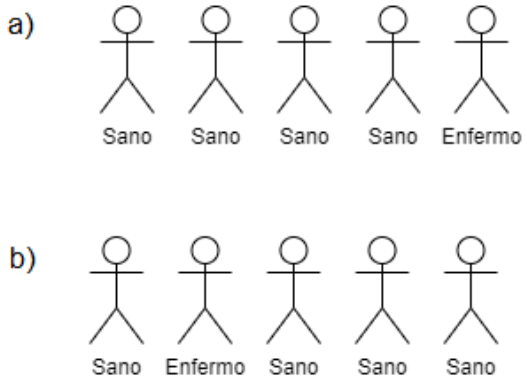


Figura 19: a) Estado real de las personas b) Predicción del modelo
Fuente: Elaboración propia.

3.2. Detalles de implementación

La propuesta de mejora se muestra en la figura 20, en donde el modelo es utilizado para determinar si debería aplicarse el operador de mutación sobre el parámetro seleccionado o si debería utilizarse sobre otro parámetro. En el caso de haber intentado con todos los parámetros disponibles y en ningún caso el modelo determina que debe ser mutado, entonces EVOCA continua su ejecución sin mutación.

En el algoritmo 3 se muestra la estructura del nuevo operador de mutación. En primer lugar se selecciona al azar un parámetro para mutar (línea 2). Luego, se utiliza el modelo para predecir la clase, recibiendo como parámetro el individuo cruzado (c_x) y el parámetro que se seleccionó para ser mutado (línea 5). En el caso que el modelo predice la clase positiva (1), la mutación es aceptada y se procede a utilizar la rutina de Hill-Climbing de EVOCA (línea 19). En el caso contrario, se escoge al azar otro parámetro para mutar y se vuelve a utilizar el modelo con este nuevo parámetro. La elección se realiza de tal forma que no se intente más de una vez con cada parámetro (línea 11-14). En caso que se haya utilizado el modelo con todos los parámetros posibles y en ningún caso se predice la clase positiva (linea 8), no se utiliza Hill-Climbing y se continúa a la siguiente iteración de EVOCA (línea 21).

Es necesario recolectar observaciones para entrenar el modelo. Para esto, se asigna una cantidad de evaluaciones donde se recolectan datos. Una vez alcanzada esta cantidad de evaluaciones, no se recolectan más datos y se empieza a utilizar el modelo. Para determinar la cantidad de evaluaciones se propone una regla con el objetivo de cumplir dos propósitos de diseño. (1) A medida que se sintonizan más parámetros, se deben ocupar más evaluaciones, puesto que el espacio de configuraciones posibles crece y aumenta la complejidad del problema. (2) La regla debe ser aplicable para un conjunto razonable de metaheurística. El número de parámetros de cada metaheurísticas es variable, por lo que se debe tener en consideración en el diseño. Dado lo anterior, se propone la siguiente regla, la cual es una adaptación de

Algorithm 3 Mutation EVOCA+NBC

```
usedParameters ← ∅
mut ← Random(Parameters)
usedParameters ← usedParameters ∪ mut
while Mutation is not accepted and Mutation is Possible do
    if NBC(c_x,mut) = 1 then
        Mutation accepted
    else
        if All parameters have been used then
            Mutation is not Possible
        else
            while mut in usedParameters do
                mut ← Random(Parameters)
            end while
            usedParameters ← usedParameters ∪ mut
        end if
    end if
end while
if Mutation accepted then
    Hill-Climbing(crossedCalibration,mut)
else
    Continue to next iteration
end if
```

la utilizada en I-Race para determinar el número de iteraciones[López-Ibáñez *et al.*, 2016]:

$$ratio = 0{,}1 * \log_2(n+1) \tag{1}$$

En donde n es el número de parámetros que se desea sintonizar. El resultado es la proporción del total de evaluaciones en las que EVOCA se ejecuta normalmente para recolectar datos para el modelo.

El coeficiente 0.1 se deriva luego de estudiar el impacto que tiene sobre el ratio y cómo afecta el número de evaluaciones que serán utilizadas para entrenar el modelo. Para esto, se analizó el caso de 5.000 iteraciones disponibles para un algoritmo objetivo de 5 y 10 parámetros. En la tabla 1 se muestra el caso de 5 parámetros. El valor en la columna evaluaciones es el producto de multiplicar el ratio por el número total de evaluaciones disponibles. En este caso se ve que no tiene sentido práctico utilizar un valor para el coeficiente igual o superior a 0.4, puesto que utilizan más evaluaciones de las disponibles. Por otro lado, el caso de 10 parámetros se muestra en la tabla 2. En este caso se ve que no tiene sentido práctico utilizar un valor para el coeficiente igual o superior a 0.3. En vista de lo anterior, el rango aceptable para el valor del coeficiente está entre 0.05 y 0.25. Cabe señalar que entre mayor es la cantidad de evaluaciones que se recolectan datos, menor es la cantidad de evaluaciones disponibles

para utilizar el modelo. Finalmente, se decide utilizar 0.1, porque con este valor se utilizan aproximadamente entre un cuarto y un tercio de las evaluaciones para entrenar el modelo, dependiendo del número de parámetros.

Tabla 1: Efecto del coeficiente para 5 parámetros. Fuente: Elaboración Propia.

Coeficiente	Ratio	Evaluaciones
0.05	0.129	647
0.10	0.258	1293
0.15	0.388	1939
0.20	0.517	2585
0.25	0.646	3232
0.30	0.775	3878
0.35	0.905	4524
0.40	1.034	5170

Tabla 2: Efecto del coeficiente para 10 parámetros. Fuente: Elaboración Propia.

Coeficiente	Ratio	Evaluaciones
0.05	0.173	865
0.10	0.346	1730
0.15	0.519	2595
0.20	0.692	3460
0.25	0.865	4325
0.30	1.038	5190
0.35	1.211	6055
0.40	1.384	6919

Los datos recolectados para el modelo son las configuraciones de los individuos cruzados antes de ser mutados, además del parámetro que se seleccionó para mutarlo. En el caso que el individuo mutado es incluido en la población se le asigna la clase positiva (1), en el caso contrario la clase negativa(0).

El modelo fue implementado utilizando Python. Para ello se utilizaron las librerías: pandas, sklearn, joblib y numpy. La implementación utilizada de EVOCA fue desarrollada en C++. Para utilizar los modelos se utilizaron las herramientas de desarrollador de Python (PythonDevTools), las cuales permiten utilizar rutinas de Python desde C++.

3.3. Modelo a utilizar

El modelo que se utilizará es el de Gaussian Naive Bayes Classifier (GNBC). GNBC es una variación de NBC, la cual trabaja bajo el supuesto que dada una clase, sus atributos siguen una distribución normal. Esta versión es comúnmente utilizada cuando se trabaja con variables continuas [Kamel *et al.*, 2019]. Por ejemplo, asumiendo que el i-ésimo parámetro es

continuo, para la clase c, su media y varianza están dadas por $\mu_{i,c}$ y $\sigma^2_{i,c}$ respectivamente. Entonces, la probabilidad de observar el valor x_i en el i-ésimo parámetro, dado la clase c, se calcula con la siguiente ecuación:

$$P(x_i|c) = \frac{1}{\sqrt{2\pi\sigma^2_{i,c}}} exp^{-\frac{(x_i-\mu_{i,c})^2}{2\sigma^2_{i,c}}}$$

GNBC no requiere hiperparámetros a diferencia de, por ejemplo, K-Nearest Neighbors(KNN) y Support Vector Machines (SVM). De esta manera, es más fácil ver el impacto del modelo construido con GNBC sobre EVOCA, puesto que se elimina la posibilidad que el modelo tenga una influencia negativa sobre EVOCA debido a una mala elección de hiperparámetros. Además, la elección de la métrica de distancia para KNN y el kernel de la SVM afectan el rendimiento del clasificador[Islam *et al.*, 2007] [Hussain *et al.*, 2011][Abu Alfeilat *et al.*, 2019]. En vista de que los espacios de búsqueda de cada instancia del problema pueden ser muy diferentes y la estocasticidad del algoritmo, es difícil determinar sin experimentación extensa un kernel o una métrica de distancia que tenga un buen rendimiento para el modelo.

Finalmente, para evaluar el modelo se utiliza k-fold con k=5 para obtener el promedio de las métricas mencionadas anteriormente. Se utiliza k=5, puesto que la evaluación del modelo se realiza en tiempo de ejecución y este valor k es favorable desde el punto de vista computacional y de esta manera se evita enlentecer la ejecución del sintonizador al ejecutar k-fold [Fushiki, 2011].

CAPÍTULO 4
VALIDACIÓN DE LA SOLUCIÓN

En este capítulo se presentan los resultados obtenidos de la experimentación realizada para obtener una medida cuantitativa del rendimiento de la propuesta, a la cual se le denominó EVOCA+NBC, con respecto a EVOCA por sí solo. En la sección 4.1, se describe el escenario de sintonización, el cual entrega la definición formal del problema de optimización que se utilizó, el algoritmo objetivo que se sintoniza , el detalle de las instancias utilizadas y el detalle de los hiperparametros tanto para EVOCA como del algoritmo objetivo. En la sección4.2, se muestran los resultados de las evaluaciones de las poblaciones finales en las instancias de prueba, en términos de su calidad y del tiempo de ejecución. En la sección 4.3, se analiza y compara la convergencia de las poblaciones finales de EVOCA+NBC y EVOCA por sí solo. Además, se comparan variaciones a la regla propuesta y el efecto en la convergencia de las poblaciones finales. En la sección 4.4, para una misma ejecución, se evalúan poblaciones obtenidas antes de terminar la ejecución de EVOCA+NBC y se comparan con las poblaciones finales de EVOCA. En la sección 4.5, se compara visualmente la calidad de las soluciones en cada ejecución mediante boxplots. En la sección 4.6, se analiza la distribución de cada parámetro del modelo NBC y la importancia de cada uno. Finalmente, en la sección 4.7 se presenta la discusión de los resultados, en donde se presentan posibles mejoras a la propuesta en función de los resultados obtenidos. Además, se analizan resultados no favorables de la propuesta y las posibles razones detrás de ellos.

4.1. Escenario de sintonización

4.1.1. Multidimensional Knapsack Problem

Multidimensional Knapsack Problem (MKP) es un problema que ha sido sujeto de estudio de forma recurrente, dado que está relacionado con el estudio de muchos problemas prácticos tales como asignación de recursos en sistemas distribuidos, asignación de presupuesto y asignación de capital en la bolsa, entre otros [Osorio y Cuaya, 2005]. Su definición es la siguiente: Dado un conjunto de objetos, cada uno con sus respectivos valores y costos (pesos), se selecciona un subconjunto de ellos y se les guarda en una mochila, sin sobrepasar su capacidad, de tal forma que al evaluar todos los objetos dentro, se obtenga el mayor valor. Puesto que la mochila no tiene la capacidad de llevar todos los objetos, es necesario utilizar alguna estrategia para decidir qué objetos dejar dentro. Más formalmente en el MKP se busca seleccionar los objetos que maximizan la siguiente ecuación:

$$max \sum_{j=1}^{n} p_j x_j$$

donde x_j tiene valor uno en caso que se decida llevar el j-ésimo objeto y cero en caso contrario. Por otro lado, p_j es el valor o la ganancia que se obtiene al llevar el j-ésimo objeto. Además, el problema está sujeto a las siguiente restricciones:

$$\sum_{j=1}^{n} r_{ij} x_i \leq b_i$$

En donde b_i denota la capacidad de cada dimensión i y r_{ij} el espacio que ocupa el objeto j en la dimensión i.

4.1.2. Ant Knapsack

El algoritmo objetivo Ant Knapsack (AK) es la adaptación de un algoritmo de Ant Colony Optimization (ACO) para resolver el MKP. La idea básica de ACO es modelar el problema de tal forma que su solución sea encontrar el camino de menor coste en un grafo [Alaya *et al.*, 2004]. Para esto se utilizan hormigas artificiales, las cuales imitan el comportamiento de hormigas reales. Estas depositan feromonas en las componentes del grafo y escogen su camino con respecto a probabilidades que dependen de la concentración de feromona en los caminos que ha sido depositada por otras hormigas anteriormente. Cabe mencionar que la concentración de feromona en los caminos se evapora con el tiempo.

En el contexto del MKP los objetos se ven como los nodos de un grafo completamente conexo. En cada iteración, cada hormiga visita los nodos y así construye una posible solución. Comenzando desde un objeto aleatorio, el próximo nodo que visitan las hormigas depende de la probabilidad de selección de cada objeto. Esta probabilidad depende proporcionalmente del factor de feromonas (τ) y del factor heurístico (η) de cada objeto. La importancia relativa de cada uno de estos factores es regulada por α y β para τ y η respectivamente. El factor de feromonas de cada objeto, depende de la concentración de feromonas de los arcos que conectan los objetos que ya son parte de la solución parcial con los posibles objetos candidatos. El factor heurístico de cada objeto depende de la relación entre su ganancia y de un factor de pseudo utilidad [Alaya *et al.*, 2004]. Una vez que cada hormiga construye una solución, se actualiza la concentración de feromonas en los arcos. Para esto, en primer lugar se disminuye la concentración de feromona en todos los arcos a través de la evaporación. La cantidad evaporada es regulada por el parámetro ρ. Luego, la hormiga que construye la mejor solución, deposita feromona en los arcos que conectan los objetos de la solución que construyó. El valor de la concentración de la feromona está acotada superiormente por τ_{max} e inferiomente por τ_{min}, puesto que se utilizó la versión Max-Min Ant System [Stutzle y Hoos, 1997].

4.1.3. Instancias del problema

Se utilizó un conjunto de instancias que ya han sido utilizadas previamente en la literatura como benchmark [Drake, 2015]. Estas instancias cuentan con un número variado de objetos (n) y dimensiones (m) como se muestra en la tabla 3. Las instancias pertenecientes a OR-Library utilizan la siguiente nomenclatura ORmxn - tightness ratio. El tightness ratio es la razón entre la suma de los coeficientes r_{ij} de cada variable para una restricción y el coeficiente b_i de la restricción. Se utilizaron 17 instancias para sintonizar el algoritmo objetivo (instancias de entrenamiento) y 77 para evaluar las configuraciones entregadas por este (instancias de prueba).

Tabla 3: Detalle de las instancias utilizadas. Parte I Fuente: Elaboración Propia.

Instancia	n	m
gk01	100	15
gk02	100	25
gk03	150	25
gk05	200	25

4.1.4. Hiperparametros

Para la sintonización de AK, los dominios utilizados para cada variable se muestran en la tabla 4. Además, para cada ejecución de AK se utilizaron 1.000 iteraciones y un mínimo de feromonas (τ_{min}) de 0,01.

Para la ejecución de EVOCA y la propuesta EVOCA+NBC se utilizaron 10.000 evaluaciones, una población de tamaño máximo de 10 y 3 pares de (semilla,instancia) para evaluar las configuraciones. Para EVOCA+NBC la regla propuesta en la ecuación 1, al evaluarla con 5 parámetros y considerando el total de 10.000 evaluaciones, esta indica que se utilizarán 2585 para recopilar datos y entrenar el modelo NBC.

Tabla 4: Dominios de las variables de AK. Fuente: Elaboración Propia.

Parámetro	Min	Max
TotalAnts	2	50
α	0,1	20
β	0,1	20
τ_{max}	0,1	30
ρ	0,001	1

4.2. Resultados

Para evaluar la propuesta se ejecutaron EVOCA y EVOCA+NBC en 5 semillas diferentes. Las poblaciones finales constan de 10 configuraciones y con la intención de observar la convergencia de EVOCA, se evaluaron las 8 mejores de cada una. La razón principal es que el operador de cruce y el operador de mutación, con el fin de diversificar, pueden incluir individuos en la población que no necesariamente son mejores que el resto de la población. Es por esto que se observa que en las poblaciones finales están presente 2 configuraciones que tienden a ser peor que el resto de la población y se excluyen del análisis. Para la evaluación, cada configuración fue ejecutada por el algoritmo objetivo 10 veces con semillas distintas en cada una de las 77 instancias de prueba, por lo que finalmente se obtuvieron 30.800 (8*5*77*10) datos del tiempo de ejecución y la calidad de solución para las configuraciones de EVOCA y EVOCA+NBC. En las tablas 5 y 6 se muestra un **extracto** de los resultados calidad y tiempo respectivamente, el resto de ellos se adjuntan en el anexo en las tablas 9,10,11,12. En las tablas, se muestra el promedio (AVG) de todas evaluaciones, su desviación estándar (SD) y el mejor resultado encontrado (Best) para cada instancia. En las tablas de calidad se muestra el GAP, esto es, qué tan lejano está el resultado obtenido con respecto al óptimo conocido, donde un valor más cercano a cero es mejor.

Tabla 5: Resultados de calidad de las soluciones parte I. Fuente: Elaboración Propia.

	EVOCA+NBC			EVOCA		
Instancia	Best	AVG	SD	Best	AVG	SD
gk02	0.682	1.561	0.509	**0.606**	**1.513**	0.497
gk03	**0.831**	1.612	0.443	0.884	**1.611**	0.448
gk05	**0.781**	1.688	0.441	0.873	**1.646**	0.453
OR10x500-0.25_1	0.491	1.384	0.876	**0.393**	**1.317**	0.685
OR10x500-0.25_10	0.461	1.305	0.912	0.461	**1.275**	0.797
OR10x500-0.25_2	0.416	1.308	0.802	0.416	**1.288**	0.802
OR10x500-0.25_3	**0.408**	1.213	0.720	0.429	**1.170**	0.683
OR10x500-0.25_4	0.382	1.329	0.913	**0.279**	**1.280**	0.815
OR10x500-0.25_6	**0.341**	1.123	0.829	0.398	**1.043**	0.556

Al comparar los resultados, con respecto a la calidad de las mejores soluciones obtenidas, se ve que la propuesta obtiene una mejor solución en el 63 % de las instancias y un mejor rendimiento promedio en aproximadamente el 49 % de las instancias. Para visualizar esto, se gráfica la diferencia entre las mejores soluciones para cada instancia en la figura 21. El valor de la diferencia se calcula como EVOCA+NBC-EVOCA, esto es, un valor negativo indica que EVOCA+NBC tuvo un mejor resultado. Al analizar las diferencias negativas se ve que en promedio es de -0.035 y el menor valor encontrado es de -0.127. En el caso de las diferencias positivas, se ve un valor promedio de 0.030 y un mayor valor de 0.103. Dado lo anterior, se ve que no hay diferencias significativas de las magnitudes en términos absolutos.

Al repetir el análisis anterior, con respecto al tiempo de ejecución de las soluciones obteni-

das, la propuesta obtiene un mejor resultado en el 56 % de las instancias y un mejor tiempo promedio en el 56 %. En la figura 22 se muestra la diferencia de los tiempos de ejecución para cada instancia. Al analizar las diferencias negativas se ve que en promedio es de -0.590 y el menor valor encontrado es de -4.220. En el caso de las diferencias positivas, se ve un valor promedio de 1.388 y un mayor valor de 4.570. Se ve que en promedio, en los casos donde la propuesta tiene un peor desempeño, la diferencia es de aproximadamente del doble de la diferencia promedio de los casos donde obtiene un mejor desempeño. En términos de los mejores resultados a favor y en contra, no se ve una diferencia significativa en términos absolutos.

Se realizó un test de Wilcoxon con los resultados de la calidad de EVOCA y EVOCA+NBC, utilizando los 30.800 resultados obtenidos para ambos. Se obtuvo un p-valor de $p = 5{,}43 * 10^{-8}$. Para un nivel de significancia del 5 % se rechaza la hipótesis nula, la cual plantea que muestras emparejadas provienen de la misma distribución, y se puede asumir que hay suficiente evidencia estadística para afirmar que las muestran son distintas.

Con respecto a las métricas utilizadas para evaluar el modelo, un resumen de estas se encuentra en la tabla 7. En primer lugar se ve un Accuracy promedio de 0.66. Además, se ve que en promedio se obtiene una Presicion de 0.66, esto es, del total de configuraciones que clasificó como positiva, en promedio clasifica correctamente a dos tercios. Por otro lado, se ve un Recall promedio de 0.23, es decir, del total de configuraciones de clase positiva en los datos de evaluación clasifica correctamente un poco menos de un cuarto correctamente. Finalmente, esto se resume en un F-score promedio de 0.34. Esta situación muestra que el modelo no tiene el mismo rendimiento para clasificar ambas clases. Esto puede deberse al desbalance de las clases en los datos de entrenamiento del modelo. La cantidad de ejemplos por clase en cada ejecución se muestra en la tabla 8. Esto muestra que hay mayor proporción de datos de la clase negativa que de la clase positiva. Adicionalmente, se puede relacionar el bajo valor de F-Score (0.21) del modelo en la ejecución 5 con la baja cantidad de datos de clase positiva (56), la cual fue la menor de todas las ejecuciones.

Tabla 6: Resultados de tiempo de ejecución parte I. Fuente: Elaboración Propia.

	EVOCA+NBC			EVOCA		
Instancia	Best	AVG	SD	Best	AVG	SD
gk02	**1.950**	**2.058**	0.032	2.000	2.064	0.030
gk03	**5.130**	5.442	0.079	5.260	**5.441**	0.079
gk05	**10.730**	**11.103**	0.145	10.810	11.153	0.137
OR10x500-0.25_1	**67.700**	**70.663**	0.868	68.180	70.736	0.811
OR10x500-0.25_10	**68.460**	71.634	1.063	68.640	**71.615**	0.965
OR10x500-0.25_2	**68.020**	**70.963**	0.806	68.600	71.125	0.788
OR10x500-0.25_3	**68.010**	**70.731**	0.873	68.360	70.753	0.825
OR10x500-0.25_4	**68.540**	**71.552**	1.006	69.100	71.567	0.934
OR10x500-0.25_6	**68.640**	**72.038**	0.871	69.630	72.073	0.791

Tabla 7: Métricas de evaluación del modelo para cada ejecución. Fuente: Elaboración Propia.

Ejecución	Accuracy	Presicion	Recall	F-Score
1	0.66	0.64	0.25	0.36
2	0.64	0.76	0.28	0.41
3	0.56	0.86	0.23	0.37
4	0.64	0.79	0.24	0.37
5	0.79	0.39	0.16	0.21

Tabla 8: Frecuencia de las clases en los datos de entrenamiento del modelo. Fuente: Elaboración Propia.

Ejecución	Clase negativa	Clase positiva
1	599	105
2	596	113
3	605	106
4	623	97
5	686	56

4.3. Análisis de convergencia

En esta sección en primer lugar se comparan el promedio de la calidad de las 8 mejores configuraciones con respecto a cada iteración del algoritmo. Luego se repite la comparación, pero variando el porcentaje de evaluaciones que se utilizan para entrenar el modelo, sin utilizar la regla (ratio) de la ecuación 1.

4.3.1. Poblaciones

En la figura 23 y 24 se muestran dos ejecuciones diferentes de EVOCA y EVOCA+NBC, en donde EVOCA+NBC logra un mejor promedio final de las configuraciones. La línea punteada indica la iteración donde se empieza a utilizar el modelo entrenado, dado que se utilizó la misma semilla y hardware, se ve que antes de empezar a utilizar el modelo tanto EVOCA como EVOCA+NBC son equivalentes. Dado el comportamiento del algoritmo para ambos se ven periodos donde el promedio no mejora, los cuales son seguidos por un salto al incluir un mejor individuo a la población. En ambos casos se ve que EVOCA+NBC llega antes de terminar su ejecución a poblaciones con rendimiento promedio similar a la de la población final de EVOCA.

4.3.2. Regla propuesta

Para analizar el efecto de diferentes cantidades de datos de entrenamiento para el modelo, se modificó el porcentaje de evaluaciones que se utiliza para el entrenamiento del modelo. Se mantienen los hiperparámetros utilizados anteriormente, solamente modificando el número de evaluaciones de 10.000 a 5.000. Se utiliza para la comparación el 5 %, 10 %, 15 %, 30 %, 40 % de las evaluaciones. Además, se utiliza para la comparación la regla propuesta en la ecuación 1, siendo del 25.8 % . Para comparar el efecto de esta variación se ve el promedio de las 8 mejores configuraciones con respecto a la iteración del algoritmo.

En el caso de 5 % (figura 25) y 10 % (figura 26) se ve que EVOCA por sí solo converge a un mejor resultado EVOCA+NBC. Por otro lado, para un 15 % (figura 27) se ve que EVOCA+NBC logra converger a un mejor resultado, sin embargo este lo hace al utilizar una mayor cantidad de iteraciones, puesto que, en este caso en particular, el modelo no acepta ninguno de los valores de parámetros de manera sucesiva en las mutaciones, provocando que no se utilice el operador de mutación.

Para el caso de la regla (figura 28), 30 % (figura 29) y 40 % (figura 30) se ve que EVOCA+NBC convergen a un mejor resultado.

Estos resultados son válidos para una semilla en particular, pero muestran la necesidad de una cantidad mínima de datos para que el modelo impacte positivamente en EVOCA. Por otro lado, muestra que hay espacio de mejora para la regla propuesta, dado que se ve que al aumentar el porcentaje de datos no tiene un efecto negativo. Además, como trabajo futuro se debe evaluar con otros algoritmos a sintonizar.

4.4. Evaluación de poblaciones intermedias

En las ejecuciones 2 y 5 se ve que EVOCA+NBC logra un mejor resultado final. Dado esto, se ve la oportunidad de una ganancia en términos de tiempo. Si durante la ejecución de EVOCA+NBC logra obtener una población que, en términos de rendimiento, sea similar a la población final de EVOCA, significaría que la propuesta puede alcanzar el resultado final de EVOCA en menos evaluaciones.

Para determinar la población intermedia a evaluar, se calcula la diferencia entre la calidad promedio de las 8 mejores configuraciones en cada iteración de EVOCA+NBC y la calidad promedio de las 8 mejores configuraciones de población final de EVOCA. Se utiliza la población donde esta diferencia es mínima.

Una vez obtenidas las poblaciones intermedias, se evalúan en las instancias de prueba y se comparan con las poblaciones finales de EVOCA. Al comparar la calidad, en las instancias donde se obtiene un resultado mejor o peor, se ve que en un 43 % de los casos se obtiene una solución de mejor calidad y en el 71 % de los casos un mejor promedio. De la misma

forma, para el tiempo de ejecución se ve en un 100 % de las instancias un menor tiempo de ejecución y en el 100 % un menor tiempo de ejecución promedio.

Las poblaciones intermedias de EVOCA+NBC son de menor calidad que las poblaciones finales de EVOCA, sin embargo tienen un mejor tiempo de ejecución y estas fueron alcanzadas con 7326 y 4821 evaluaciones para la ejecución 2 y 5 respectivamente . Finalmente, se realiza un test de Wilcoxon con los datos de la calidad de las poblaciones intermedias de EVOCA+NBC y las poblaciones finales de EVOCA. Se obtine un p-valor de $p = 3{,}39 * 10^{-54}$. Para un nivel de significancia del 5 % se rechaza la hipótesis nula y se puede asumir que hay suficiente evidencia estadística para afirmar que las muestran son distintas.

4.5. Boxplots

Para evaluar visualmente el rendimiento en cada ejecución de EVOCA+NBC en comparación con EVOCA, se elaboran boxplots. En la figura 31 se muestran los boxplots elaborados con los datos de la calidad de las configuraciones finales para cada ejecución. En el eje x se muestran las distintas semillas utilizadas para cada ejecución, mientras que en el eje y la calidad en términos de GAP. Se ve que en general el promedio y la dispersión de los resultados de EVOCA+NBC son equiparables con EVOCA, por lo que se ve que no hay una pérdida de desempeño por parte de la propuesta.

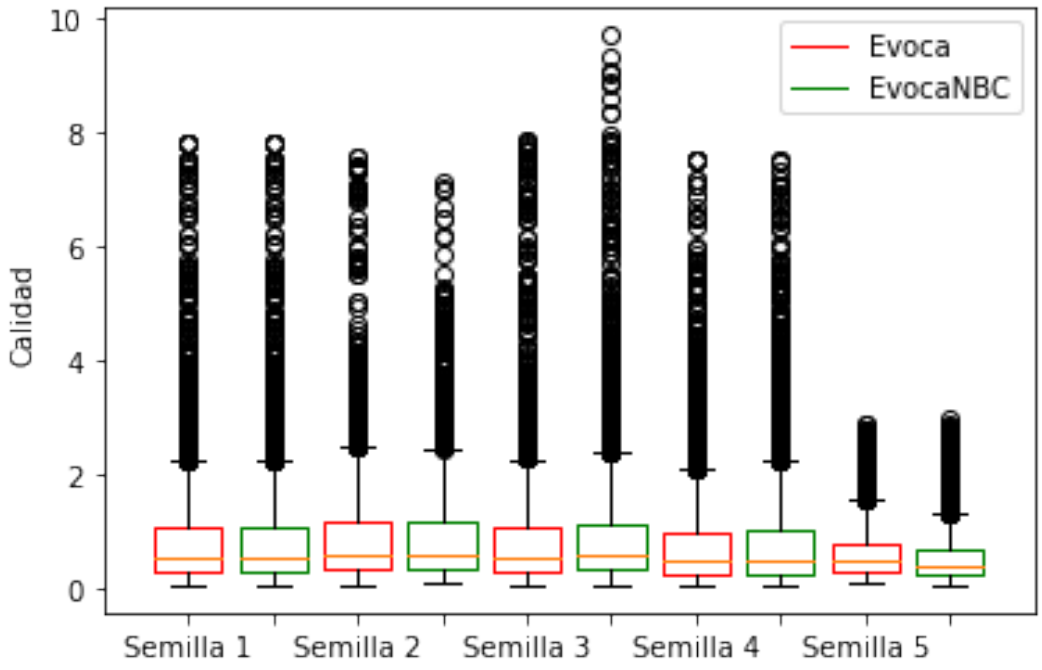


Figura 31: Boxplots por cada ejecución
Fuente: Elaboración propia.

4.6. Análisis de importancia de las características

A continuación se presentan las funciones de densidad de probabilidad de cada clase para cada una de las variables del modelo dentro del rango de sus dominios. En cada gráfico, la linea color azul representa la densidad de probabilidad de la clase negativa, mientras que la linea color naranjo representa la densidad de probabilidad de la clase positiva. Se ve que tanto para el total de hormigas (32), τ_{max} (figura 35), ρ (figura 36) y el parámetro que se muta (figura 37), las distribuciones de ambas clases no son significativamente diferentes. Por otro lado, en el caso de α (figura 33) y β (figura 34) se ve que las distribuciones son significativamente diferentes. Dado lo anterior, se puede concluir que para el modelo NBC las variables más relevantes son α y β. Por ejemplo, si el modelo considerara solamente α para predecir la clase. Las configuraciones con valores de α cercanos a aproximadamente 3.0, serian aceptadas (1) y las con valores cercanos a aproximadamente 6.0 serian rechazadas (0). De manera similar, si el modelo considerara solamente β. Las configuraciones con valores de β cercanos a aproximadamente 6.0, serian aceptadas (1) y las con valores cercanos a aproximadamente 8.0 serian rechazadas (0).

4.7. Discusión

En las secciones anteriores se ve que NBC tiene un impacto positivo en EVOCA. Sin embargo en algunas de las ejecuciones se ve el efecto contrario. En las figuras 38 y 39 se muestra esta situación. Se hizo un análisis de las evaluaciones de las poblaciones de estas ejecuciones de forma separada y conjunta. Al analizar las ejecuciones por separado, considerando solamente las instancias donde hay diferencias, se ve que en la ejecución 3 (figura 38) en términos de calidad se ve que EVOCA+NBC obtiene un mejor rendimiento en el 17 % de las instancias y un mejor rendimiento promedio 9 % de las instancias. Por otro lado, en la ejecución 4 (figura 39) se ve que EVOCA+NBC obtiene un mejor rendimiento en el 93 % de las instancias y un mejor rendimiento promedio en el 31 % de las instancias. Finalmente, se analizan los resultados de forma conjunta, en donde EVOCA+NBC obtiene un mejor rendimiento en el 77 % de las instancias y un mejor rendimiento promedio en el 9 % de las instancias. Estos resultados muestran que la adición del modelo puede empeorar el rendimiento de EVOCA. Además, muestra que la pérdida en el rendimiento promedio no necesariamente está acompañada con una pérdida en la capacidad de encontrar mejores soluciones.

Adicionalmente, se realiza un test de Wilcoxon con los datos de la calidad de las ejecuciones 3 y 4. Se obtiene un p-valor de $p = 8{,}69 * 10^{-25}$. Para un nivel de significancia del 5 % se rechaza la hipótesis nula y se puede asumir que hay suficiente evidencia estadística para afirmar que las muestras son distintas.

En la ejecución 1 (figura 40) de los experimentos se ve una situación completamente distinta a lo mencionado anteriormente. En este caso EVOCA+NBC no tiene ningún impacto. Esto se debe a las clasificaciones del modelo. En esta ejecución el modelo predice con la clase

positiva 418 veces, mientras que con la clase negativa 9 veces. Esto provoca que en gran parte de la ejecución se comporte de la misma manera que EVOCA, puesto que no cambia el parámetro utilizado en la mutación.

La evaluación del modelo muestra que tiene una mejor capacidad de identificar la clase positiva que la negativa, puesto que produce una baja cantidad de falsos positivos, pero una gran cantidad de falsos negativos. Para mejorar el rendimiento del modelo se puede explorar la posibilidad de añadir más variables al modelo. Por ejemplo, la variación del parámetro que se va a mutar o bien la calidad del individuo cruzado. Desde otra perspectiva, existe la posibilidad de eliminar variables, dado que solamente en dos de las variables hay una diferencia significativa en su distribución. Finalmente, otro aspecto a considerar es el de volver a entrenar el modelo durante la ejecución. Se puede continuar recolectando datos durante toda la ejecución del algoritmo y utilizarlos para actualizar el modelo.

Al analizar las variaciones del porcentaje de evaluaciones utilizadas para entrenar el modelo, se ve que existe la posibilidad de aumentar el porcentaje y que esto no tenga un efecto negativo. Sin embargo, al utilizar más evaluaciones en entrenar el modelo se retarda el uso de este, lo cual a su vez puede dar menos espacio para divergir de la población de EVOCA. Como se ve en el caso de la ejecución 1, al no variar lo suficiente el parámetro de mutación se puede dar el caso que la población final sea la misma que la de EVOCA por sí solo. Además, entre más se retarda el uso del modelo, menos espacio hay para una ganancia de tiempo en el uso del sintonizador. En las ejecuciones 2 y 5 se ve que se puede alcanzar una población que tiene un rendimiento similar en menos evaluaciones. Si el modelo se empieza a utilizar, por ejemplo, en el momento en que se alcanzan estas poblaciones la ganancia en evaluaciones, en el caso que la haya, será menor a la que se vio.

En el diseño de la propuesta se optó por no utilizar el operador de mutación si no se puede intentar mutar más parámetros, sin embargo no es la única opción. Una posible alternativa es intentar mutar otra configuración de la población actual. Otro enfoque puede cambiar de manera aleatoria el valor de uno de los parámetros y volver a intentar aplicar el operador de mutación sobre esta nueva configuración.

Finalmente, en el contexto de algoritmos evolutivos, la adición del modelo puede interpretarse como una probabilidad de ejecutar el operador mutación sobre un parámetro en cada iteración, la cual se determina de manera dinámica durante la ejecución del algoritmo.

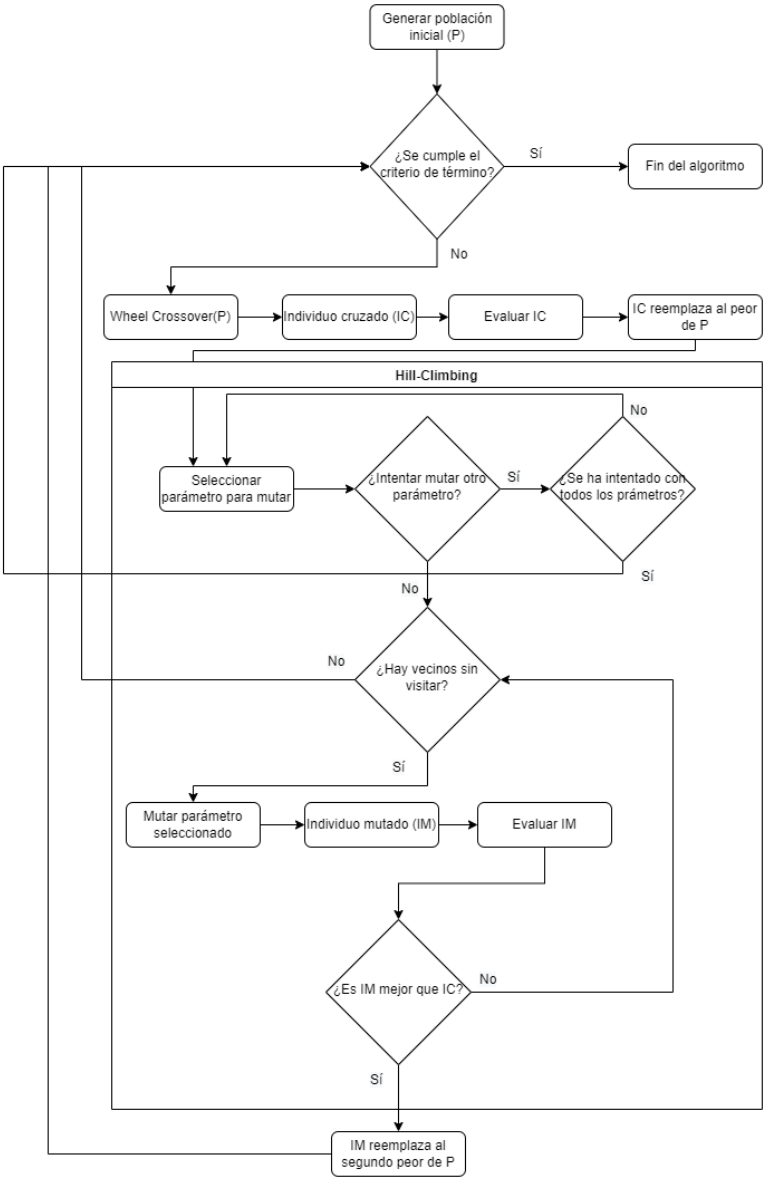


Figura 20: Diagrama de flujo de EVOCA con la propuesta
Fuente: Elaboración propia.

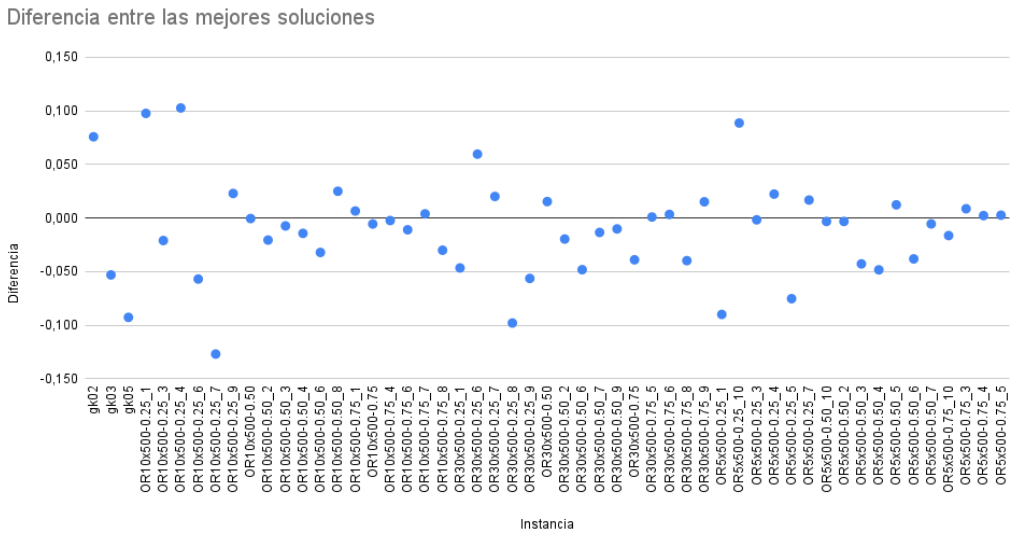


Figura 21: Diferencia entre la calidad de las mejores soluciones
Fuente: Elaboración propia.

Figura 22: Diferencia entre el tiempo de ejecución de las mejores soluciones
Fuente: Elaboración propia.

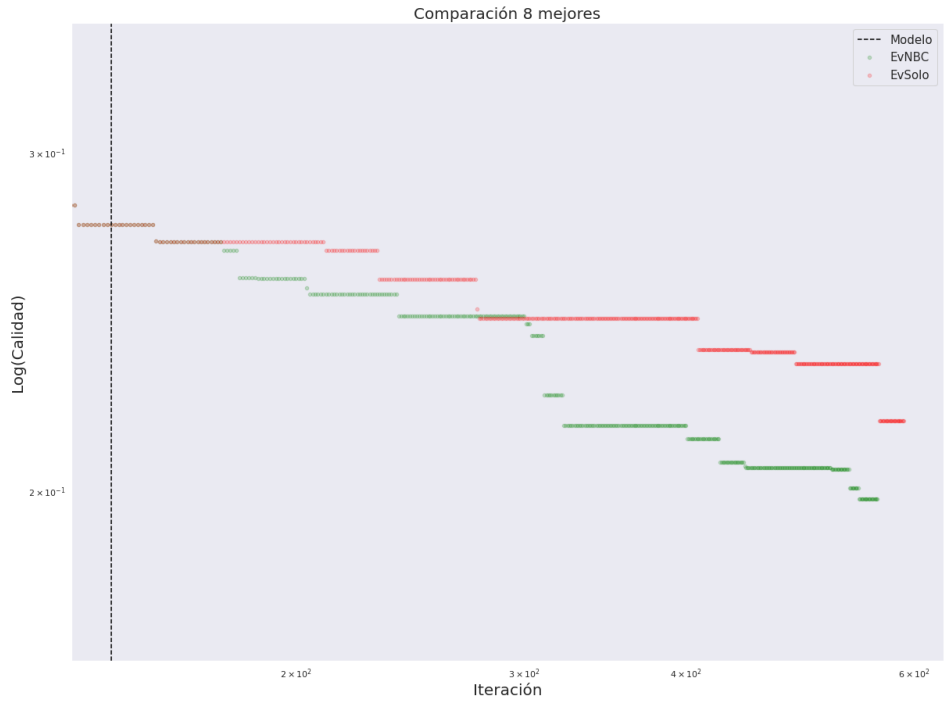


Figura 23: Promedio de configuraciones v/s iteración - Ejecución 2
Fuente: Elaboración propia.

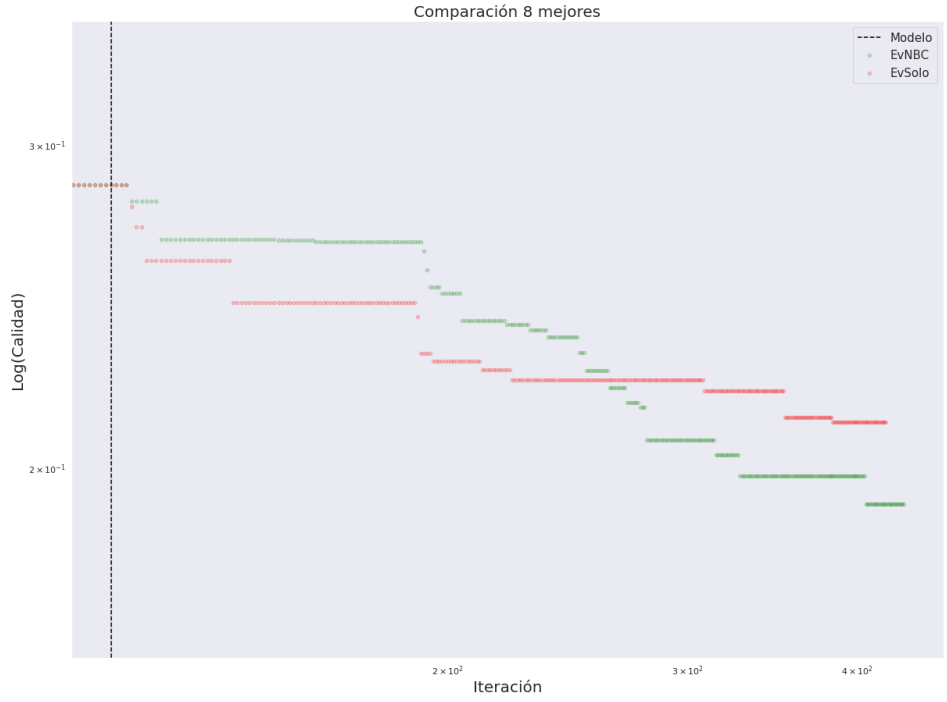


Figura 24: Promedio de configuraciones v/s iteración - Ejecución 5
Fuente: Elaboración propia.

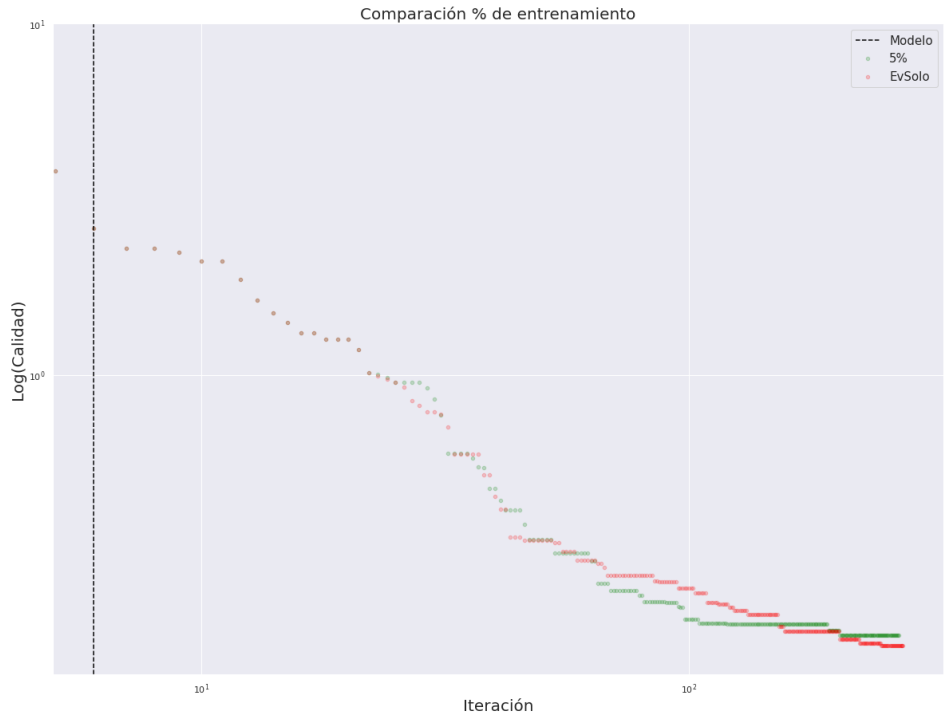


Figura 25: Promedio de las 8 mejores configuraciones v/s iteraciones - 5 % de evaluaciones
Fuente: Elaboración propia.

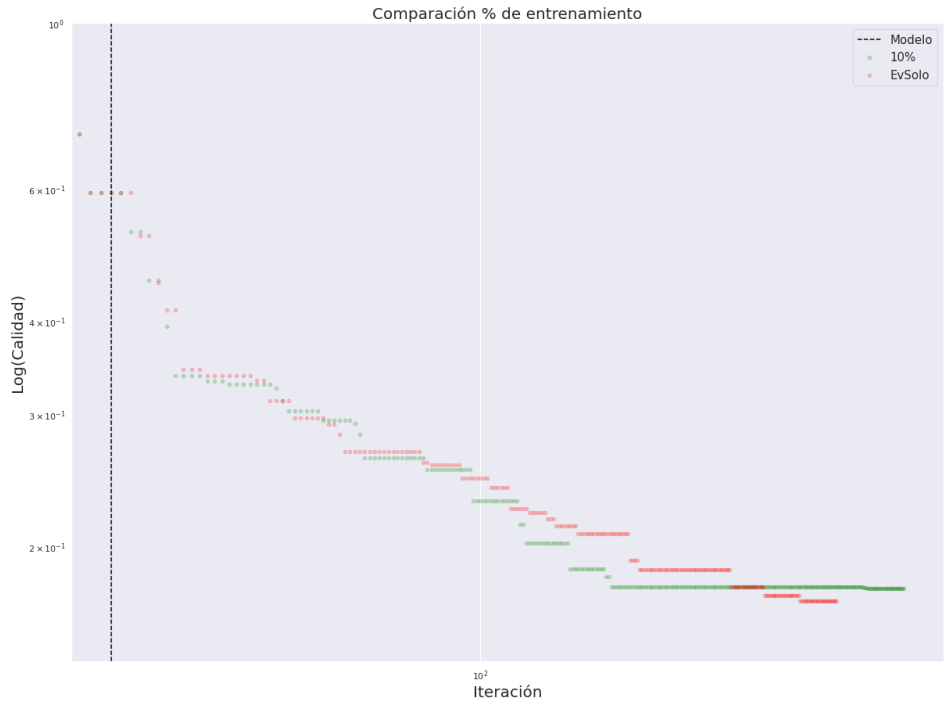


Figura 26: Promedio de las 8 mejores configuraciones v/s iteraciones - 10 % de evaluaciones
Fuente: Elaboración propia.

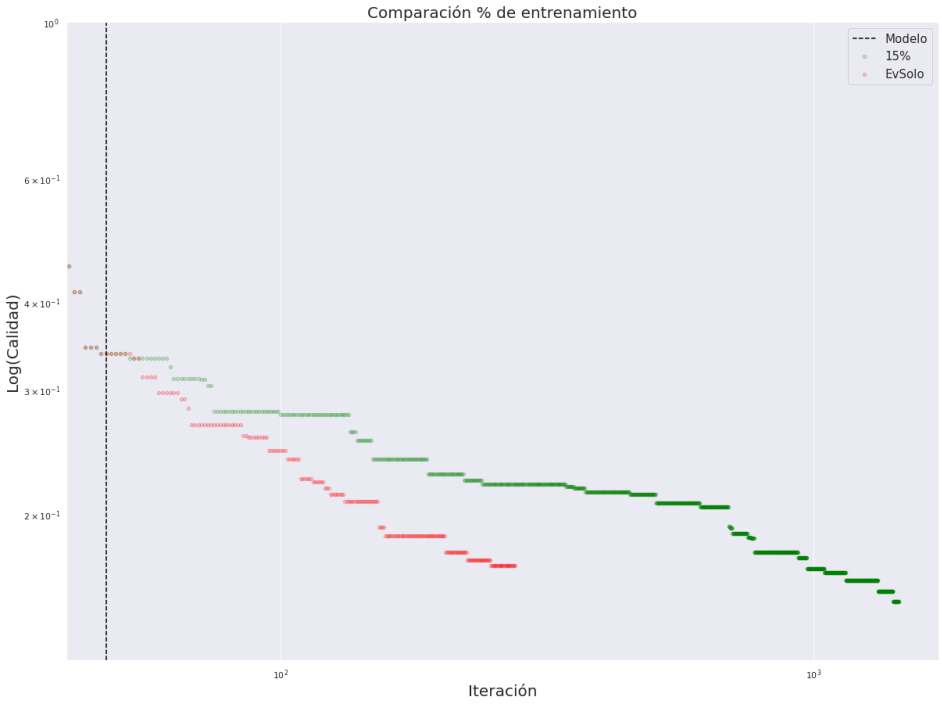


Figura 27: Promedio de las 8 mejores configuraciones v/s iteraciones - 15 % de evaluaciones
Fuente: Elaboración propia.

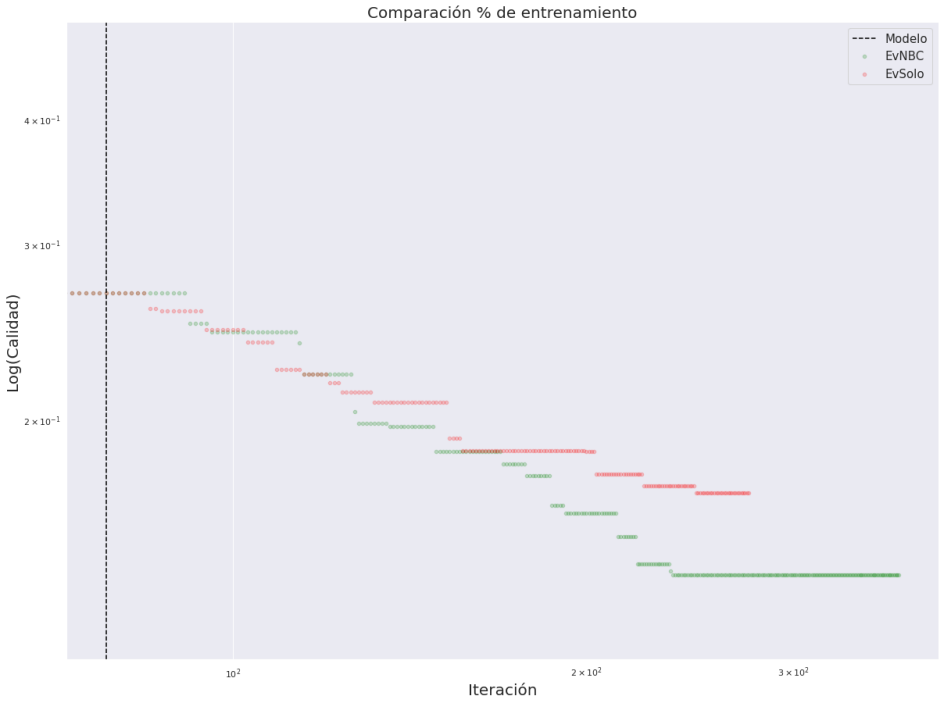


Figura 28: Promedio de las 8 mejores configuraciones v/s iteraciones - Regla propuesta
Fuente: Elaboración propia.

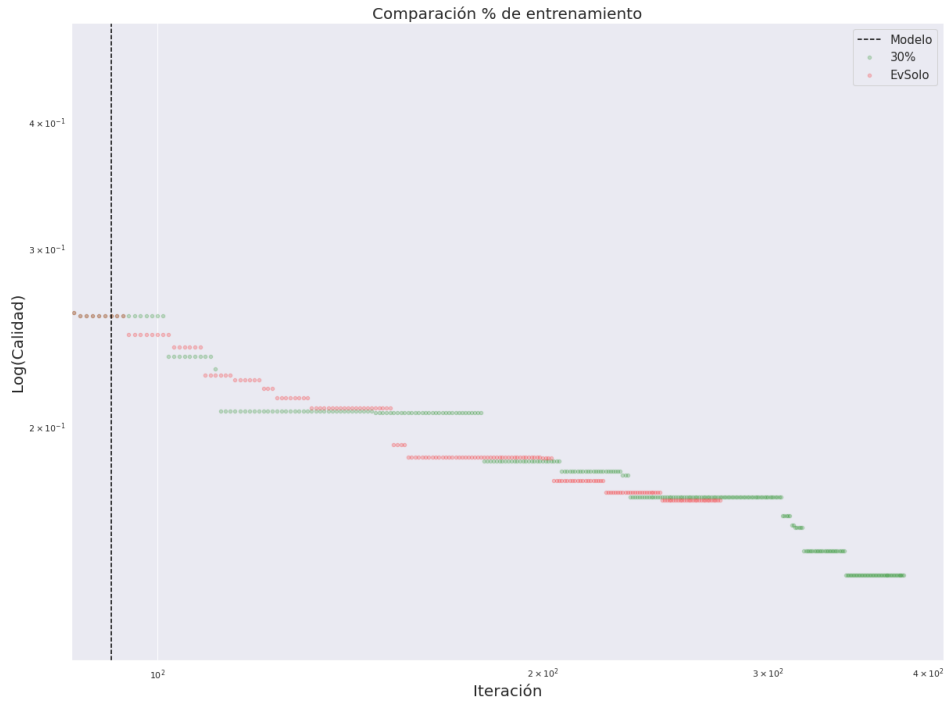


Figura 29: Promedio de las 8 mejores configuraciones v/s iteraciones - 30 % de evaluaciones
Fuente: Elaboración propia.

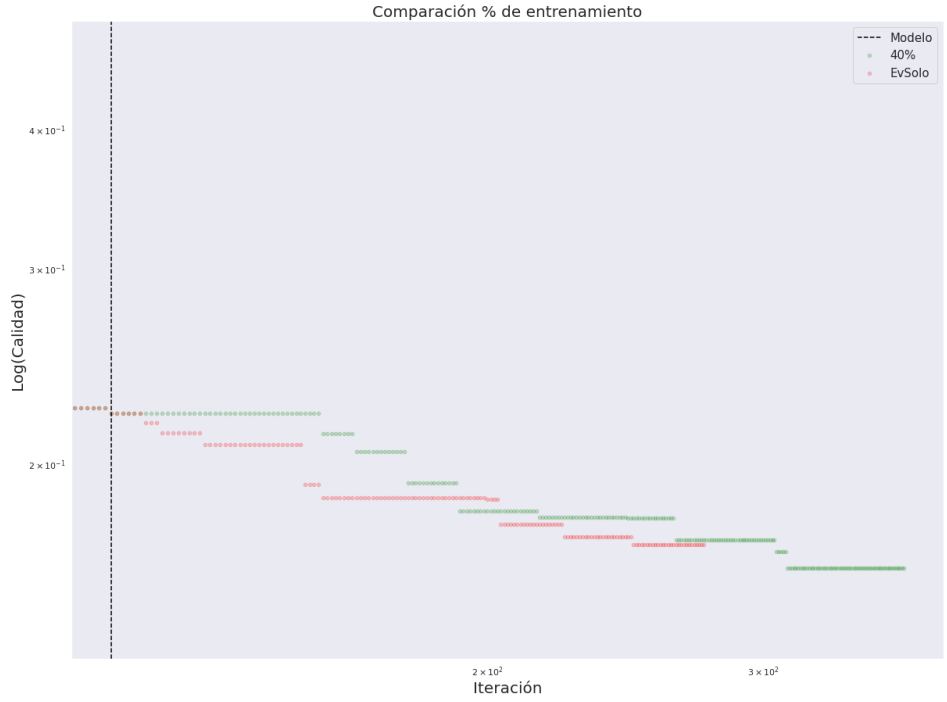


Figura 30: Promedio de las 8 mejores configuraciones v/s iteraciones - 40 % de evaluaciones
Fuente: Elaboración propia.

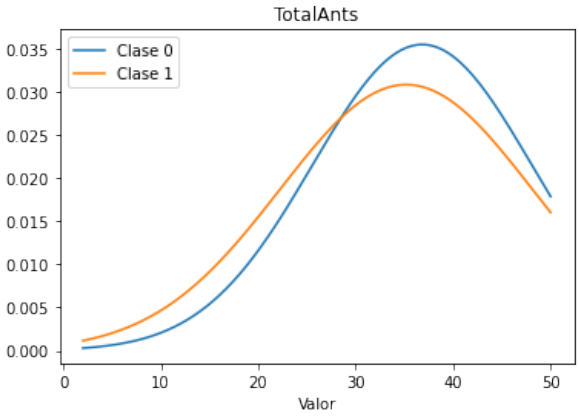


Figura 32: Función de densidad de probabilidad del total de hormigas
Fuente: Elaboración propia.

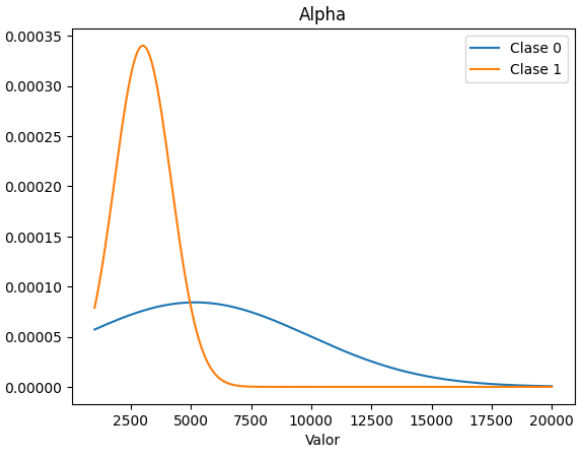


Figura 33: Función de densidad de probabilidad de α
Fuente: Elaboración propia.

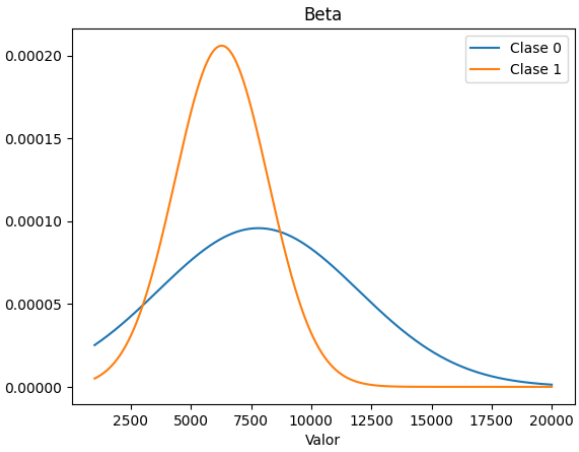


Figura 34: Función de densidad de probabilidad de β
Fuente: Elaboración propia.

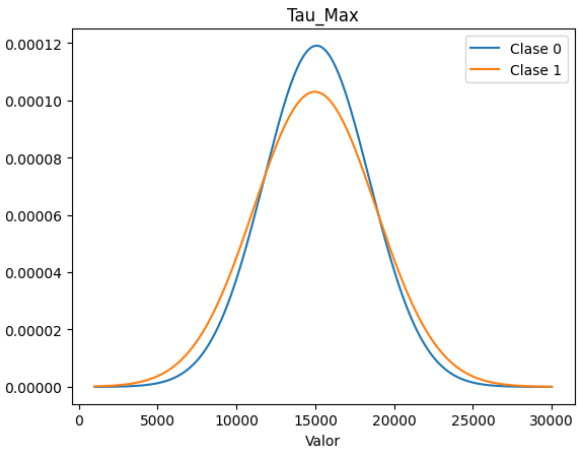


Figura 35: Función de densidad de probabilidad de τ_{max}
Fuente: Elaboración propia.

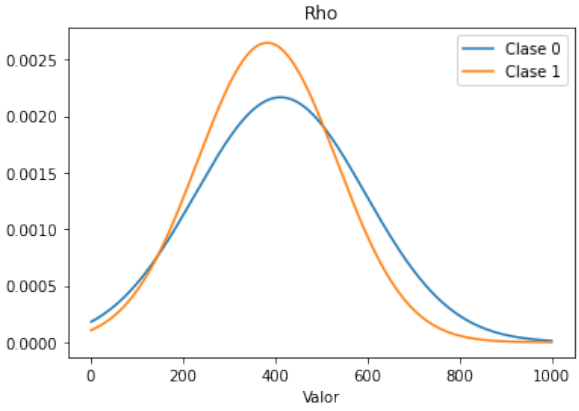


Figura 36: Función de densidad de probabilidad de ρ
Fuente: Elaboración propia.

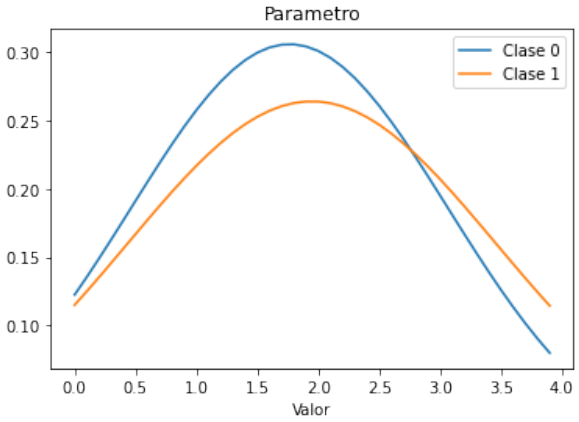


Figura 37: Función de densidad de probabilidad del parámetro seleccionado para la mutación

Fuente: Elaboración propia.

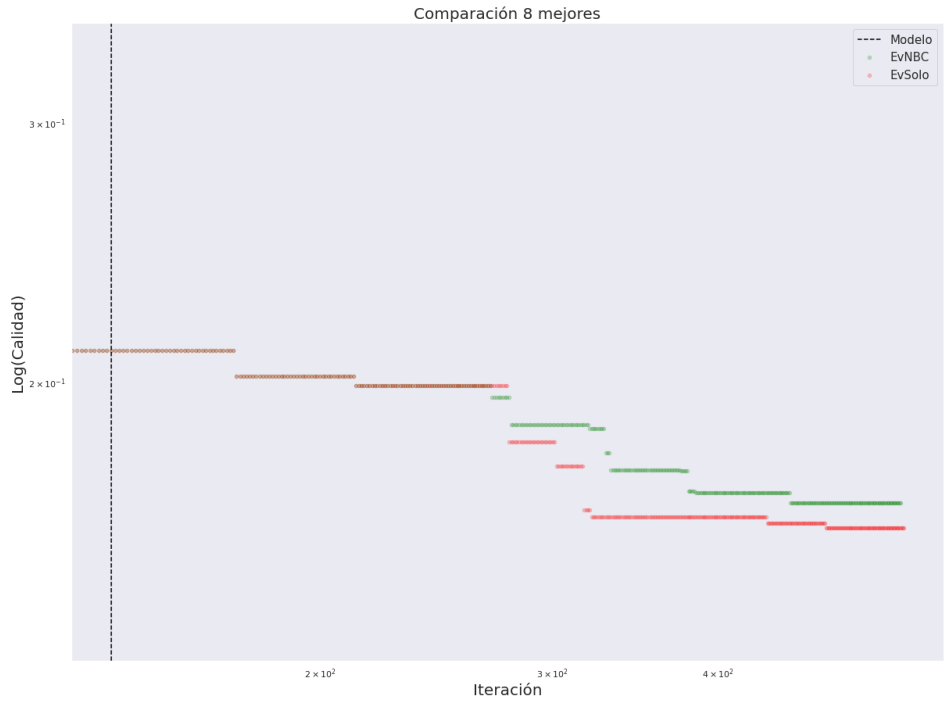


Figura 38: Promedio de configuraciones v/s iteración - Ejecución 3
Fuente: Elaboración propia.

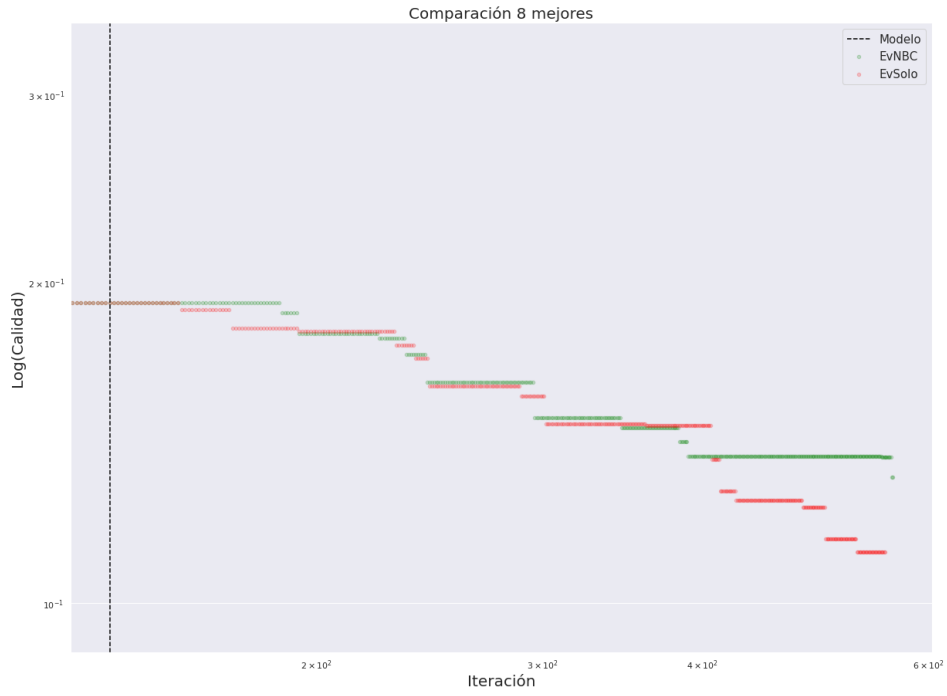


Figura 39: Promedio de configuraciones v/s iteración - Ejecución 4
Fuente: Elaboración propia.

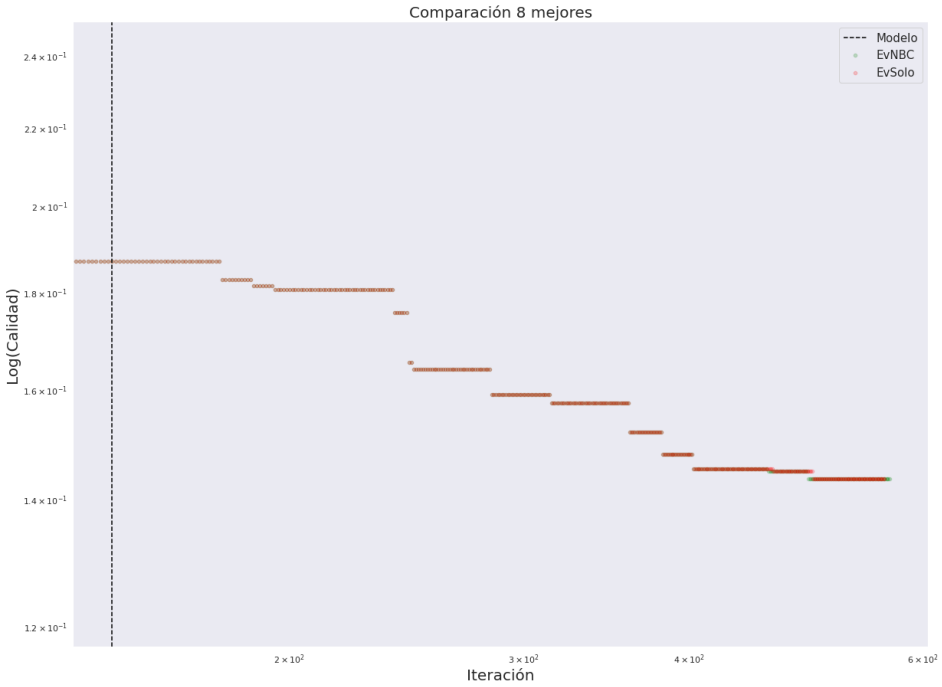


Figura 40: Promedio de configuraciones v/s iteración - Ejecución 1
Fuente: Elaboración propia.

CAPÍTULO 5
CONCLUSIONES

En esta memoria, se estudia la implementación de técnicas de obtención de conocimiento en el sintonizador de parámetros Evolutionary Calibrator (EVOCA), con el fin de mejorar su desempeño, en términos de la calidad de las configuraciones obtenidas o del tiempo de ejecución. En los problemas de optimización más complejos es poco factible evaluar todas las soluciones candidatas y escoger la mejor. Es por esto, que se utilizan metaheurísticas, algoritmos que pueden producir soluciones a los problemas de optimización de buena calidad en tiempos razonables. Sin embargo, el rendimiento de estos algoritmos está fuertemente vinculado a los valores de los parámetros que utilizan, razón por la cual se utilizan sintonizadores de parámetros como EVOCA. Los sintonizadores ayudan a determinar una configuración de parámetros para el algoritmo objetivo, de tal manera que tenga el mejor rendimiento en un conjunto de instancias.

En los primeros capítulos se definen formalmente los problemas de optimización, además de conceptos comúnmente utilizados en el trabajo de metaheurísticas. Luego, se define formalmente el problema de seteo de parámetros y se presentan distintos sintonizadores presentes en la literatura. Finalmente, se presenta una revisión de las técnicas de obtención de conocimiento, específicamente clasificadores, además de aplicaciones de técnicas de obtención de conocimiento en el contexto del problema de seteo de parámetros. Se vio que se ha sido posible utilizar la información proveniente de la misma metaheurística que se desea sintonizar, en conjunto con una técnica de obtención de conocimiento, para determinar los valores para sus parámetros. Además, se vio que es posible construir un modelo predictivo utilizando el rendimiento empírico de las configuraciones. Sin embargo, esta tarea requiere una extensa recopilación del rendimiento de una diversa gama de configuraciones en diversas instancias. Por otro lado, los modelos que fueron entrenados con los datos que se obtienen durante la ejecución del algoritmo no necesitan la recopilación de un conjunto de datos con antelación. Dado lo anterior, se concluye que para la propuesta de es deseable que el modelo sea construido durante el tiempo de ejecución, puesto que de esta manera puede adecuarse de manera dinámica sin requerir el tiempo de preparar un conjunto de datos de entrenamiento adecuado para cada metaheurística que se utilice.

Se propone una nueva versión de EVOCA, la cual utiliza un modelo de Naive Bayes Classifier (NBC). A esta nueva versión se le dio el nombre de EVOCA+NBC. El operador de mutación de EVOCA utiliza recursos evaluando configuraciones que no necesariamente son incluidas en la población final, no utilizando de la mejor manera los recursos dispuestos para el algoritmo, puesto que estos recursos pudieron haber sido utilizados en evaluar configuraciones con mayores probabilidades de ser incluidas en la población final. Es por esto, que se exploró utilizar la información de las configuraciones aceptadas y rechazadas para entrenar un modelo de clasificación NBC. Se utiliza NBC, porque a diferencia de otros modelos de clasificación este no requiere hiperparámetros, lo cual permite apreciar de forma más clara el impacto que

tiene utilizarlo. Además, esto sigue con el espíritu de diseño de EVOCA, ser un sintonizador simple, en donde el usuario debe preocuparse lo menos posible de la configuración inicial del sintonizador.

El problema de clasificación se planteó como un problema de clasificación binario, esto es, predecir la clase positiva (se utiliza el operador de mutación sobre el parámetro) o negativa (no se utiliza el operador de mutación sobre el parámetro). Las variables que considera el modelo son los valores de los parámetros del algoritmo objetivo más una variable que identifica el parámetro que se va a mutar. Los datos utilizados para entrenar el modelo son recopilados en tiempo de ejecución, dado que esto le permite adaptarse a las instancias del problema con las que se esté trabajando. De no ser así, ajustar un modelo a priori requeriría una gran cantidad de evaluaciones en una amplia gama de instancias de un problema, lo cual va más allá del objetivo de esta memoria.

El modelo se utiliza para predecir si vale la pena mutar el parámetro seleccionado antes de aplicar el operador de mutación. Si el modelo clasifica con la clase positiva aplica el operador de mutación sobre ese parámetro, en el caso contrario se cambia el parámetro a mutar. Esto se repite hasta que el modelo clasifique con la clase positiva o hasta intentar con todos los parámetros posibles. En el caso de intentar con todos los parámetros posibles y que en ningún caso sea de clase positiva, no se muta ningún parámetro y se empieza otra iteración. Finalmente, para determinar cuántas evaluaciones se utilizan para recolectar datos para entrenar el modelo, se propuso una regla derivada de I-race (ecuación 1). Esta considera el número de variables del algoritmo objetivo y entrega la razón de las evaluaciones que serán utilizadas para recopilar datos.

En la evaluación de propuesta de diseño, se trabajó con el algoritmo AK, un algoritmo basado en ACO adaptado para resolver el MKP. Se utilizaron 17 instancias de entrenamiento y 77 instancias de prueba. Para evaluar la propuesta se ejecutaron EVOCA y EVOCA+NBC en 5 semillas diferentes. Se evaluaron las 8 mejores configuraciones de cada población, puesto que el operador de cruce y el operador de mutación, con el fin de diversificar, incluyen individuos en la población que no necesariamente son mejores que el resto de la población. Cada configuración se ejecuta 10 veces en cada instancia de prueba, obteniendo 30.800 resultados de la calidad, expresados en GAP, y el tiempo de ejecución. Se observó que EVOCA+NBC obtienen mejores resultados en términos de calidad en el 63 % de las instancias y un mejor rendimiento promedio en el 49 % de las instancias. En términos del tiempo de ejecución, la propuesta obtiene un mejor resultado en el 56 % de las instancias y un mejor tiempo promedio en el 56 %. Para evaluar el modelo se utilizó Accuracy, Precision, Recall y F-Score. El modelo obtuvo un Accuracy promedio de 66 % y un F-Score promedio de 0.34. Además, se analizaron las magnitudes de las diferencias de los resultados. Al analizar las diferencias en los resultados a favor, en donde la propuesta alcanza una mejor calidad, se vio que en promedio es de 0.035 y la mayor diferencia encontrada es de 0.127. En el caso de los resultados no favorables, se vio un valor promedio de 0.030 y un mayor valor de 0.103. De la misma manera, con respecto al tiempo de ejecución, se vio que para los resultados a favor, dónde se obtiene un menor tiempo de ejecución, la diferencia en promedio es de 0.590 y el mayor

encontrada es de 4.220. En el caso de los resultados no favorables, se vio un valor promedio de 1.388 y un mayor valor de 4.570. En vista de lo anterior, se vio que la propuesta puede ser una mejora substancial, en donde en el mejor de los casos puede mejorar el GAP en 0.127 unidades y el tiempo de ejecución en 4.22 segundos. Sin embargo, es necesario trabajar en la consistencia de los resultados, puesto que en el peor casos puede empeorar los resultados en casi la misma magnitud. No obstante lo anterior, es necesario evaluar la propuesta en otras metaheurísticas para obtener conclusiones más robustas sobre el desempeño de la propuesta.

Al analizar individualmente cada ejecución, se vio que hay escenarios en donde EVOCA+NBC obtiene peores resultados. En la ejecución 3, en términos de calidad, se vio que EVOCA+NBC obtiene un mejor rendimiento en el 17 % de las instancias y un mejor rendimiento promedio 9 % de las instancias. Aún más, en la ejecución 1, al comparar las 8 mejores configuraciones, tanto EVOCA+NBC como EVOCA llegan al mismo resultado. Esto se debió a que el modelo clasificó en la gran mayoría de los casos con la clase positiva, lo que provocó que no se cambiará el parámetro que se mutó y se comportara como EVOCA por sí solo. En vista de lo anterior, se concluye que es indispensable para el modelo pueda predecir con precisión la clase negativa, puesto estas predicciones están ligadas a que el algoritmo cambie el parámetro que se muta, lo cual a su vez permite divergir con respecto a los resultados de EVOCA por sí solo.

Al evaluar el modelo se vio que obtiene un Accuracy promedio de 0.66, una Presicion de 0.66, un Recall promedio de 0.23 y un F-score promedio de 0.34. Esto muestra que tiene una mejor capacidad de identificar la clase positiva que la negativa, puesto que produce una baja cantidad de falsos positivos, pero una gran cantidad de falsos negativos. Esta es la mayor falencia de la propuesta, puesto que esto puede provocar que no se clasifique con la clase negativa, lo cual deja sin efecto el uso modelo al comportarse como EVOCA por sí solo. Por otro lado, al analizar las distribuciones utilizadas por NBC se concluyó que las variables más relevantes son α y β, puesto que las distribuciones para cada clase son significativamente diferentes a diferencia de los otros parámetros, en donde las distribuciones de cada clase no presentaban diferencias significativas.

El desempeño del modelo puede atribuirse a que otro clasificador podría ser más apropiado para este problema de clasificación, puesto que el supuesto que el los atributos de las clases se distribuyen de manera normal no sea el adecuado. Por ejemplo, puede que las configuraciones con un valor β cercano a 1 sean frecuentemente aceptadas en las primeras iteraciones del algoritmo y la distribución de la clase positiva esté centrada en valores cercanos a 1. Sin embargo, puede que en iteraciones posteriores del algoritmo, al cambiar el valor de los otros parámetros, configuraciones con valores cercanos a 1 sean comúnmente rechazadas. Esta situación puede no ser capturada por el clasificador y sea la razón de su mal desempeño en predecir la clase negativa. Otro factor que puede influir en el desempeño del modelo son los datos utilizados para entrenarlo. Se observó que en promedio por cada dato clasificado con la clase positiva hay aproximadamente seis de la clase negativa. Este desbalance puede influir de manera significativa en el rendimiento, porque no se tiene información suficiente

sobre las características que diferencian a las clases. En vista de lo anterior, puede explorarse la alternativa de utilizar otro clasificador que requiera de hiperparámetros y los resultados obtenidos en este trabajo pueden ser utilizados como benchmark.

Para mejorar el rendimiento del modelo se puede explorar la posibilidad de añadir más variables al modelo o eliminar variables del modelo. Por ejemplo, añadir la variación del parámetro que se va a mutar o bien la calidad del individuo cruzado. Otro aspecto a considerar es el de volver a entrenar el modelo durante la ejecución. Se puede continuar recolectando datos durante toda la ejecución del algoritmo y utilizarlos para actualizar el modelo.

Para evaluar la regla propuesta para determinar el porcentaje de evaluaciones utilizadas para entrenar el modelo, se comparó con ejecuciones de EVOCA+NBC que utilizaron el 5 %/10 %/15 %/30 %/40 % de las evaluaciones para entrenar el modelo. Se observó que existe la posibilidad de aumentar el porcentaje y que esto no tenga un efecto negativo. Sin embargo, al utilizar más evaluaciones en entrenar el modelo se retarda el uso de este, lo cual a su vez puede dar menos espacio para divergir de la población de EVOCA.

En ejecuciones en donde EVOCA+NB logra obtener mejores resultados finales, se compararon poblaciones intermedias con las poblaciones finales de EVOCA, en las respectivas ejecuciones. Para esto se seleccionaron poblaciones con un rendimiento promedio de las 8 mejores configuraciones lo más cercano a las de las poblaciones finales de EVOCA. Al comparar la calidad, se vio que en un 43 % de los casos se obtiene una solución de mejor calidad y en el 71 % de los casos un mejor promedio. Con respecto al tiempo de ejecución, se vio en un 100 % de las instancias un menor tiempo de ejecución y en el 100 % un menor tiempo de ejecución promedio. Si bien hay una pérdida de calidad, estos resultados fueron obtenidos con 7326 y 4821 evaluaciones para la ejecución 2 y 5 respectivamente, en vez de las 10.000 utilizadas por EVOCA.

En síntesis, el trabajo realizado muestra que utilizando un modelo de clasificación en EVOCA, se pudo obtener soluciones de mejor calidad. Además, muestra que es posible obtener resultados de calidad comparable a los de EVOCA por sí solo en un menor número de evaluaciones. De manera adicional, en el contexto de algoritmos evolutivos, la adición del modelo puede interpretarse como una probabilidad de ejecutar el operador mutación sobre un parámetro en cada iteración, la cual se determina de manera dinámica durante la ejecución del algoritmo.

Como trabajo futuro, se propone abordar el problema del desbalance de las clases. Para esto puede utilizarse técnicas de undersampling y/o oversampling. Finalmente, se propone explorar alternativas a no utilizar el operador de mutación en el caso que se intente con todos los parámetros posibles sin éxito. Para esto se pueden realizar mutaciones de dos parámetros a la vez para expandir el vecindario. De manera alternativa, puede explorarse mutar otra configuración de la población actual en vez del individuo cruzado actual.

ANEXOS

Tabla 9: Resultados de calidad de las soluciones parte II. Fuente: Elaboración Propia.

	EVOCA+NBC			EVOCA		
Instancia	Best	AVG	SD	Best	AVG	SD
OR10x500-0.25_7	0.265	1.178	0.627	0.392	1.153	0.641
OR10x500-0.25_8	0.435	1.349	0.825	0.435	1.279	0.674
OR10x500-0.25_9	0.466	1.260	0.663	0.443	1.306	0.768
OR10x500-0.50_1	0.207	0.501	0.225	0.207	0.520	0.242
OR10x500-0.50_10	0.206	0.525	0.311	0.207	0.546	0.333
OR10x500-0.50_2	0.213	0.507	0.239	0.234	0.532	0.334
OR10x500-0.50_3	0.194	0.483	0.193	0.201	0.512	0.261
OR10x500-0.50_4	0.195	0.479	0.290	0.209	0.452	0.209
OR10x500-0.50_5	0.216	0.521	0.211	0.216	0.531	0.209
OR10x500-0.50_6	0.146	0.427	0.264	0.178	0.446	0.280
OR10x500-0.50_7	0.161	0.467	0.283	0.161	0.463	0.185
OR10x500-0.50_8	0.224	0.509	0.247	0.199	0.511	0.238
OR10x500-0.50_9	0.216	0.546	0.319	0.216	0.549	0.305
OR10x500-0.75_1	0.093	0.239	0.115	0.087	0.228	0.092
OR10x500-0.75_10	0.101	0.229	0.079	0.106	0.226	0.080
OR10x500-0.75_3	0.111	0.264	0.126	0.111	0.256	0.099
OR10x500-0.75_4	0.093	0.221	0.096	0.095	0.219	0.093
OR10x500-0.75_5	0.055	0.211	0.097	0.055	0.215	0.106
OR10x500-0.75_6	0.134	0.301	0.094	0.145	0.303	0.106
OR10x500-0.75_7	0.066	0.213	0.101	0.062	0.217	0.100
OR10x500-0.75_8	0.080	0.254	0.120	0.110	0.254	0.094
OR10x500-0.75_9	0.095	0.235	0.101	0.095	0.234	0.090
OR30x500-0.25_1	0.989	2.165	0.976	1.036	2.143	0.930
OR30x500-0.25_10	1.014	2.288	1.150	1.014	2.218	0.993
OR30x500-0.25_2	0.777	1.834	0.942	0.777	1.909	1.015
OR30x500-0.25_3	1.053	2.241	0.960	1.053	2.353	1.093
OR30x500-0.25_4	1.010	2.525	1.494	1.010	2.507	1.480
OR30x500-0.25_6	0.908	2.285	1.352	0.848	2.388	1.455
OR30x500-0.25_7	0.865	2.296	1.089	0.845	2.267	1.000
OR30x500-0.25_8	1.052	2.378	1.136	1.149	2.280	0.991
OR30x500-0.25_9	1.044	2.298	1.078	1.100	2.317	1.041
OR30x500-0.50_1	0.314	0.784	0.278	0.314	0.775	0.240
OR30x500-0.50_10	0.387	0.877	0.375	0.371	0.868	0.291
OR30x500-0.50_2	0.493	0.949	0.473	0.513	0.964	0.414
OR30x500-0.50_5	0.423	0.856	0.288	0.423	0.883	0.356
OR30x500-0.50_6	0.424	0.887	0.348	0.472	0.890	0.295

Tabla 10: Resultados de calidad de las soluciones parte III. Fuente: Elaboración Propia.

	EVOCA+NBC			EVOCA		
Instancia	Best	AVG	SD	Best	AVG	SD
OR30x500-0.50_7	0.338	0.787	0.510	0.352	0.764	0.387
OR30x500-0.50_9	0.357	0.815	0.388	0.367	0.785	0.301
OR30x500-0.75_1	0.165	0.387	0.173	0.165	0.374	0.133
OR30x500-0.75_10	0.157	0.371	0.126	0.196	0.382	0.128
OR30x500-0.75_2	0.178	0.395	0.146	0.178	0.387	0.123
OR30x500-0.75_4	0.186	0.422	0.157	0.186	0.406	0.121
OR30x500-0.75_5	0.241	0.432	0.126	0.240	0.442	0.121
OR30x500-0.75_6	0.241	0.428	0.135	0.237	0.443	0.143
OR30x500-0.75_8	0.168	0.409	0.110	0.207	0.412	0.121
OR30x500-0.75_9	0.242	0.476	0.162	0.227	0.481	0.158
OR5x500-0.25_1	0.237	0.756	0.492	0.327	0.773	0.508
OR5x500-0.25_10	0.282	0.789	0.528	0.193	0.767	0.447
OR5x500-0.25_3	0.277	0.879	0.626	0.279	0.836	0.558
OR5x500-0.25_4	0.241	0.786	0.521	0.219	0.704	0.299
OR5x500-0.25_5	0.162	0.747	0.482	0.237	0.769	0.557
OR5x500-0.25_7	0.280	0.801	0.536	0.264	0.757	0.401
OR5x500-0.50_10	0.140	0.371	0.179	0.143	0.363	0.184
OR5x500-0.50_2	0.126	0.384	0.164	0.129	0.392	0.162
OR5x500-0.50_3	0.134	0.358	0.175	0.177	0.370	0.187
OR5x500-0.50_4	0.054	0.295	0.192	0.102	0.310	0.216
OR5x500-0.50_5	0.086	0.310	0.154	0.074	0.306	0.166
OR5x500-0.50_6	0.054	0.276	0.113	0.092	0.277	0.110
OR5x500-0.50_7	0.132	0.358	0.191	0.137	0.369	0.205
OR5x500-0.50_9	0.102	0.330	0.190	0.102	0.351	0.212
OR5x500-0.75_1	0.053	0.178	0.066	0.053	0.185	0.079
OR5x500-0.75_10	0.054	0.158	0.064	0.071	0.166	0.066
OR5x500-0.75_2	0.058	0.160	0.059	0.058	0.167	0.072
OR5x500-0.75_3	0.054	0.149	0.059	0.045	0.155	0.063
OR5x500-0.75_4	0.054	0.155	0.067	0.052	0.158	0.071
OR5x500-0.75_5	0.057	0.165	0.103	0.055	0.158	0.075
OR5x500-0.75_6	0.040	0.156	0.068	0.040	0.160	0.076
OR5x500-0.75_8	0.056	0.149	0.067	0.056	0.149	0.070

Tabla 11: Resultados de tiempo de ejecución parte II. Fuente: Elaboración Propia.

	EVOCA+NBC			EVOCA		
Instancia	Best	AVG	SD	Best	AVG	SD
OR10x500-0.25_7	69.770	72.516	0.820	70.560	72.572	0.798
OR10x500-0.25_8	67.480	70.422	0.882	67.900	70.474	0.810
OR10x500-0.25_9	68.320	71.444	0.912	68.800	71.510	0.962
OR10x500-0.50_1	119.820	122.689	1.244	120.800	122.893	1.168
OR10x500-0.50_10	117.560	122.017	1.345	117.870	122.379	1.354
OR10x500-0.50_2	120.620	123.682	1.287	121.240	123.782	1.190
OR10x500-0.50_3	120.290	122.981	1.199	120.040	123.064	1.238
OR10x500-0.50_4	121.160	123.657	1.201	121.520	123.717	1.154
OR10x500-0.50_5	120.100	123.124	1.158	119.870	123.087	1.257
OR10x500-0.50_6	120.210	122.655	1.125	120.330	122.702	1.230
OR10x500-0.50_7	121.540	124.112	1.163	121.490	124.029	1.242
OR10x500-0.50_8	120.780	123.217	1.128	120.780	123.179	1.229
OR10x500-0.50_9	121.350	123.488	1.239	121.000	123.613	1.387
OR10x500-0.75_1	154.540	158.694	1.470	153.630	158.680	1.677
OR10x500-0.75_10	153.280	158.794	1.579	153.660	158.619	1.678
OR10x500-0.75_3	155.130	158.979	1.525	154.720	158.736	1.759
OR10x500-0.75_4	154.380	159.283	1.516	153.990	159.136	1.765
OR10x500-0.75_5	155.460	160.068	1.595	155.190	159.975	1.724
OR10x500-0.75_6	153.420	158.363	1.540	154.280	158.358	1.635
OR10x500-0.75_7	155.060	159.208	1.545	155.580	159.075	1.553
OR10x500-0.75_8	154.750	159.293	1.562	155.130	159.126	1.615
OR10x500-0.75_9	155.080	159.179	1.453	153.300	158.840	1.573
OR30x500-0.25_1	68.530	71.460	0.839	69.280	71.617	0.907
OR30x500-0.25_10	68.320	70.872	0.763	68.280	70.895	0.801
OR30x500-0.25_2	68.350	70.669	0.794	68.390	70.742	0.885
OR30x500-0.25_3	68.240	70.678	0.763	68.910	70.690	0.814
OR30x500-0.25_4	67.800	70.638	0.910	68.070	70.699	0.947
OR30x500-0.25_6	68.820	71.571	0.933	68.960	71.529	0.927
OR30x500-0.25_7	68.690	70.752	0.817	68.740	70.771	0.825
OR30x500-0.25_8	67.610	70.219	0.864	68.060	70.368	0.973
OR30x500-0.25_9	67.930	70.657	1.133	68.060	70.834	1.201
OR30x500-0.50_1	125.070	128.270	1.273	126.270	128.750	1.396
OR30x500-0.50_10	122.560	127.919	1.707	123.830	128.149	1.856
OR30x500-0.50_2	124.980	127.708	1.165	125.770	128.127	1.380
OR30x500-0.50_5	125.030	127.767	1.173	125.070	128.095	1.372
OR30x500-0.50_6	125.980	128.607	1.295	125.560	128.767	1.422
OR30x500-0.50_7	125.420	128.720	1.310	125.510	128.947	1.441
OR30x500-0.50_9	126.350	129.151	1.468	125.600	129.461	1.649

Tabla 12: Resultados de tiempo de ejecución parte III. Fuente: Elaboración Propia.

	EVOCA+NBC			EVOCA		
Instancia	Best	AVG	SD	Best	AVG	SD
OR30x500-0.75_1	163.480	168.309	1.611	163.280	168.483	1.777
OR30x500-0.75_10	162.870	168.161	1.818	164.060	168.372	1.865
OR30x500-0.75_2	163.580	168.128	1.660	163.580	168.281	1.708
OR30x500-0.75_4	164.150	168.540	1.642	163.620	168.646	1.780
OR30x500-0.75_5	164.810	169.021	1.715	164.220	169.159	1.725
OR30x500-0.75_6	163.660	168.443	1.613	163.550	168.490	1.731
OR30x500-0.75_8	163.780	168.853	1.757	164.430	168.878	1.764
OR30x500-0.75_9	162.390	167.691	1.600	161.780	167.564	1.605
OR5x500-0.25_1	71.320	75.084	0.972	71.810	75.118	0.953
OR5x500-0.25_10	74.370	77.657	0.942	75.650	77.860	0.801
OR5x500-0.25_3	71.950	74.703	0.932	72.370	74.683	0.917
OR5x500-0.25_4	72.500	75.674	0.979	72.740	75.788	0.935
OR5x500-0.25_5	72.770	75.528	1.075	72.150	75.478	1.085
OR5x500-0.25_7	72.990	75.868	1.270	71.050	75.772	1.402
OR5x500-0.50_10	120.600	125.107	1.749	116.700	124.899	2.201
OR5x500-0.50_2	121.290	124.202	1.254	118.090	124.043	1.570
OR5x500-0.50_3	122.420	124.709	1.204	117.850	124.536	1.681
OR5x500-0.50_4	121.620	124.210	1.187	118.990	124.021	1.532
OR5x500-0.50_5	122.920	124.960	1.168	120.360	124.827	1.500
OR5x500-0.50_6	120.580	123.507	1.181	120.820	123.434	1.405
OR5x500-0.50_7	122.280	125.506	1.211	118.420	125.214	1.668
OR5x500-0.50_9	121.180	124.437	1.444	117.390	124.269	1.748
OR5x500-0.75_1	154.350	157.693	1.530	152.610	157.489	1.733
OR5x500-0.75_10	153.140	157.088	1.780	150.640	156.710	1.939
OR5x500-0.75_2	152.690	157.431	1.800	152.010	157.437	1.720
OR5x500-0.75_3	154.390	158.160	1.521	152.390	158.036	1.758
OR5x500-0.75_4	153.260	157.670	1.713	151.410	157.479	1.902
OR5x500-0.75_5	155.100	158.381	1.526	154.430	158.182	1.619
OR5x500-0.75_6	95.640	156.832	6.500	99.860	153.445	12.013
OR5x500-0.75_8	84.150	139.371	24.037	82.550	133.776	25.281

REFERENCIAS BIBLIOGRÁFICAS

[Abu Alfeilat *et al.*, 2019] Abu Alfeilat, H. A., Hassanat, A. B., Lasassmeh, O., Tarawneh, A. S., Alhasanat, M. B., Eyal Salman, H. S., y Prasath, V. S. (2019). Effects of distance measure choice on k-nearest neighbor classifier performance: a review. *Big data*, 7(4):221–248.

[Agrawal *et al.*, 2021] Agrawal, P., Abutarboush, H. F., Ganesh, T., y Mohamed, A. W. (2021). Metaheuristic algorithms on feature selection: A survey of one decade of research (2009-2019). *IEEE Access*, 9:26766–26791.

[Alaya *et al.*, 2004] Alaya, I., Solnon, C., y Ghedira, K. (2004). Ant algorithm for the multidimensional knapsack problem. En *International conference on Bioinspired Methods and their Applications (BIOMA 2004)*, pp. 63–72.

[Amra y Maghari, 2017] Amra, I. A. A. y Maghari, A. Y. (2017). Students performance prediction using knn and naïve bayesian. En *2017 8th International Conference on Information Technology (ICIT)*, pp. 909–913. IEEE.

[Back, 1992] Back, T. (1992). The interaction of mutation rate, selection, and self-adaptation within a genetic algorithm. En *Proc. 2nd Conference of Parallel Problem Solving from Nature, 1992*. Elsevier Science Publishers.

[Baghel *et al.*, 2012] Baghel, M., Agrawal, S., y Silakari, S. (2012). Survey of metaheuristic algorithms for combinatorial optimization. *International Journal of Computer Applications*, 58(19).

[Berrar, 2018] Berrar, D. (2018). Bayes' theorem and naive bayes classifier. *Encyclopedia of Bioinformatics and Computational Biology: ABC of Bioinformatics; Elsevier Science Publisher: Amsterdam, The Netherlands*, pp. 403–412.

[Bhavsar y Panchal, 2012] Bhavsar, H. y Panchal, M. H. (2012). A review on support vector machine for data classification. *International Journal of Advanced Research in Computer Engineering & Technology (IJARCET)*, 1(10):185–189.

[Blum y Roli, 2003] Blum, C. y Roli, A. (2003). Metaheuristics in combinatorial optimization: Overview and conceptual comparison. *ACM computing surveys (CSUR)*, 35(3):268–308.

[Brodersen *et al.*, 2010] Brodersen, K. H., Ong, C. S., Stephan, K. E., y Buhmann, J. M. (2010). The balanced accuracy and its posterior distribution. En *2010 20th international conference on pattern recognition*, pp. 3121–3124. IEEE.

[Budiman *et al.*, 2017] Budiman, Edy and Kridalaksana, Awang Harsa and Wati, Masna and others (2017). Performance of decision tree c4. 5 algorithm in student academic evaluation. En *International Conference on Computational Science and Technology*, pp. 380–389. Springer.

[Cervantes *et al.*, 2020] Cervantes, J., Garcia-Lamont, F., Rodríguez-Mazahua, L., y Lopez, A. (2020). A comprehensive survey on support vector machine classification: Applications, challenges and trends. *Neurocomputing*, 408:189–215.

[Corriveau *et al.*, 2016] Corriveau, G., Guilbault, R., Tahan, A., y Sabourin, R. (2016). Bayesian network as an adaptive parameter setting approach for genetic algorithms. *Complex & Intelligent Systems*, 2(1):1–22.

[Dokeroglu *et al.*, 2019] Dokeroglu, T., Sevinc, E., Kucukyilmaz, T., y Cosar, A. (2019). A survey on new generation metaheuristic algorithms. *Computers & Industrial Engineering*, 137:106040.

[Dorigo *et al.*, 1996] Dorigo, M., Maniezzo, V., y Colorni, A. (1996). Ant system: optimization by a colony of cooperating agents. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, 26(1):29–41.

[Drake, 2015] Drake, J. (2015). Benchmark instances for the multidimensional knapsack problem. *University of London*.

[Fitrani *et al.*, 2019] Fitrani, A., Rosid, M., Findawati, Y., Rahmawati, Y., y Anam, A. (2019). Implementation of id3 algorithm classification using web-based weka. En *Journal of Physics: Conference Series*, volumen 1381, p. 012036. IOP Publishing.

[Furey *et al.*, 2000] Furey, T. S., Cristianini, N., Duffy, N., Bednarski, D. W., Schummer, M., y Haussler, D. (2000). Support vector machine classification and validation of cancer tissue samples using microarray expression data. *Bioinformatics*, 16(10):906–914.

[Fushiki, 2011] Fushiki, T. (2011). Estimation of prediction error by using k-fold cross-validation. *Statistics and Computing*, 21(2):137–146.

[Geem *et al.*, 2001] Geem, Z. W., Kim, J. H., y Loganathan, G. V. (2001). A new heuristic optimization algorithm: harmony search. *simulation*, 76(2):60–68.

[Gupta y Gupta, 2019] Gupta, S. y Gupta, A. (2019). Dealing with noise problem in machine learning data-sets: A systematic review. *Procedia Computer Science*, 161:466–474.

[Holland, 1992] Holland, J. H. (1992). Genetic algorithms. *Scientific american*, 267(1):66–73.

[Hossin y Sulaiman, 2015] Hossin, M. y Sulaiman, M. (2015). A review on evaluation metrics for data classification evaluations. *International Journal of Data Mining & Knowledge Management Process*, 5(2):1.

[Huang *et al.*, 2019] Huang, C., Li, Y., y Yao, X. (2019). A survey of automatic parameter tuning methods for metaheuristics. *IEEE transactions on evolutionary computation*, 24(2):201–216.

[Hussain *et al.*, 2019] Hussain, K., Mohd Salleh, M. N., Cheng, S., y Shi, Y. (2019). Metaheuristic research: a comprehensive survey. *Artificial intelligence review*, 52(4):2191–2233.

[Hussain *et al.*, 2011] Hussain, M., Wajid, S. K., Elzaart, A., y Berbar, M. (2011). A comparison of svm kernel functions for breast cancer detection. En *2011 eighth international conference computer graphics, imaging and visualization*, pp. 145–150. IEEE.

[Hutter *et al.*, 2010] Hutter, F., Hoos, H. H., y Leyton-Brown, K. (2010). Sequential model-based optimization for general algorithm configuration (extended version). *Technical Report TR-2010–10, University of British Columbia, Computer Science, Tech. Rep.*

[Islam *et al.*, 2007] Islam, M. J., Wu, Q. J., Ahmadi, M., y Sid-Ahmed, M. A. (2007). Investigating the performance of naive-bayes classifiers and k-nearest neighbor classifiers. En *2007 International Conference on Convergence Information Technology (ICCIT 2007)*, pp. 1541–1546. IEEE.

[Kamel *et al.*, 2019] Kamel, H., Abdulah, D., y Al-Tuwaijari, J. M. (2019). Cancer classification using gaussian naive bayes algorithm. En *2019 International Engineering Conference (IEC)*, pp. 165–170. IEEE.

[Kashan, 2009] Kashan, A. H. (2009). League championship algorithm: a new algorithm for numerical function optimization. En *2009 international conference of soft computing and pattern recognition*, pp. 43–48. IEEE.

[Kennedy y Eberhart, 1995] Kennedy, J. y Eberhart, R. (1995). Particle swarm optimization. En *Proceedings of ICNN'95-international conference on neural networks*, volumen 4, pp. 1942–1948. IEEE.

[Kirkpatrick *et al.*, 1983] Kirkpatrick, S., Gelatt Jr, C. D., y Vecchi, M. P. (1983). Optimization by simulated annealing. *science*, 220(4598):671–680.

[Lessmann *et al.*, 2011] Lessmann, S., Caserta, M., y Arango, I. M. (2011). Tuning metaheuristics: A data mining based approach for particle swarm optimization. *Expert Systems with Applications*, 38(10):12826–12838.

[López-Ibáñez *et al.*, 2016] López-Ibáñez, M., Dubois-Lacoste, J., Cáceres, L. P., Birattari, M., y Stützle, T. (2016). The irace package: Iterated racing for automatic algorithm configuration. *Operations Research Perspectives*, 3:43–58.

[Montero *et al.*, 2014] Montero, E., Riff, M.-C., y Neveu, B. (2014). A beginner's guide to tuning methods. *Applied Soft Computing*, 17:39–51.

[Mukherjee y Sharma, 2012] Mukherjee, S. y Sharma, N. (2012). Intrusion detection using naive bayes classifier with feature reduction. *Procedia Technology*, 4:119–128.

[Neumaier *et al.*, 2005] Neumaier, A., Shcherbina, O., Huyer, W., y Vinkó, T. (2005). A comparison of complete global optimization solvers. *Mathematical programming*, 103(2):335–356.

[Osorio y Cuaya, 2005] Osorio, M. A. y Cuaya, G. (2005). Hard problem generation for mkp. En *Sixth Mexican International Conference on Computer Science (ENC'05)*, pp. 290–295. IEEE.

[Pavón *et al.*, 2009] Pavón, R., Díaz, F., Laza, R., y Luzón, V. (2009). Automatic parameter tuning with a bayesian case-based reasoning system. a case of study. *Expert Systems With Applications*, 36(2):3407–3420.

[Prati *et al.*, 2015] Prati, R. C., Batista, G. E., y Silva, D. F. (2015). Class imbalance revisited: a new experimental setup to assess the performance of treatment methods. *Knowledge and Information Systems*, 45(1):247–270.

[Qiu *et al.*, 2016] Qiu, J., Wu, Q., Ding, G., Xu, Y., y Feng, S. (2016). A survey of machine learning for big data processing. *EURASIP Journal on Advances in Signal Processing*, 2016(1):1–16.

[Riff y Montero, 2013] Riff, M.-C. y Montero, E. (2013). A new algorithm for reducing metaheuristic design effort. En *2013 IEEE Congress on Evolutionary Computation*, pp. 3283–3290. IEEE.

[Rojas *et al.*, 2002] Rojas, I., González, J., Pomares, H., Merelo, J., Castillo, P., y Romero, G. (2002). Statistical analysis of the main parameters involved in the design of a genetic algorithm. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, 32(1):31–37.

[Schrijver, 2005] Schrijver, A. (2005). On the history of combinatorial optimization (till 1960). *Handbooks in operations research and management science*, 12:1–68.

[Singh y Gupta, 2014] Singh, S. y Gupta, P. (2014). Comparative study id3, cart and c4. 5 decision tree algorithm: a survey. *International Journal of Advanced Information Science and Technology (IJAIST)*, 27(27):97–103.

[Sokolova *et al.*, 2006] Sokolova, M., Japkowicz, N., y Szpakowicz, S. (2006). Beyond accuracy, f-score and roc: a family of discriminant measures for performance evaluation. En *Australasian joint conference on artificial intelligence*, pp. 1015–1021. Springer.

[Stutzle y Hoos, 1997] Stutzle, T. y Hoos, H. (1997). Max-min ant system and local search for the traveling salesman problem. En *Proceedings of 1997 IEEE international conference on evolutionary computation (ICEC'97)*, pp. 309–314. IEEE.

[Thierens, 2005] Thierens, D. (2005). An adaptive pursuit strategy for allocating operator probabilities. En *Proceedings of the 7th annual conference on Genetic and evolutionary computation*, pp. 1539–1546.

[Wolpert y Macready, 1997] Wolpert, D. H. y Macready, W. G. (1997). No free lunch theorems for optimization. *IEEE transactions on evolutionary computation*, 1(1):67–82.

[Yang *et al.*, 2014] Yang, X.-S., Deb, S., y Fong, S. (2014). Metaheuristic algorithms: optimal balance of intensification and diversification. *Applied Mathematics & Information Sciences*, 8(3):977.

[Zennaki y Ech-Cherif, 2010] Zennaki, M. y Ech-Cherif, A. (2010). A new machine learning based approach for tuning metaheuristics for the solution of hard combinatorial optimization problems. *Journal of Applied Sciences(Faisalabad)*, 10(18):1991–2000.

[Zhang *et al.*, 2007] Zhang, J., Chung, H. S.-H., y Lo, W.-L. (2007). Clustering-based adaptive crossover and mutation probabilities for genetic algorithms. *IEEE Transactions on evolutionary Computation*, 11(3):326–335.

[Zhang, 2016] Zhang, Z. (2016). Introduction to machine learning: k-nearest neighbors. *Annals of translational medicine*, 4(11).

[Zulfikar *et al.*, 2021] Zulfikar, W., Taufik, I., Atmadja, A., y Rahayu, R. (2021). A classification model for student exchange using cart algorithm. En *IOP Conference Series: Materials Science and Engineering*, volumen 1098, p. 032054. IOP Publishing.