الجمهورية الشعبية الديمقراطية الجزائرية
People's Democratic Republic of Algeria
وزارة التعليم العالي و البحث العلمي
Ministry of Higher Education and Scientific Research
جامعة سعد دحلب - االبليدة 1
UNIVERSITÉ SAAD DAHLEB – BLIDA 1

Master's Thesis

To obtain the diploma of **Master's Degree**

Field of Study: **Computer Science**

Specialization: **Naturel language processing**

Theme

Data Mining Techniques and Metaheuristics for Problem Solving : Application to the SAT Problem

---

Presented by
BERGOUGUI Chaima
BELACEL Intissar

Defended on: **[June, 2025]**
In front of the jury composed of

Mr. [Jury Member Name]	President of the Jury
Mrs. Hireche Celia	Thesis Supervisor
Mr. [Jury Member Name]	Examiner

Academic Year: 2024/2025

Dedication

It was only by His facilitation that we were able to begin this journey, and we only reached the goals by His guidance, and we only achieved them by His grace. ***Alhamdulillah*** *who granted me success to complete this step in my academic journey.*

To my ambitious ***self****, who stumbled and grew tired, but still kept going. I am proud of the strength and patience you showed throughout this journey. Thank you for not giving up.*

To my ***dear father****, who planted ambition in my heart and taught me to love knowledge. You have always been my support, with endless love and constant encouragement. This success is yours before it is mine. I love you, Dad. May Allah protect you, and may we always be a source of joy and pride for you.*

To my ***dear mother****, you have always been a role model in goodness, a kind teacher, a caring guide, and a true friend. Despite all the responsibilities and pressures, you were always my support and source of warmth. May Allah protect you for us a flower that never withers.*

To my ***aunt Hakima*** *and second mother, the one who loves to see me succeed more than anyone after my parents. May Allah protect you and keep you in my life as a constant support, a true friend, and a pure heart that never changes.*

To my dear little brothers and sisters: ***Amira****,* ***Abdou****,* ***Israa****, and* ***Khalil*** *may Allah protect you always. I pray you reach places far greater and higher than I ever have, and that you always make us proud.*

To my dear friends and classmates who shared this journey with me in every moment especially ***Bilal****,* ***Ibtissem****, and* ***Malak****. Thank you for your love, your genuine friendship, and for always being there for me. With you, this journey was lighter and more joyful.*

To all my dear teachers who taught me in Bouira and Blida University, thank you for every lesson and word of guidance. A special thank you to ***Melyara Mezzi****,* ***Celia Hireche****, and* ***Tebbi Hanane*** *for their constant support and encouragement. You were among the best and kindest teachers I've ever had.*

Bergougui Chaima.
Bouira, June 12, 2025.

Dedication

This dissertation is dedicated to my parents,
who instilled in me the virtues of perseverance and commitment,
and who relentlessly encouraged me to strive for excellence.
To my brothers,
my sister,
my relatives,
and all my friends

BELACEL Intissar

ACKNOWLEDGEMENT

*We would like to express our gratitude to our supervisor, **Dr. Hireche Celia**, for her guidance, ideas, and support. She encouraged us to express ourselves freely and gave us the motivation to go further and do our best. Her mentorship shaped the direction and quality of this research.*

We would like to sincerely thank the President for the honor of chairing the defense jury and for taking the time to evaluate our work.

We would like to thank the examiners for providing us the opportunity to learn from their insightful remarks, constructive feedback, and valuable judgment.

We extend our heartfelt thanks to all the professors of our Natural Language Processing specialty for their efforts, support, and dedication.

*A special and sincere appreciation goes to the head of the specialty, **Ms. Mezzi Melyara**, whose commitment, kindness, and excellence in her work have truly inspired us. Words of thanks are not enough to express our gratitude.*

Finally, we would like to thank our university, the Computer Science Department, all the teachers who guided us throughout our academic journey, and everyone who contributed, whether directly or indirectly, to the completion of this work. Your support has been truly appreciated.

Chaima & Intissar.
Algeria, June 13, 2025.

CONTENTS

LIST OF ACRONYMS

KDD	Knowledge Discovery in Databases
KNN	k-Nearest Neighbors
BMC	Bounded Model Checking
CBR	Case-Based Reasoning
SVM	Support Vector Machines
K-Means	K-Means Clustering Algorithm
DBSCAN	Density-Based Spatial Clustering of Applications with Noise
MinPts	Minimum Points required to form a dense region (DBSCAN parameter)
PSO	Particle Swarm Optimization
ACO	Ant Colony Optimization
ABC	Artificial Bee Colony
GWO	Grey Wolf Optimizer
EHO	Elephant Herding Optimization
SAT	Boolean Satisfiability Problem
CNF	Conjunctive Normal Form
NP	Non-deterministic Polynomial Time
DPLL	Davis-Putnam-Logemann-Loveland Algorithm
CDCL	Conflict-Driven Clause Learning
GA	Genetic Algorithm
IEHO-M	Improved EHO with Mutation
IEHO-MRD	Combined Mutation and Re-Division for Improved EHO
IEHO-RD	Improved EHO with Re-Division

LIST OF FIGURES

LIST OF TABLES

LIST OF ALGORITHMS

Abstract

This research explores the integration of data mining techniques with metaheuristic algorithms to address hard combinatorial problems, with a particular focus on the Boolean Satisfiability Problem (SAT). The core metaheuristic employed is Elephant Herding Optimization (EHO), a swarm-based algorithm inspired by the social behavior of elephants, known for its balance between exploration and exploitation.

To align EHO with the discrete nature of the SAT problem, a first adaptation phase was carried out. During this stage, stagnation emerged as a significant limitation, affecting the algorithm's ability to maintain progress toward better solutions. To mitigate this, three strategies were proposed: mutation, re-division, and a combination of both. These simple improvements led to a notable reduction in stagnation and significantly increased the satisfiability rate, particularly with the mutation strategy, which also offered the shortest execution time in most test cases.

The second phase consisted of selecting suitable data mining techniques to hybridize with EHO, aiming to further enhance performance. K-Means and DBSCAN clustering methods were chosen due to their alignment with EHO's internal mechanisms, such as clan division and worst-solution handling. K-Means was integrated by replacing random clan division with similarity-based grouping. DBSCAN, on the other hand, was adapted both to replace the random clan division using a density-based similarity approach and, in addition, to detect and manage outliers by improving the considered worst individuals through guided reintegration. These hybridizations resulted in substantial performance gains, with the DBSCAN-based approach achieving the best balance of high satisfiability and low execution time. These findings confirm that combining data-driven clustering with swarm-based metaheuristics offers a promising direction for solving complex discrete problems.

Keywords: Metaheuristic, Swarm intelligence, EHO, Data Mining, Clustering, Problem Solving, SAT, Stagnation, Hybridization.

Résumé

Cette recherche exploite l'intégration des techniques de fouilles de données aux métaheuristiques pour résoudre des problèmes combinatoires complexes, en particulier le problème de satisfiabilité booléenne (SAT). La métaheuristique utilisée est l'Elephant Herding Optimization (EHO), un algorithme basé sur l'intelligence en essaim, inspiré du comportement social des éléphants, connu pour son équilibre entre exploration et exploitation.

Pour adapter EHO à la nature discrète du problème SAT, une première phase d'adaptation a été réalisée. Durant cette étape, la stagnation s'est avérée être une limite importante, réduisant la capacité de l'algorithme à progresser vers de meilleures solutions. Pour y remédier, trois stratégies ont été proposées : la mutation, la redivision et une combinaison des deux. Ces améliorations simples ont permis de réduire la stagnation et d'augmenter considérablement le taux de satisfiabilité, en particulier avec la stratégie de mutation, qui a aussi offert le temps d'exécution le plus court dans la plupart des cas.

La deuxième phase a consisté à sélectionner des techniques de fouilles de données pour hybrider EHO, afin d'améliorer davantage ses performances. Les méthodes de clustering K-Means et DBSCAN ont été choisies en raison de leur compatibilité avec les mécanismes internes de EHO, comme la division en clans ou la gestion des mauvaises solutions. K-Means a remplacé la division aléatoire des clans par un regroupement basé sur la similarité. DBSCAN a été utilisé à la fois pour remplacer cette division par une approche basée sur la densité et pour détecter et corriger les solutions de mauvaise qualité. Ces hybridations ont permis d'obtenir des performances nettement meilleures, avec DBSCAN atteignant le meilleur équilibre entre taux de satisfiabilité élevé et temps d'exécution réduit. Ces résultats montrent que combiner des techniques de clustering avec des métaheuristiques de type intelligence en essaim est une voie prometteuse pour résoudre des problèmes discrets complexes.

Mots-clés: Métaheuristique, intelligence en essaim, EHO, Fouille de données, Regroupement (Clustering), Résolution de problèmes, SAT, Stagnation, Hybridation.

ملخص

تستكشف هذه الدراسة دمج تقنيات التنقيب في البيانات مع الخوارزميات الاستكشافية لحل المشاكل التوافقية المعقدة، مع التركيز بشكل خاص على مسألة SAT. الخوارزمية الأساسية المستخدمة هي "تحسين قطيع الأفيال" (EHO)، وهي خوارزمية مستوحاة من الذكاء الاجتماعي للأفيال، وتتميز بالتوازن بين الاستكشاف والاستغلال.

لتكييف EHO مع الطابع المنفصل لمشكلة SAT، تم تنفيذ مرحلة تكييف أولية. خلال هذه المرحلة، ظهرت مشكلة الركود كعائق رئيسي، مما أثر على قدرة الخوارزمية على التقدم نحو حلول أفضل. وللتغلب على ذلك، تم اقتراح ثلاث استراتيجيات: الطفرة، وإعادة التقسيم، والجمع بينهما. وقد أدت هذه التحسينات البسيطة إلى تقليل الركود بشكل ملحوظ وزيادة معدل الإرضاء، خاصة باستخدام استراتيجية الطفرة، التي أظهرت أيضًا أقصر وقت تنفيذ في معظم الحالات.

المرحلة الثانية تمثلت في اختيار تقنيات مناسبة من تنقيب البيانات لدمجها مع خوارزمية EHO، بهدف تعزيز الأداء بشكل أكبر. تم اختيار طريقتي التجميع DBSCAN KMeans نظراً لتوافقهما مع آليات عمل EHO، مثل تقسيم العشائر ومعالجة أسوأ الحلول. تم دمج KMeans من خلال استبدال تقسيم العشائر العشوائي بتجميع يعتمد على التشابه. أما DBSCAN فقد تم تكييفه ليقوم أيضًا باستبدال التقسيم العشوائي للعشائر باستخدام نهج تشابه قائم على الكثافة، بالإضافة إلى اكتشاف القيم الشاذة والتعامل معها عبر تحسين الأفراد الأسوأ من خلال إعادة إدماج موجهة. وقد أدت هذه الادماجات إلى تحقيق تحسينات كبيرة في الأداء، حيث حقق نهج DBSCAN أفضل توازن بين معدل الإشباع العالي وزمن التنفيذ المنخفض. تُظهر هذه النتائج أن الجمع بين تقنيات التجميع و الخوارزميات الاستكشافية القائمة على الذكاء الجماعي يُعد مساراً واعداً لحل المشكلات المنفصلة والمعقدة.

الكلمات المفتاحية: الخوارزميات الاستكشافية، الذكاء الجماعي، OHE، التنقيب في البيانات، التجميع، حل المشكلات، TAS، الركود، الدمج.

Part I

Introduction

GENERAL INTRODUCTION

Solving complex real-world problems often requires addressing large-scale optimization tasks. Many such problems like scheduling, resource allocation, and strategic planning belong to the class of NP-complete problems. These are known for their computational complexity, as no algorithm can solve them exactly in polynomial time for all cases. Among these, the Boolean Satisfiability Problem (SAT) is a fundamental and widely studied example, with applications ranging from artificial intelligence and hardware verification to cybersecurity and cryptography.

To handle such problems, exact methods often fall short as the solution space grows exponentially with problem size. This limitation has led to increasing reliance on heuristic and metaheuristic approaches. Heuristics provide fast, approximate solutions based on problem-specific rules, while metaheuristics offer more general frameworks inspired by nature or mathematical processes. These methods, especially metaheuristics, strike a practical balance between solution quality and computational efficiency, making them particularly suitable for hard combinatorial problems like SAT.

Among the families of metaheuristics, swarm-based algorithms have gained significant attention due to their distributed nature, simplicity, and robust performance across diverse applications. Of the recently developed algorithms, Elephant Herding Optimization (EHO), inspired by the herding behavior of elephants, stands out as a promising approach whose adaptation to solving SAT problems offers interesting potential. In addition to metaheuristics, data mining techniques also provide valuable tools for extracting structure from complex datasets. This synergy is particularly relevant in the context of hybrid approaches, where combining metaheuristics like EHO with clustering methods from data mining can help overcome specific challenges and enhance solution quality.

This work aims to investigate and enhance the EHO algorithm to efficiently solve SAT problems. The manuscript follows a structured methodology that progresses in several stages:

- First, a theoretical foundation is established through a state-of-the-art study (Part II), which presents essential concepts of data mining (Chapter 1), introduces metaheuristic algorithms with a focus on Elephant Herding Optimization (Chapter 2), and discusses the SAT problem and its variants along with existing solving strategies (Chapter 3). This background provides the conceptual support required for the proposed algorithmic contributions.
- Throughout third part (Part III),The EHO algorithm is first adapted and implemented to solve the SAT problem, followed by initial testing to assess its behavior and address observed limitations (chapitre 04). Subsequently, hybridization with selected data mining techniques is

introduced to further enhance performance (Chapter 05),followed by a comprehensive experimental evaluation to validate the effectiveness of the proposed improvements (Chapter 6).

- Finally, a general conclusion is presented, summarizing the main outcomes of the study, along with some perspectives and potential directions for future research.

Part II

State of the Art

CHAPTER 1

DATA MINING TECHNIQUES

1.1 Introduction

With the exponential growth of data and the diversity of its sources, extracting valuable knowledge has become increasingly challenging. This led to the emergence of Data Mining, a powerful solution that leverages artificial intelligence, machine learning, and statistical techniques to uncover hidden patterns and relationships within massive amounts of information.

This chapter is dedicated to Data Mining by exploring its essential methodologies and applications. Key techniques like clustering and classification for pattern extraction are first, discussed. Preprocessing techniques such as cleaning and transformation to enhance data quality are then discussed. Finally, post-processing, which includes evaluation and visualization is covered.

1.2 Understanding Data

Before applying data mining techniques, it is essential to understand the nature of data, its different forms, and the challenges it presents. The quality of data directly impacts the effectiveness of analyses and the results obtained.

Data analysis begins with identifying and collecting relevant datasets from various sources. It is crucial to document their properties, such as format, number of records, and attributes. A deeper exploration through queries and visualizations helps uncover relationships, patterns, or inconsistencies. Finally, verifying data quality is essential to detect errors and missing values, ensuring optimal processing [1].

Data is generally classified into two main categories [2, 3]: **qualitative** and **quantitative**.

- **Qualitative data** (categorical) includes:
  - Nominal attributes (unordered categories, e.g., color, product type).
  - Binary attributes (two possible states, e.g., true/false).
  - Ordinal attributes (ordered categories without measurable differences, e.g., education level).

➢ **Quantitative data** (numerical) includes:

- Interval-scaled attributes (meaningful differences but no true zero, e.g., temperature).
- Ratio-scaled attributes (with a true zero, allowing ratio comparisons, e.g., weight, income).

The type of data directly influences how it is cleaned, transformed, and analyzed, making this step crucial before processing.

1.3 Data Mining: fundamental concepts

Data mining refers to extracting or "mining" knowledge from large amounts of data.
Also referred to as Knowledge Discovery in Databases (KDD), it is the process of uncovering interesting patterns, models, and other valuable insights from massive datasets. It involves more than just collecting data, be focuses on analyzing and extracting meaningful knowledge that can drive decision-making [3].

A data mining process integrates concepts from statistics; such as sampling, estimation, and hypothesis testing, as well as from artificial intelligence, pattern recognition, and machine learning, including search algorithms, modeling techniques, and learning theories. It also draws on fields such as optimization, evolutionary computing, information theory, signal processing, visualization, and information retrieval [2]. The illustrated diagram (1.1) exhibits the relationship between data mining and other fields.

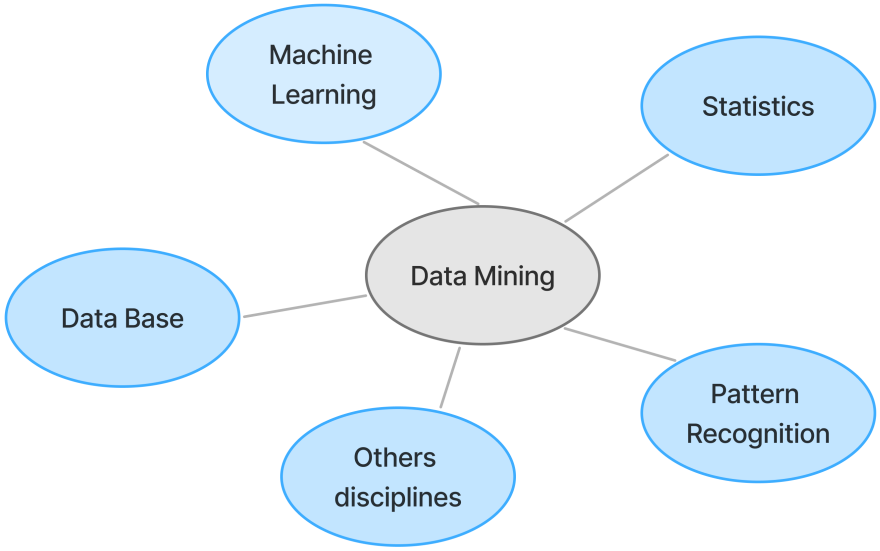

Figure 1.1: Data Mining as the Integration of Multiple Disciplines

The data mining process is a structured sequence of steps aimed at transforming raw input data into actionable knowledge, as illustrated in Figure 1.2. This process begins with data preprocessing, where the raw data is cleaned, transformed, and prepared for analysis. Next, the data mining phase applies algorithms and techniques to extract meaningful patterns and insights from the processed data. Finally, a post-processing stage evaluates and refines the extracted patterns, ensuring that the discovered knowledge is both accurate and interpretable [3][2].

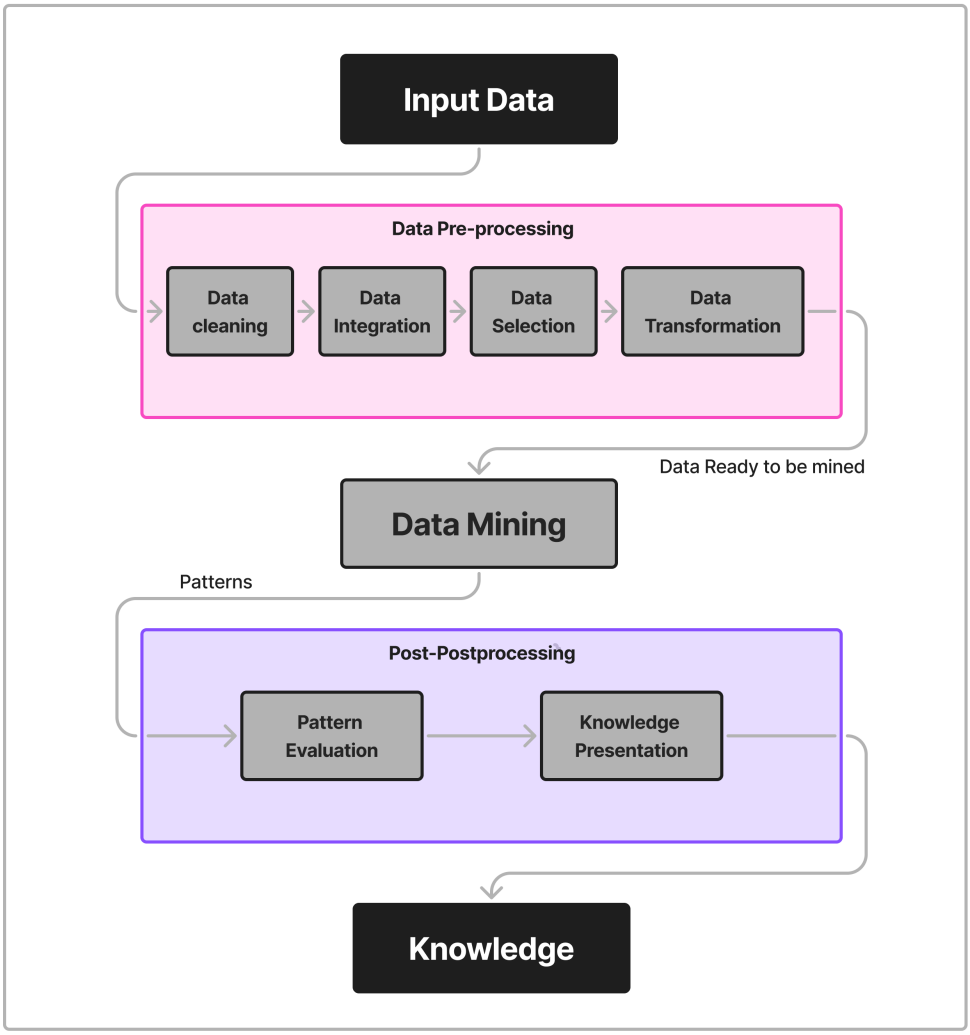

Figure 1.2: Main Phases of a Data Mining process

The following subsections will delve deeper into each main step preprocessing, mining, and post-processing highlighting their specific roles and techniques.

1.3.1 Data preprocessing

Today's real-world databases often contain noisy, incomplete , and inconsistent data due to their large size and heterogeneous sources. such low-quality data can significantly impact the quality of data mining process and lead to poor mining results. The quality of data plays a crucial role in a data mining process. To ensure data quality and enhance the effectiveness of data mining techniques, a data preprocessing step is essential.

It involves several key techniques:

1. **Data Cleaning** ensures data reliability by handling missing values through imputation methods (e.g., mean or median replacement) and reducing noise using techniques like binning, clustering, and regression [3, 4]

2. **Data Integration** combines data from multiple sources, resolving redundancies and inconsistencies using object identification and correlation analysis [3, 4].

3. **Data Transformation and Discretization** convert data into suitable formats for analysis, including smoothing (reducing noise), aggregation (summarizing data), generalization (abstracting details), normalization (scaling data), and discretization (categorizing continuous attributes) [4].

4. **Data Reduction** optimizes storage and processing by reducing dataset size while preserving essential patterns through dimensionality reduction, numerosity reduction, and sampling [4].

Note: Data preprocessing does not require executing all stages for every dataset; it depends on the nature of the data and the specific problem. Typically, the process starts with data cleaning to remove missing values and noise, followed by data integration if multiple sources are involved. Then, data transformation is applied through normalization or discretization, and finally, data reduction is performed if the dataset is too large. These steps are not always mandatory, and only the necessary ones are selected based on the analysis requirements.

1.3.2 Data mining techniques

Data mining techniques can be categorized into three main types: descriptive, predictive, and optimization [5] (Figure 1.3).

1. **Descriptive techniques (unsupervised)** uncover hidden structures within data using methods such as clustering and frequent patterns mining.

2. **Predictive techniques (supervised)** aim to forecast outcomes based on existing data through classification and regression.

3. **Optimization techniques** enhance model performance by refining parameters, utilizing approaches like genetic algorithms and metaheuristics.

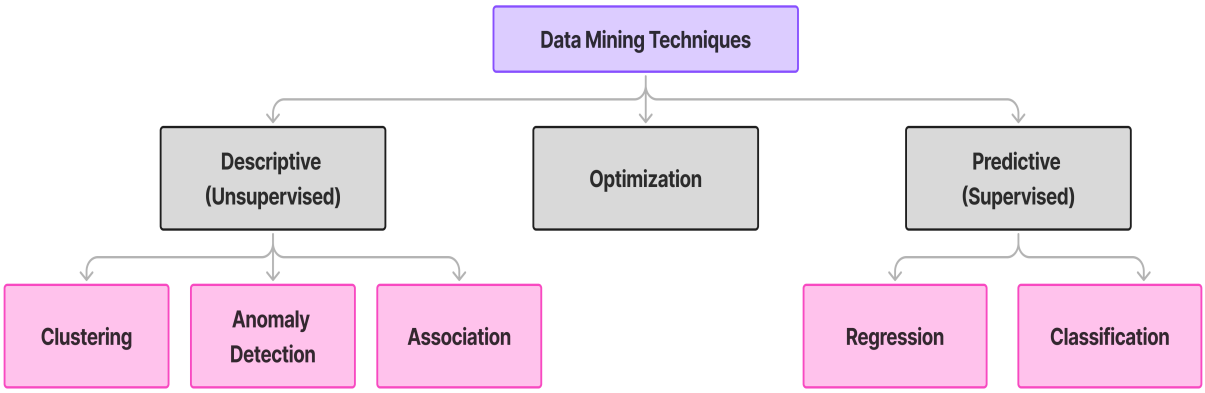

Figure 1.3: Main Data Mining Taskes

1.4 Clustering

Clustering is a data mining technique that aims to group similar data points into clusters (Figure 1.4), identifying patterns and relationships without prior knowledge of the data structure or classification. This unsupervised learning method helps uncover hidden insights within datasets. And is widely applied in various fields, including marketing segmentation, image processing, and anomaly detection, making it a valuable tool for data analysis and pattern recognition.

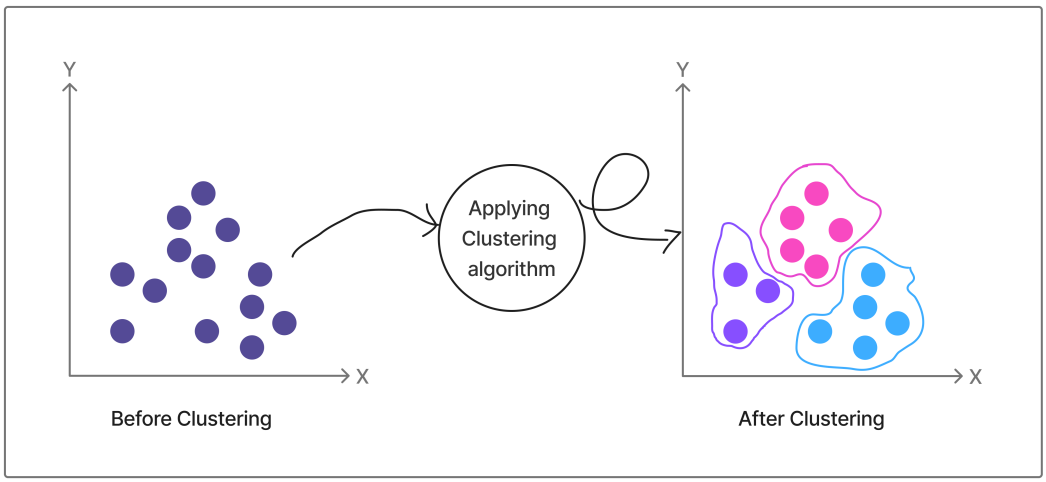

Figure 1.4: Illustration of Clustering Algorithm Process

To be effective, clustering algorithms must meet several requirements [3]:

- **Ability to handle diverse data types**, including numerical, categorical, textual, and image data.
- **Scalability** to process large datasets without compromising accuracy.
- **Detection of clusters with various shapes**, beyond Euclidean or Manhattan-based distances.
- **Flexibility in parameter selection** or assistance for users in defining parameters.
- **Robustness to noisy data**, handling missing or incorrect values effectively.
- **Support for incremental updates** without requiring full re-clustering.
- **Insensitivity to data input order** to ensure consistency.
- **Effective handling of high-dimensional data**, with subspace clustering as a viable approach.

A plethore of clustering algorithms exist and often share overlapping features making their classification difficult. Nonetheless, they can be broadly categorized based on their main characteristics to provide a clearer understanding of their underlying approaches.
Figure 1.5 exhibits this categorization [3] .

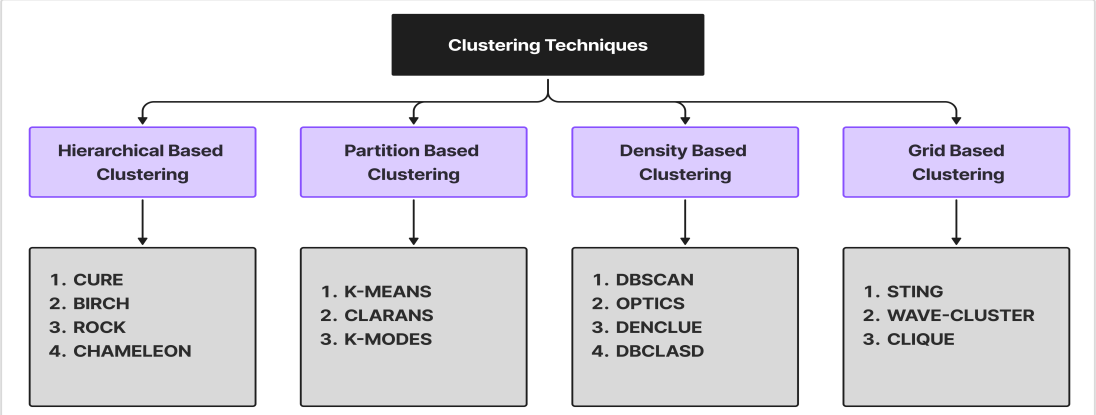

Figure 1.5: Types of Clustering Techniques

1.4.1 Partitioning Methods

Based on distance or similarity between data points, partitioning methods aim to divide a dataset into a predefined number of clusters,denoted as k. Each data point is assigned to one cluster, ensuring that all groups contain at least one object. Data objects within the same cluster should be as similar as possible, while objects in different clusters should be as dissimilar or distant as possible.

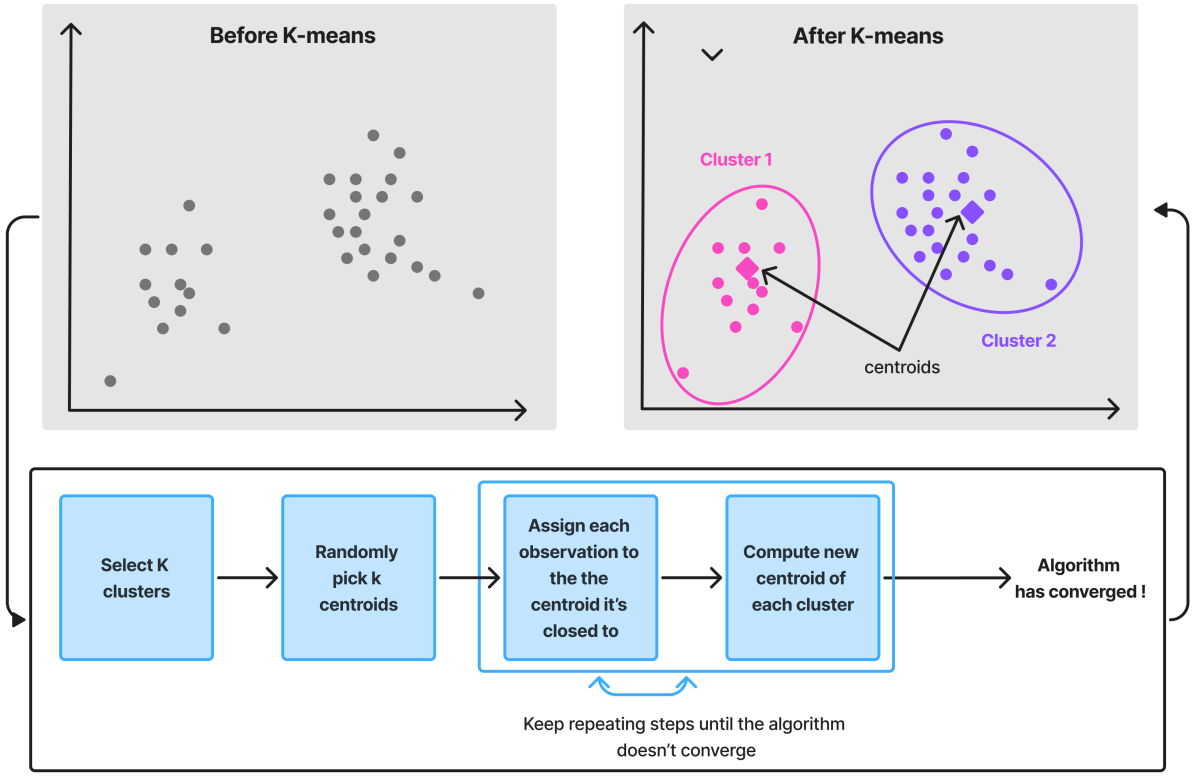

Figure 1.6: The Process of K-means Algorithm

One of the mostly studied and used partitioning algorithm is the k-means. It starts, after selecting the number of clusters (k), by randomly initializing k centroids. Then, each data point is assigned to the nearest centroid, and new centroids are computed as the mean of the assigned points. This iterative process continues until the centroids no longer change significantly or a predefined number of iterations is reached.

Figure 1.6 illustrates this entire process, from initial random centroids to the final stable clusters.

1.4.2 Density-Based Methods

Contrarily to partitioning clustering which are distance-based methods, density-based methods rely on the concept of data density while identifying clusters.
In such methods, density refers to the number of data points within a given region of space. A region is said to be dense if its number of data points exceeds a certain threshold.
These methods identify clusters by grouping data points that are densely packed together while filtering out noise. A cluster continues to grow as long as the number of points in its neighborhood exceeds a specified density threshold.

Unlike partitioning methods, these approaches can detect clusters of arbitrary shapes, making them suitable for complex real-world data.
Density-Based Spatial Clustering of Applications with Noise -DBSCAN- is a well-known density-based algorithm. It groups data points based on density, making it suitable for datasets with irregular cluster shapes.

This figure (1.7) illustrates the DBSCAN clustering process:

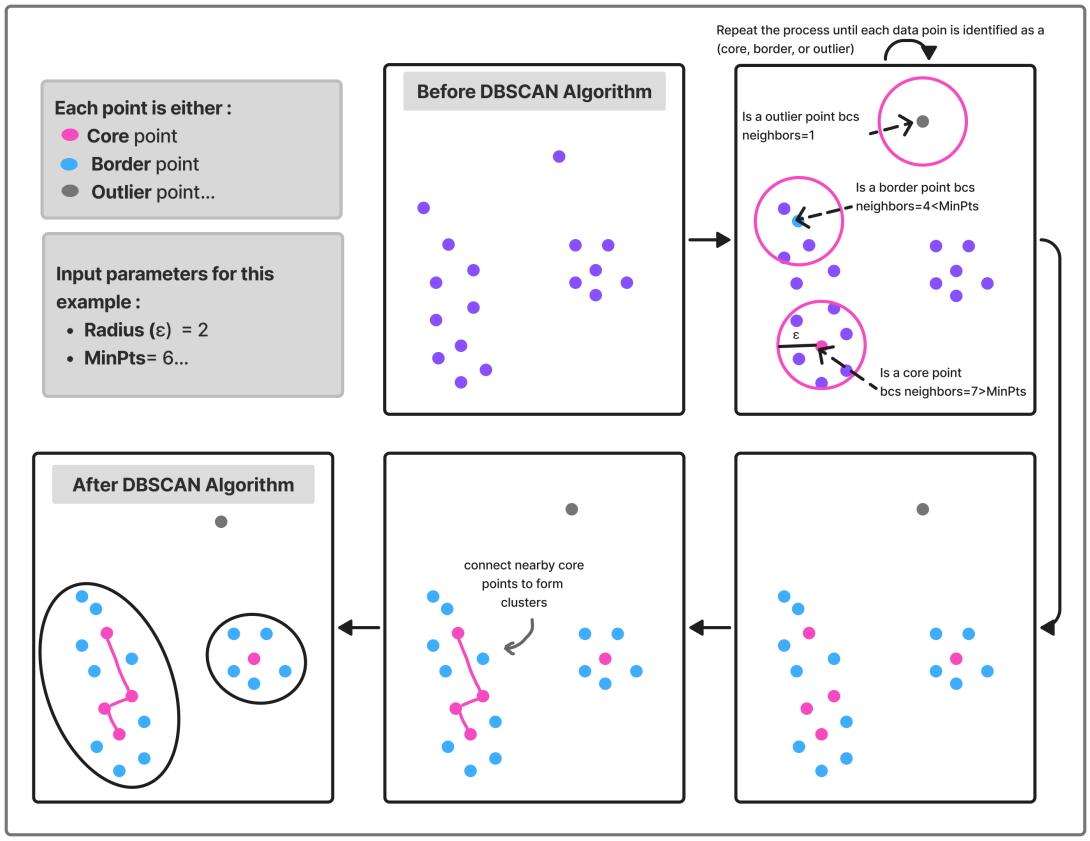

Figure 1.7: Exemple of DBSCAN Algorithm Process

The algorithm requires two key parameters:

- ϵ **(radius)**: Defines the neighborhood of a point.
- **MinPts**: The minimum number of points required to form a dense region.

Based on these parameters, DBSCAN classifies points into three categories:

- **Core points** : Have at least MinPts neighbors within ϵ.
- **Border points** : Are within ϵ of a core point but do not have enough neighbors to be core points.
- **Outliers**: Do not belong to any cluster due to insufficient density.

The figure shows the process step by step. First, all points are unclassified. The algorithm then examines each point, determining whether it is a core, border, or outlier. Once all points are classified, DBSCAN connects nearby core points to form clusters, while border points remain at the edges, and outliers are left ungrouped. The final result highlights how DBSCAN effectively identifies clusters of arbitrary shapes while detecting noise points.

1.4.3 Hierarchical Methods

Hierarchical methods build a hierarchical decomposition of data points, either through an agglomerative (bottom-up) or divisive (top-down) approach.
Agglomerative clustering starts with individual points as separate clusters and iteratively merges them. Contrarily, divisive clustering begins with a single cluster and progressively splits it.
Unlike partitioning methods, hierarchical clustering does not require specifying k in advance. However, once a merge or split occurs, it cannot be undone, which may lead to suboptimal clustering. Despite this limitation, hierarchical methods are useful for capturing nested structures in data and can be extended to subspace clustering.

1.4.4 Grid-Based Methods

Grid-based methods divide the data space into a finite grid structure, where clustering operations are performed on individual cells rather than directly on data points. This significantly reduces computational complexity, making it highly efficient for large datasets. Dense cells, containing a sufficient number of data points, are identified and combined to form clusters. The processing time of grid-based methods depends on the number of grid cells rather than the number of data points, making them ideal for high-dimensional data and spatial clustering tasks. Additionally, these methods can be integrated with hierarchical or density-based techniques for improved performance.

1.5 Classification

Classification is a supervised learning approach and is widely used for predicting categorical outcomes. It involves the assignment of input data to predefined categories based on learned patterns.

This technique is extensively applied in various fields, including healthcare, finance, and marketing, where accurate predictions are crucial [6].

The classification process consists of two main phases: training and testing (Figure 1.8). In the training phase, the algorithm learns patterns from a labeled dataset, where both the predictor attributes and their corresponding class labels are known. This step helps the model identify relationships between features and class outcomes.

Once trained, the model undergoes the testing phase, where it is evaluated on unseen data to assess its accuracy and generalizability. The algorithm predicts class labels without prior knowledge of their actual values, ensuring an objective evaluation.

The performance of a classification model is typically measured using metrics such as accuracy, precision, recall, and F1-score [6, 7].

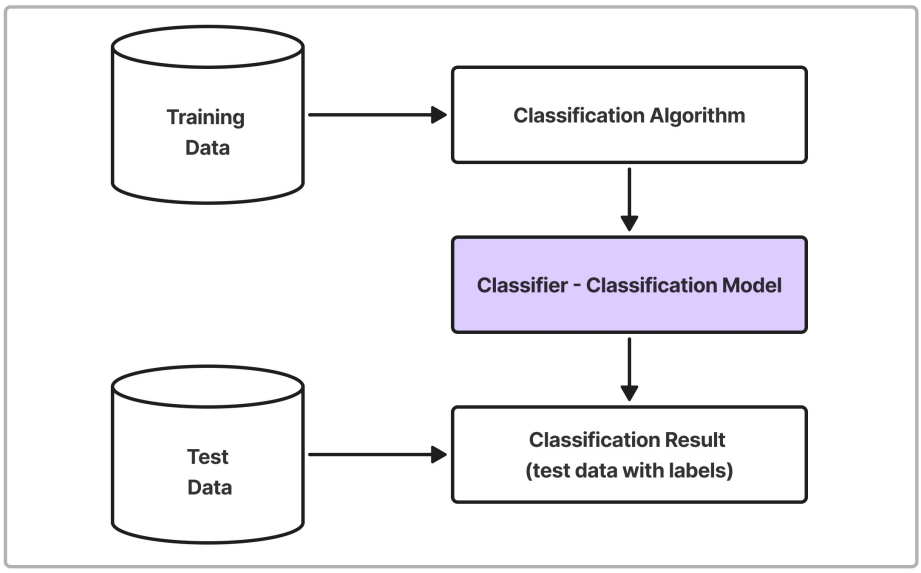

Figure 1.8: Classification Process

Classification has two primary categories [6, 7]: **eager learners and lazy learners**.

1.5.1 Eager Learners

Eager learners construct a model during the training phase and then use it to classify new instances. These algorithms depend not only on the data, but also on parameters such as weights, rules, support vectors, and probabilities, to make predictions.
As a result, eager learners require high computational effort during training but enable fast classification since predictions are made using a pre-built model.
Some well-known eager learners include Decision Trees, Naïve Bayes, and Support Vector Machines (SVM).

For example, figure (1.9) illustrates a Decision Tree used for classification, where data points are split based on feature values to assign them to different categories. Decision Trees aim to create a hierarchical structure of decisions that efficiently classify data by following simple rules at each node.

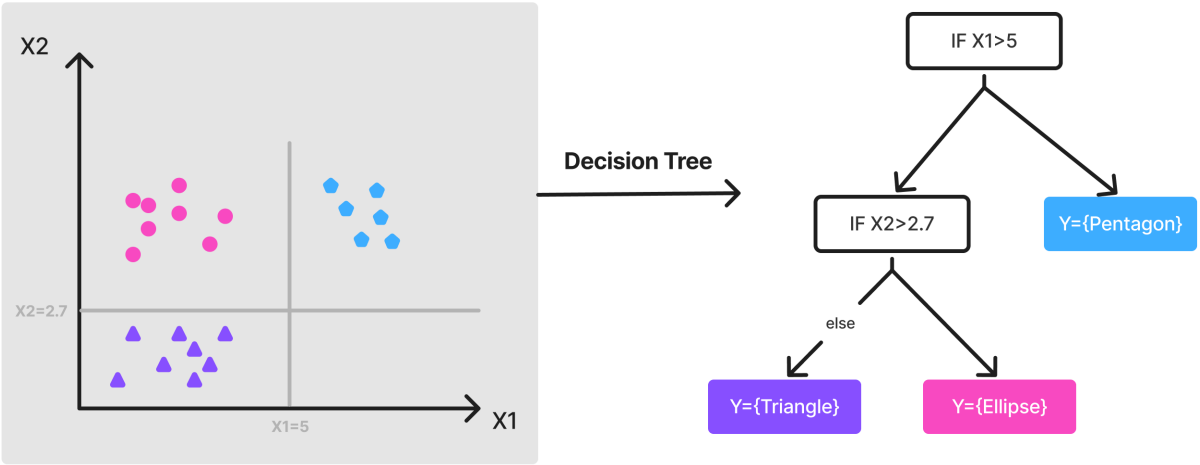

Figure 1.9: Example of Decision Trees Process

1.5.2 Lazy Learners

Lazy learners do not construct an explicit model during training. Instead, they store training instances and classify new data based on similarity measures.
Unlike eager learners, lazy learners depend only on data and do not rely on parameters such as weights or probability distributions. Because they do not build a model beforehand, they have low training time but high computational cost during prediction, as they must compare new instances against all stored data. Common examples include k-Nearest Neighbors (KNN) and Case-Based Reasoning (CBR).

For example, figure (1.10) illustrates the KNN classification process. A new sample needs to be classified based on its K = 4 nearest neighbors. Among them, three belong to Category A and one to Category B . Since the majority class is Category A, the new sample is assigned to this category.

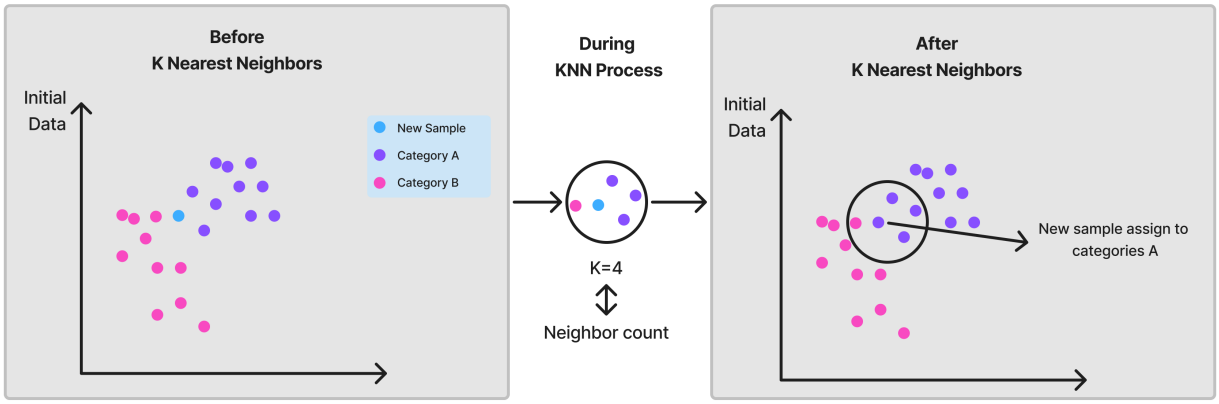

Figure 1.10: Example of K-Nearest Neighbors Process

> **Note:** The choice of K influences the classification, balancing sensitivity to noise and decision boundary smoothness.

1.6 Optimization

Optimization in data mining plays a crucial role in enhancing the efficiency and accuracy of analytical models by maximizing useful factors and minimizing errors. It enables effective processing of large datasets, reducing computational costs and increasing the speed of pattern extraction. Additionally, it is essential in data preprocessing, where it helps clean noisy data, remove outliers, and improve overall data quality before analysis. By optimizing data mining processes, researchers can achieve more accurate and reliable insights from complex datasets [5].

One of the most widely used optimization techniques in data mining is genetic algorithms (GA), which are inspired by natural selection to find the best solutions for complex problems [8]. These algorithms start with a randomly generated set of possible solutions, then iteratively refine them through selection, crossover, and mutation, ensuring that only the most optimal solutions survive each generation [8]. Compared to traditional methods, genetic algorithms are more effective at handling large and complex datasets, as they do not rely on gradient information and can efficiently explore multiple solutions in parallel [5]. They are particularly useful in feature selection, clustering, and rule generation , where they help improve model accuracy and efficiency by optimizing data representation and decision-making processes [5, 8, 9].

1.7 Post-Preprocessing Data

Post-processing is a key step in the Knowledge Discovery in Databases (KDD) process. It aims to refine, interpret, and integrate extracted knowledge, making it more useful for practical applications [10].

- One of the main goals of post-processing is to filter and simplify models by removing unnecessary or overly specific rules. Techniques such as decision tree pruning and rule set reduction help improve model generalization while reducing complexity [11].
- Post-processing also includes model evaluation, which involves measuring accuracy, stability, and performance on new data. This step is essential to ensure the reliability and applicability of the extracted knowledge [10].
- Another important aspect of post-processing is knowledge interpretation and visualization. Converting models into understandable formats, documenting results, and using visualization tools help make insights accessible to end users [11].
- Finally, post-processing enables knowledge integration by combining multiple models from different methods. This approach enhances the robustness of decision-support systems and improves the relevance of results [10].

Although often overlooked, post-processing is a crucial step in data mining. It ensures that extracted knowledge is not only accurate but also comprehensible and applicable in real-world scenarios [11].

1.8 Conclusion

In this chapter, we explored the fundamental aspects of data mining, from understanding different data types to preprocessing techniques that enhance data quality. We examined the knowledge discovery process and highlighted essential data mining techniques, such as clustering, classification. These methods, driven by statistical and machine learning principles, enable the extraction of meaningful patterns from large datasets. Finally, we discussed the importance of post-processing, where the extracted knowledge is refined, evaluated, and presented in a meaningful way to support informed decision-making.

CHAPTER 2

METAHEURISTICS AND ELEPHANT HERDING OPTIMIZATION ALGORITHM

2.1 Introduction

Metaheuristic algorithms have gained increasing attention as powerful tools for solving complex and large-scale optimization problems that are difficult to address using exact methods. Their ability to provide near-optimal solutions within reasonable computational times has made them suitable for a wide range of applications, particularly in combinatorial and continuous optimization. Among these techniques, nature-inspired algorithms stand out due to their adaptive behavior and robust search capabilities, leading to the development of several novel strategies, including the Elephant Herding Optimization algorithm.
In this chapter, metaheuristics are introduced as a powerful class of optimization methods, followed by a detailed focus on the EHO algorithm, its inspiration, and fundamental mechanisms.Several enhanced versions of EHO proposed in the literature are then examined to understand how they tackle existing limitations and boost the algorithm's performance.

2.2 Problem Solving

Problem solving refers to the process of identifying, analyzing, and designing effective approaches to find solutions to complex tasks or problems. It often requires algorithms that operate efficiently under a set of given constraints. In this context, NP-complete represents a major challenge in problem solving.
To solve such problems, several methods are used, including exact methods that aim to find the solution to the problem by exploring the whole search space. These methods guarantee to find the solution to the problem or prove that the problem cannot have a solution. This whole exploration of the search space is computationally expensive, often leading to excessive processing time and a combinatorial explosion in the number of possible solutions especially for large-scale or highly constrained problems, making them impractical for many problems.
In such case, approximate methods offer a viable alternative. Heuristics are problem-specific strategies that leverage domain knowledge to find good-quality solutions efficiently, though without guaranteeing optimality. These methods try to find a compromise between the quality of the solution and consuming time.They are particularly useful for large or combinatorial problems where speed and simplicity are essential [12].

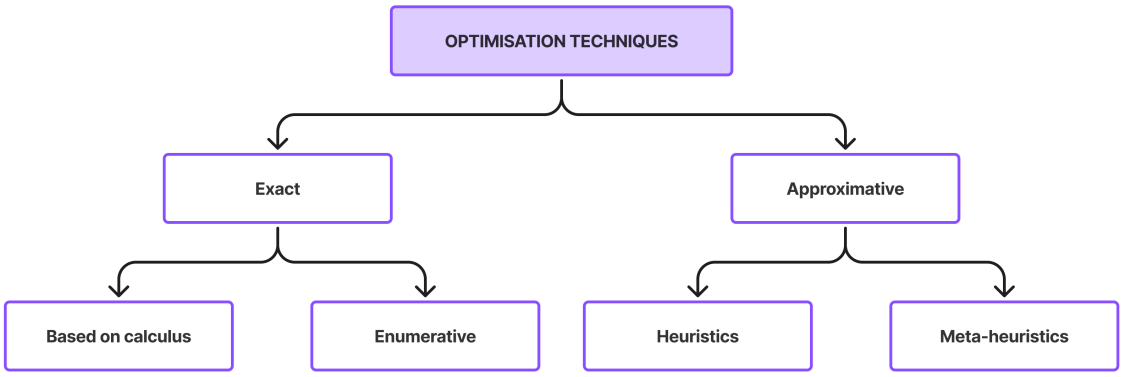

Figure 2.1: Optimisation techniques

The Figure 2.1 illustrates the main categories of optimization techniques specifically within combinatorial discrete optimization, divided into two branches "exact" and "approximative" methods.

2.3 Metaheuristics

Metaheuristics are high-level optimization strategies designed to guide the search for optimal or near-optimal solutions, especially in complex and large-scale problems. Unlike traditional heuristics, which are typically problem-specific, metaheuristics offer a more abstract and general-purpose framework. This adaptability allows them to be applied to a wide range of optimization problems with minimal customization, earning them the label of generalist heuristics [12].

These algorithms are often inspired by natural phenomena such as genetic evolution, swarm behavior, or physical processes such as annealing (Figure 2.2). They are particularly effective in navigating large and complex search spaces. One of their main advantages is that they do not require prior knowledge of a good initial solution, making them suitable for problems that are difficult to model or where the search space is poorly understood. Their flexibility and exploratory capabilities help avoid the common issue of getting stuck in local optima, which often hampers traditional optimization methods [13].

Despite their strengths, metaheuristics have some limitations. Most are based on stochastic processes, which means that there is no guarantee of finding the global optimum. Additionally, their effectiveness often depends on careful tuning of parameters, and the computational cost can be significant especially for very large problems [13].

Nevertheless, metaheuristic algorithms remain powerful tools in modern optimization. They strike a balance between exploration and exploitation, and their ability to generalize across diverse problems makes them indispensable in practice provided their limitations are properly understood and managed [13].

2.3.1 Metaheuristic Algorithms Classifications

Metaheuristics can be classified within have different classes, each defined by distinct characteristics and behaviors. However, before exploring these classes, it is important to understand the

criteria used to classify metaheuristic algorithms. These criteria provide a structured way to group algorithms based on their inspiration, search strategies, memory usage, and other relevant aspects. This classification criteria is represented in the following key dimensions [13]:

- **Nature-inspired vs. Non-nature-inspired:** Metaheuristics can be classified based on whether they are inspired by natural processes or not. Nature-inspired metaheuristics include algorithms such as Genetic Algorithms (GA), which are inspired by the process of natural selection. Non-nature-inspired metaheuristics include algorithms such as Simulated Annealing, inspired by the process of metal cooling.
- **Deterministic vs. Stochastic:** Metaheuristics can be classified based on whether they use deterministic or stochastic processes. Deterministic metaheuristics use predefined processes to generate new solutions, ensuring that the same input always produce the same result. Examples include Hill Climbing and Deterministic Annealing, where the search follows a fixed pattern. Stochastic metaheuristics, on the other hand, introduce randomness into the solution search. Examples include Genetic Algorithms, where crossover and mutation operators are random.
- **Trajectory-based vs. Population-based:** Metaheuristics can also be classified based on whether they focus on finding a single solution or multiple solutions. Trajectory-based metaheuristics focus on refining a single solution over time through iterative improvement. Examples include Hill Climbing and Simulated Annealing, which both start with an initial solution and attempt to improve it iteratively. Population-based metaheuristics, such as Genetic Algorithms (GA), Particle Swarm Optimization (PSO), and Ant Colony Optimization (ACO), operate on a set of candidate solutions (a population) and explore the search space more broadly by evolving or updating the entire population simultaneously.
- **Local search-based vs. Global search-based:** Metaheuristics can be classified based on whether they focus on exploring the local search space or the global search space. Local search-based metaheuristics focus on improving the current solution by exploring its neighborhood. Hill Climbing and Tabu Search are examples of local search methods, where the algorithm looks for better solutions in the immediate vicinity of the current solution. Global search-based metaheuristics focus on exploring the broader search space. Genetic Algorithms, Ant Colony Optimization, and Particle Swarm Optimization are examples, as they explore the entire search space by employing strategies such as crossover, pheromone trails, or particle movement, respectively.

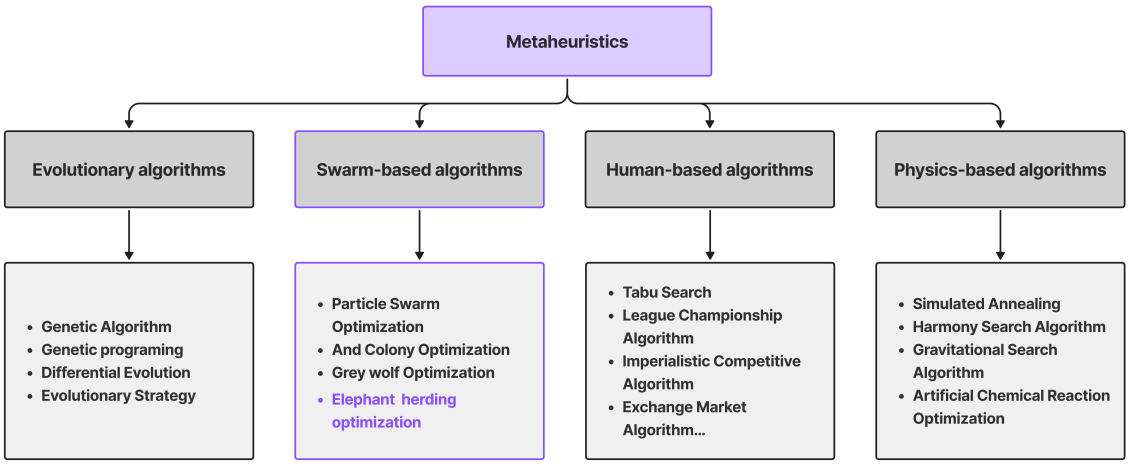

Figure 2.2: Metaheuristics Classification

The existence of multiple classifications arises because each classification considers one or more criteria to categorize metaheuristic algorithms. One of the existing classifications shows that metaheuristics can be nature-inspired or non-nature-inspired. As illustrated in the following figure 2.2, nature-inspired metaheuristic algorithms can be grouped into several categories, one of which is the Swarm-Based Algorithms [14].

2.3.1.1 Swarm-Based Algorithms

Swarm intelligence methods are one of the most well-known paradigms in metaheuristic methods which have been widely used in various applications. The inspiration of the swarm intelligence algorithms, originates from the collective behaviour of animals [15].

Swarm intelligence algorithms are typically based on the following principles [13]:

- ✓ **Decentralization:** There is no centralized control over the system. Instead, each agent follows simple rules that are based on local information and interaction with its neighbors.
- ✓ **Self-organization:** The agents interact with each other and with their environment to form a pattern or structure that emerges from the collective behavior of the system.
- ✓ **Adaptation:** The system is capable of adapting to changes in its environment, either through the individual behavior of the agents or through the emergence of new collective patterns.

Swarm intelligence algorithms, drawing inspiration from various natural phenomena, have emerged as powerful tools for solving complex optimization problems across diverse fields such as engineering, finance, biology, and computer science.

Two of the most widely used swarm intelligence algorithms are Particle Swarm Optimization (PSO) and Ant Colony Optimization (ACO). While the idea of PSO originated from the social behavior of bird flocking while searching for food. ACO is inspired by ants, which are well capable of keeping the past paths in mind by pheromone. Inspired by this phenomenon, the ACO algorithm was proposed by Dorigo et al [15].

Another swarm-based algorithm, Elephant Herding Optimization (EHO), was inspired by the social behavior of elephants, where the herd's movement is influenced by the collective decision-making process to search for food and adapt to environmental conditions. This behavior has been translated into an optimization algorithm for solving complex problems.

2.4 Elephent Herding Optimisation Algorithm

Elephant Herding Optimization (EHO) is a nature-inspired metaheuristic algorithm that mimics the herding behavior of elephant groups. This method draws from the social structure and movement patterns of elephant clans, using these biological principles to guide a population-based search for optimal solutions in complex problem spaces.

In this section, the core components of the EHO algorithm are detailed, including its biological inspiration, the roles of clans and matriarchs, and the two main operators that drive the search process clan updating and separation. A step-by-step explanation of the algorithm's workflow is also provided to clarify its practical implementation in optimization contexts.

2.4.1 Elephant Herding Optimization Research Studies

Nature-inspired metaheuristics have attracted significant attention in recent years due to their ability to deliver high-quality solutions for complex optimization problems. Their robustness, flexibility, and efficiency in balancing exploration and exploitation have made them a focal point in optimization research. In this context, the Elephant Herding Optimization (EHO) algorithm was introduced in 2015 by Gai-Ge Wang et al [16]. This metaheuristic, inspired by the social behavior of elephant herds, simulates clan-based movement through two key mechanisms: clan updating and separating operations.

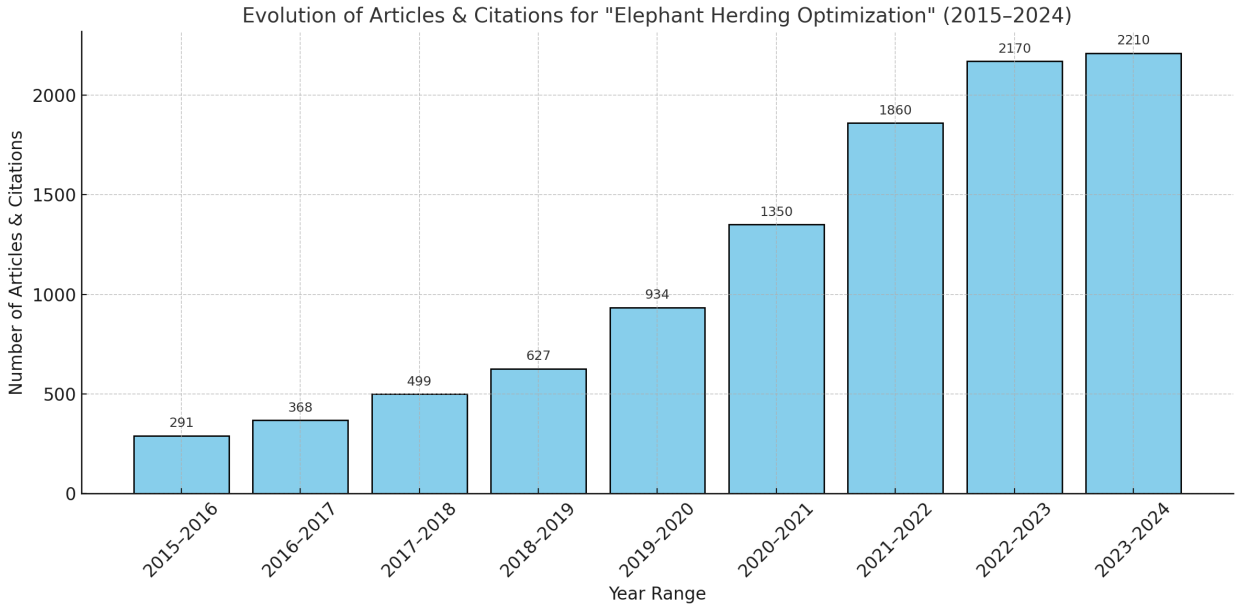

Figure 2.3: Evolution of Articles & Citations for 'Elephant Herding Optimization' (2015–2024)

The evolution of publications and citations related to EHO between 2015 and 2024 clearly illustrates this growing interest. As shown in the figure (2.3), the number of research papers has increased steadily: from 291 publications in 2015–2016, to 499 in 2017–2018, then reaching 934 in 2019–2020, and peaking at 2210 publications in 2023–2024. This exponential growth reflects not only the academic acceptance of the algorithm but also its application across various fields and problem domains.

Through an extensive review and exploration of the existing research articles on EHO, two main directions are identified in which the majority of studies have evolved. The first involves algorithmic enhancement, where researchers have aimed to improve the original algorithm by modifying update equations, integrating advanced solution generation strategies, or dynamically adjusting parameters to improve convergence and solution quality. The second direction focuses on hybridization, where a metaheuristics algorithm are combined with other metaheuristics, or machine learning techniques, data mining techniques. These hybrid approaches aim to capitalize on the strengths of each component method.

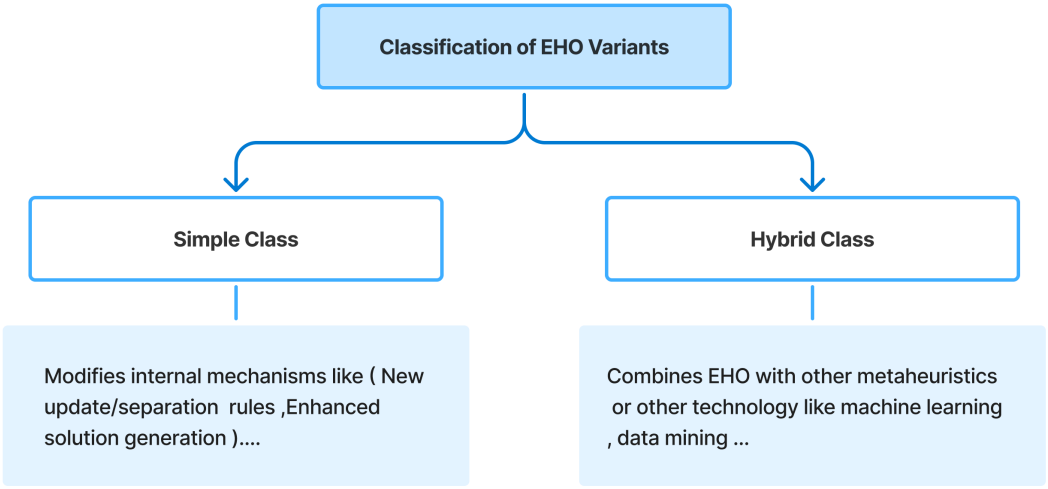

Figure 2.4: Classification of EHO Variants

Based on these two directions, the existing variants of EHO can be broadly categorized into two main classes: the Simple Class and the Hybrid Class, as illustrated in the figure (2.4). The Simple Class includes variants that make internal improvements to the core algorithm, while the Hybrid Class encompasses those that integrate EHO with other optimization or intelligent techniques to enhance performance and adaptability.

NB: All the statistics, classifications, and observations presented in this section were derived from a thorough investigation of research articles available on Google Scholar. This process involved the use of a variety of carefully formulated search queries to ensure comprehensive coverage of the literature on Elephant Herding Optimization (EHO). Among the queries used were: "Elephant Herding Optimization", "Elephant Herding Optimization Algorithm", "elephant herding optimization hybrid","improved elephant herding optimization" . These queries enabled us to retrieve a significant number of relevant publications from 2015 to May 2025, offering valuable insights into the development, categorization, and applications of the EHO algorithm.

2.4.2 Herding behavior of Elephants

Elephants are the largest terrestrial mammals. The African and Asian elephants are the two traditionally recognized species. They exhibit complex social structures, where female elephants serving as matriarchs lead clans composed of related females and their calves (Figure 2.5). Male elephants, on the other hand, prefer solitude and typically leave their family groups upon reaching maturity. Despite living independently, they maintain contact with their clan through low-frequency vocalizations (Figure 2.6). Elephants also possess a remarkable trunk, which serves multiple purposes such as breathing, grasping objects, and lifting water (Figure 2.7). This unique social behavior has inspired research in global optimization, where elephant herding is modeled through specific algorithmic operators [16][17].

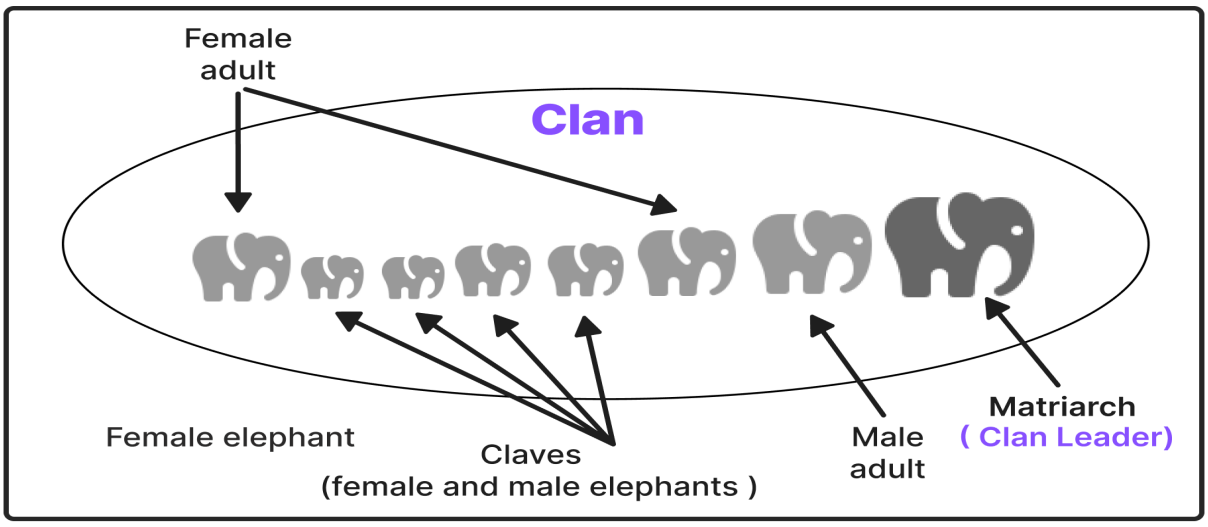

Figure 2.5: social structure of elephants herd

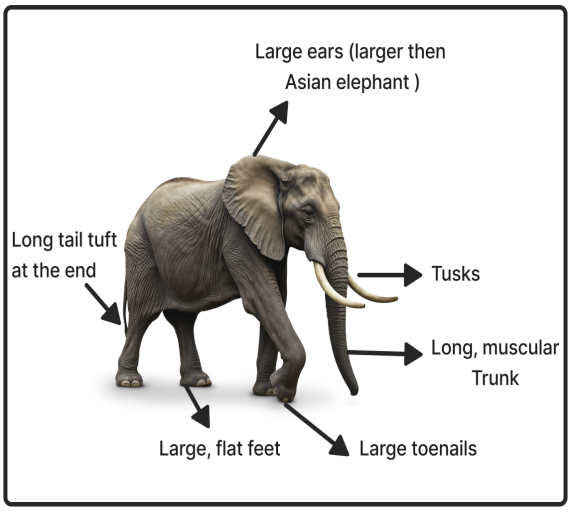

Figure 2.6: Structure of Elephant

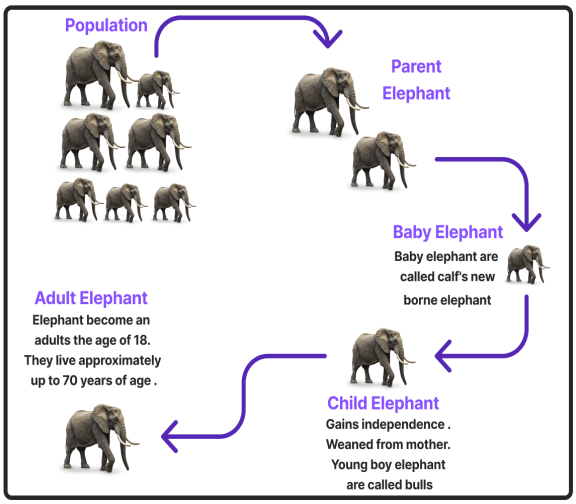

Figure 2.7: Life Cycle of EHO Algorithm

The Figure 2.6 illustrates the anatomical features of an elephant, highlighting its trunk, ears, tusks, feet, and tail, each serving distinct functions crucial for the elephant's survival and behavior.

The Figure 2.7 illustrates the life cycle of elephants, starting with reproduction and progressing through stages of baby, child, and adult elephants, with adults living up to around 70 years, contributing significantly to their herds and ecosystems.

2.4.3 Elephant Herding Optimization Algorithm

The Elephant Herding Optimization (EHO) algorithm stands as a groundbreaking swarm-based approach, tailored to address a wide array of optimization challenges. Inspired by the intricate dynamics observed within elephant herds, EHO mimics the coordinated behaviors of these majestic creatures.

Within the algorithm, elephants are organized into clans, each overseen by a matriarch, mirroring the hierarchical structure prevalent in nature. Notably, as male elephants mature, they embark on solitary journeys away from their family groups, a behavior integral to the algorithm's design. This division into clans and the departure of mature males give rise to two fundamental operators within EHO: the clan updating operator and the separating operator, each contributing to the algorithm's efficacy in navigating complex optimization landscapes[18].

To enable the application of elephant herding behavior to address diverse global optimization challenges, it has been distilled it into the following streamlined rules:

i) Elephant Population is composed of some Clans and each clan contains a fixed number of Elephants

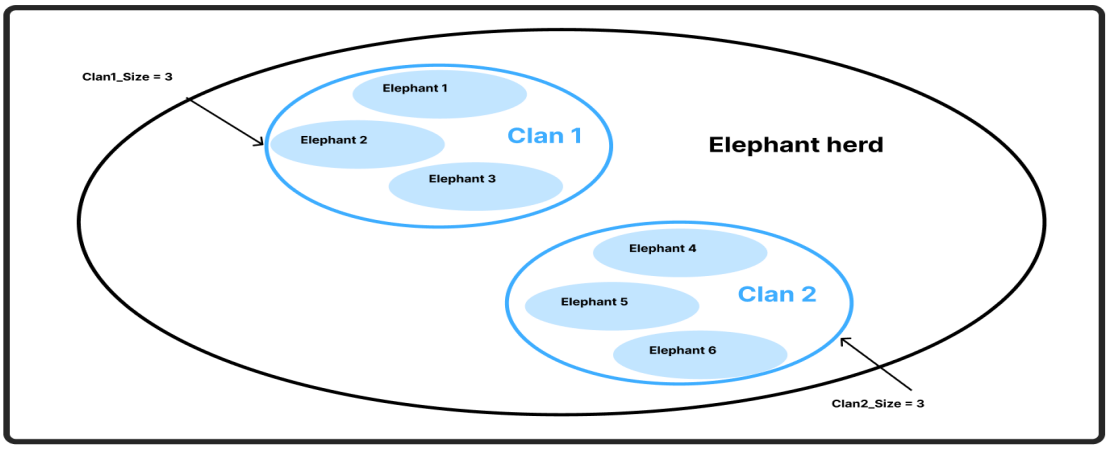

Figure 2.8: Elephant herd population

ii) The individual clan consists of a group of elephants that are under the command of a Matriarch.

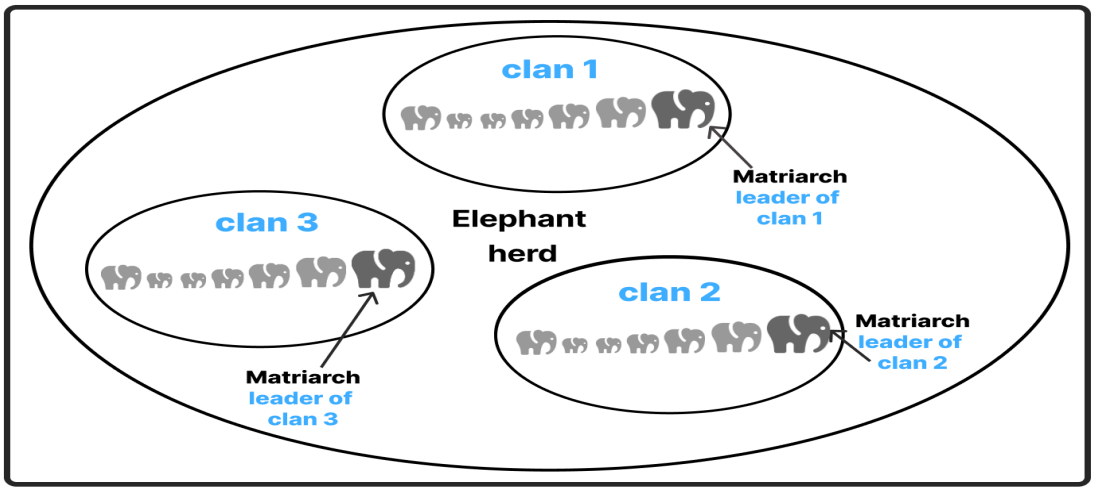

Figure 2.9: The leader of each clan in the herd

iii) A fixed number of male elephants will leave their family group and live solitarily far away from the main elephant group at the start of each generation.

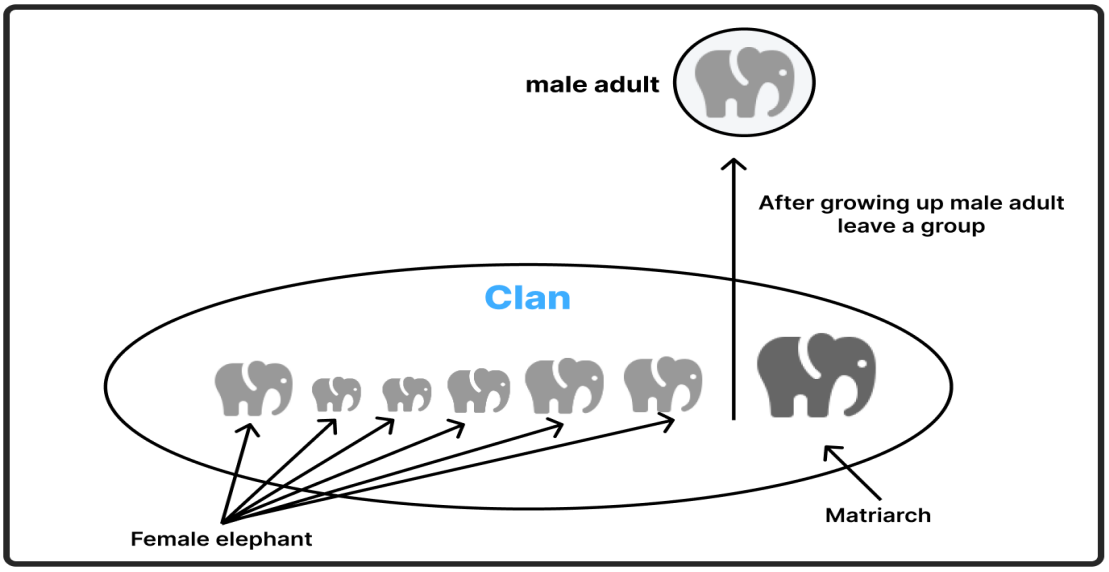

Figure 2.10: The male adult separating process

Building upon the previously outlined natural rules that govern elephant herding behavior, it becomes essential to translate these biologically inspired concepts into their algorithmic equivalents to enable practical implementation. The following table (2.1) serves this purpose by mapping each element of the natural system such as elephants, clans, and matriarchs to corresponding components in the optimization process. This conceptual bridge not only facilitatces a clearer understanding of the EHO algorithm's structure but also highlights how the dynamics observed in nature are harnessed to guide the search for optimal solutions in complex problem domains.

In nature	**In problem solving (algorithm)**
Elephant	Decision variable (solution)
Clan of elephants	Recommended solutions
Matriarch	Best solution
Male which leaves the clan	Worst solution removed by separation operator
Elephant position	How well the solution is? (update operator)

Table 2.1: Elephant Herding Behaviour VS Elephant Herding Optimization Algorithm

The EHO algorithm relies on two main operators along with an elitism strategy, to guide the population dynamics.

2.4.3.1 Clan updating operators

Elephants live in clans of equal size, where elephants live together under the leadership of a matriarch (Figure 2.9). It is presumed that each clan is made up of an equivalent, unchangeable number of elephants for modeling purposes (Figure 2.8).

The positions of elephants within a clan are updated according to on their relationship with the matriarch. By analogy Clan Updating operator is used for this behaviour in EHO algorithm [17, 16] .

The next position of an elephant in clan ci, is influenced by the matriarch of the same clan. The position of the elephant jin clan ci, is updated as follows:

$$X_{\text{new},ci,j} = X_{ci,j} + \alpha \cdot (X_{\text{best},ci} - X_{ci,j}) \cdot r \tag{2.1}$$

where ;

- $x_{\text{new},ci,j}$ and $x_{\text{ci},j}$ are newly updated and old position for elephant j in clan ci, respectively.
- $\alpha \in [0, 1]$: is a scale factor that determines the influence of matriarch ci on $x_{\text{ci},j}$.
- $X_{\text{best},ci,}$ represents matriarch ci, which is the fittest elephant individual in clan ci.
- $r \in [0, 1]$: a random value generated from a uniform distribution, introducing randomness into the position update to maintain exploration ability.

As for the rest of elephants, the position of the matriarch (the fittest elephant within a clan) is updated. Its update is influenced by the statistical central elephant calculated as the average position of the clan members as follows:

$$X_{\text{new},ci,j} = \beta \cdot X_{\text{center},ci} \tag{2.2}$$

where ;

- $\beta \in [0, 1]$ is a factor that determines the influence of $x_{\text{center},ci}$ on $x_{\text{new},ci,j}$.
- $x_{\text{center},ci}$, the center of clan ci, which is calculated for the d-th dimension using this equation (2.3).

$$X_{\text{center},ci,d} = \frac{1}{n_{ci}} \sum_{j=1}^{n_{ci}} X_{ci,j,d} \tag{2.3}$$

where ;

- $1 \leq d \leq D$ indicates the d-th dimension, and D is its total dimension.
- n_{ci} is the number of elephants in clan ci.
- $x_{ci,j,d}$ is the d-th dimension of the elephant individual $x_{ci,j}$.

Based on the description above, the clan updating operator can be formulated as shown in Algorithm 1 [15].

Algorithm 1: Clan updating operator

```
Begin;
for ci = 1 to total number of clans in elephant population do
    for j = 1 to n_ci (number of elephants in clan ci) do
        Update x_ci,j and generate x_new,ci,j according to equation (2.1);
        if x_ci,j = x_best,ci then
            Update x_ci,j and generate x_new,ci,j according to equation (2.2);
End;
```

2.4.3.2 Separating operator

Within an elephant group, male elephants leave their family group and live alone when they reach puberty Figure 2.10. By analogy, this separating process can be modeled into separating operator when solving optimization problems [16].
In order to further improve the search ability of EHO method, let's assume that the elephant individuals with the worst fitness are considered for separating operator at each generation. Equation (2.4) exhibits this separating operator.

$$X_{\text{worst},ci} = X_{\min} + (X_{\max} - X_{\min} + 1) \cdot \text{rand} \tag{2.4}$$

where ;

- $X_{\max}$ and $X_{\min}$ are respectively the upper and lower bounds of the position of the elephant individual.
- $X_{\text{worst},ci}$ is the worst elephant individual in clan ci.
- rand $\in [0, 1]$ is a kind of stochastic distribution and uniform distribution in the range $[0, 1]$.

Accordingly, the separating operator can be formed as shown in Algorithm 2 [15].

Algorithm 2: Separating Operator

```
Begin;
for ci = 1 to total number of clans in elephant population do
    Replace the worst elephant individual in clan ci by equation (2.4);
End;
```

2.4.3.3 Elitism Strategy

The elitism strategy is a key enhancement in metaheuristic algorithms, introduced to preserve the best individuals during the evolutionary process and prevent the degradation of solution quality. In the context of the EHO algorithm, this concept was first integrated by Gai-Ge Wang in 2016 [15]. Wang proposed saving the best elephant individuals at the beginning of the process and reintroducing them at the end of the clan updating and separating strategy, ensuring that the final population retains high-quality solutions.

The elitism strategy was later adopted into the basic version of the EHO algorithm [19], where a specific number of top-performing elephants are preserved at each generation and used to replace the worst-performing ones. This integration not only safeguards the best solutions but also helps maintain or improve the overall quality of the population, reinforcing the algorithm's ability to converge toward optimal solutions.

2.4.4 Procedure and Pseudocode of the EHO Algorithm

The Elephant Herding Optimization algorithm operates through a structured sequence of steps aimed at iteratively improving a population of solutions.

This process can be divided into three main steps:

Step 1: Initialization

This step prepares the algorithm's components and population:

- **Parameter setup:** Initialize all necessary parameters of the algorithm.
- **Random population generation:** Generate an initial population of N elephants with random positions within the search space.
- **Fitness evaluation:** Compute the fitness of each individual.
- **Sorting:** Sort the elephants based on their fitness to identify the best individuals.
- **Clan division:** Partition the population into c clans of S solutions.

Step 2: Process Execution

This step forms the core iterative procedure of EHO, repeated until finding the optimal solution or reaching the maximum generation :

- **Clan update operator:** The elephants of each clan except the matriarch update their positions under the influence of the clan matriarch, while the matriarch updates her position by moving toward the center of the clan (Algorithm 1).
- **Separation operator:** The worst elephant of each clan is replaced to promote diversity (Algorithm 2).
- **Elitism strategy:** The top m elephants saved at initialization replace the m worst individuals. This ensures the preservation of high-quality solutions.

Step 3: Output

Once the stopping condition is reached (the optimal solution is found or the algorithm has reached the maximum number of generations):

- The best solution x_{best}, identified across all generations, is returned as the final output of the algorithm.

All these steps are summarized in the pseudocode presented below [20], which captures the overall structure of the EHO algorithm.

Algorithm 3: Elephant Herding Optimization (EHO)

Input: Maximum generation t_{max}; Population size N; Number of clans c;
Output: Best solution x_{best}

Initialize population and parameters α, β, r;
Calculate and sort fitness of all individuals;
Save the best m elephants (elitism);
Divide the population into c clans;
while $t < t_{\text{max}}$ **do**
    for $c_i = 1$ *to* c **do**
        Apply **Clan Update Operator** to clan c_i ; `// See Algorithm 1`
    for $c_i = 1$ *to* c **do**
        Apply **Separation Operator** on worst elephant in c_i ; `// See Algorithm 2`
    Recalculate and update fitness values;
    Sort entire population;
    Replace worst m individuals with saved best m (elitism);
    $t = t + 1$;
return x_{best}

2.5 Existing Variants of the EHO Algorithm

Since the introduction of the Elephant Herding Optimization algorithm in 2015 [16], an increasing number of studies have explored its development and improvement. These studies can be grouped into two primary categories (2.4.1): those that focus on algorithmic enhancements and those that pursue hybridization with other optimization techniques or intelligent systems. Based on this classification [21], this section synthesizes the principal EHO variants proposed in recent years, highlighting the specific mechanisms they modify or the techniques with which EHO has been hybridized.

2.5.1 Simple Variants of the EHO Algorithm

The simple variants comprises approaches that modify specific components within the original EHO process without incorporating external algorithms or techniques. These methods aim to improve performance by refining internal operators such as clan updating, separation, or position adjustment. By directly altering the structural behavior of the algorithm, these enhancements address known limitations such as premature convergence, stagnation, or lack of exploitation efficiency.
Table 2.2 presents a selection of existing EHO variants classified under the simple category.

Variant Title	**Brief Description**	**Targeted Limitation**	**Improved Component**	**Ref.**
Updated Clan Leader Position	Update each leader based on the average position of other clan leaders.	Clans become isolated and search in small local areas.	Update operator	[22]
Use Cauchy Mutation	Separated individuals undergo Cauchy mutation to explore farther regions of the solution space.	Algorithm stays focused in one area, limiting exploration.	Separator operator	[22]
Alpha-Tuning	Dynamically compute α using a decreasing function from $\alpha_{\max}$ to $\alpha_{\min}$.	Fixed α causes over-exploration or early convergence.	Alpha parameter	[23]
Separation Strategy	Replace worst elephant only if diversity improves or condition exceeds a threshold.	Blind replacement reduces solution quality and wastes evaluations.	Separator operator	[24]
EHO Adaptive (ADEHO)	Replaces random variables with adaptive probabilities that decrease over time.	Sharp exploration-to-exploitation shift causes instability.	Random variables	[25]

Table 2.2: Summary of Selected Simple EHO Variants

2.5.2 Hybrid Variants of the EHO Algorithm

Hybrid variants of the Elephant Herding Optimization algorithm aim to overcome its inherent limitations by integrating complementary techniques from other metaheuristics or data-driven methods. These approaches enhance EHO's search capabilities, convergence behavior, and robustness by modifying or extending specific components such as update mechanisms, clan division strategies, or solution replacement criteria.

The table 2.3 summarizes notable hybrid EHO variants, highlighting their motivations, structural changes, and benefits.

Hybridiza -tion Technique	Brief Description	Integrated Component	Problem Addressed	Benefit	Ref.
EHO +PSO	Particle Swarm Optimization (PSO) is a population-based metaheuristic inspired by social behaviors like bird flocking and fish schooling. It is widely used for its simplicity and effectiveness in global exploration. In this hybridization, the velocity-based update mechanism of PSO is incorporated into EHO to improve the update of the worst solutions, maintaining better connectivity within clans and preventing premature convergence.	Position Update Mechanism	EHO lacks global guidance in the exploitation phase and struggles with poor convergence and local optima entrapment.	Improved convergence speed, enhanced exploration, and better ability to escape local optima.	[26]
EHO + GA	A genetic algorithm is an inference algorithm inspired by the process of natural selection, operating through selection, genetic exchange, and mutation. In this approach, Integrates genetic crossover and mutation after the EHO clan updating phase.	Update operator	Standard EHO may lack global diversity and get trapped in local optima.	Helps escape local optima and improves diversity.	[27]
EHO + ABC	Artificial Bee Colony metaheuristic inspired by the foraging behavior of honey bee swarms. It integrates foraging behavior into EHO, using solution improvement checks and residence tracking to decide whether to keep or replace a solution.	After update operator	Premature convergence and no mechanism to track solution stagnation.	Adds a memory-based mechanism to detect and replace stagnant solutions.	[28]
EHO + K-Means	K-means clustering algorithm Replaces fixed-size clan division with dynamic clustering using K-Means, based on similarity of elephant positions.	Clan dividing	Standard EHO may group dissimilar solutions in the same clan, harming cooperation.	Produces more coherent clans of similar solutions, enhances local exploitation.	[29]

EHO + GWO	Grey Wolf Optimizer inspired by the social hierarchy and hunting behavior of grey wolves in nature, it is guides worst solutions toward the best ones using GWO's leader-following strategy.	Update operator	Premature convergence due to limited exploration.	Robust search behavior across diverse and complex landscapes.	[30]

Table 2.3: Summary of Selected Hybrid EHO Variants

2.6 Conclusion

Throughout this chapter, the foundational understanding of the EHO algorithm has been established within the broader context of metaheuristic optimization. By analyzing its original structure, behavior, and key mechanisms, along with reviewing existing improvements and hybridizations, the chapter has highlighted both the strengths and limitations of the method. This exploration underscores the ongoing interest in enhancing EHO and sets the stage for the development of more effective EHO-based approaches.

CHAPTER 3

THE BOOLEAN SATISFIABILITY PROBLEM

3.1 Introduction

Satisfiability is a fundamental problem in mathematical logic that aims to determine whether a formula admits an interpretation or assignment that makes it true. Despite its simple definition, the Boolean Satisfiability (SAT) problem plays a central and crucial role in artificial intelligence and computational complexity due to its significant theoretical and practical implications. SAT is considered a fundamental reference point, as it was the first problem proven to be N-complete[31].

It involves determining whether a Boolean Conjunctive Normal Form (CNF) formula has an assignment of truth values to its variables that satisfies it.

Due to its theoretical significance and practical applications SAT has gained significant attention, especially in artificial intelligence and problem-solving domains, leading to the development of many efficient solvers for diverse SAT instances.

This chapter introduces the foundational concepts of the Boolean Satisfiability Problem (SAT) and its role in computational complexity, particularly its classification as NP-complete. Then it explores various forms and real-world applications of SAT. The chapter concludes by outlining important problem-solving strategies, including both the complete methods like the DPLL and CDCL algorithms and the incomplete methods discussed in the previous chapter.

3.2 The Boolean Satisfiability Problem (SAT): Definition and variants

Before formally introducing the SAT problem, some key concepts from propositional logical are remained:

- **Literal:** A literal is either a propositional variable (atom) or its negation.
- **Clause:** A clause is a finite disjunction of literals.
- **Conjunctive Normal Form (CNF):** A formula is in CNF if it is expressed as a conjunction of clauses that is, a conjunction of disjunctions of literals.

3.2.1 SAT problem presentation

The Boolean Satisfiability problem (SAT) is defined over conjunctive normal form (CNF), and aims to determine whether there is a truth-value assignment to its variables that satisfies the entire formula. The formula is satisfiable if all clauses are satisfied. A clause formula is satisfiable if there exists at least one assignment of truth values that makes all clauses true (its value is 1) and an unsatisfiable clause where all literals are false under every possible assignment (their value is 0).

Defined formally by the following couple of Instance Question:

- **Instance**: m clauses C_i formed from n variables.
- **Question**: Is there an assignment of truth values to $\phi = C_1 \vee C_2 \vee \cdots \vee C_m$ that satisfies the entire formula?

Example 1.1: Assume the following SAT instance with variables $V = \{x_1, x_2, x_3\}$ and clauses $C = \{C_1, C_2, C_3\}$:

$$C_1 = (x_1 \vee x_2 \vee x_3)$$
$$C_2 = (\neg x_1 \vee x_2)$$
$$C_3 = (x_1 \vee \neg x_2 \vee \neg x_3)$$

- **Question:** Is the formula $C = \{C_1, C_2, C_3\}$ satisfiable? Is there an interpretation for which the formula is true?
- **Solution:** A potential solution to this instance is:

3.2.2 SAT problem and NP-completeness

SAT is the first problem proven to be NP-complete, a result established by Stephen Cook in 1971 *Cook-Levin Theorem*. [31]

To prove that a problem π is NP-complete, the following two conditions must be satisfied:

- π belongs to the class NP.
- For a problem π' known to be NP-complete, there exists a polynomial-time reduction from π' to π.

1. **Belongs in the NP Class:**

   A problem is considered NP if it can be determined in polynomial time and using a non-deterministic algorithm whether an instantiation of the problem is a solution to it or not. A polynomial verification time does not imply a polynomial solving time.
   A non-deterministic algorithm generally consists of two phases:

   - A *guess* phase that generates a candidate solution.
   - A *check* phase that verifies whether the guessed solution satisfies the problem constraints.

2. **Polynomial Reduction:**

   The polynomial reduction associates each instance of a problem π_1 with an equivalent instance of another problem π_2, such that solving π_2 allows us to solve π_1.
   Formally, a *polynomial transformation* is a function:

$$f : D_{\pi_1} \to D_{\pi_2}$$

where D_{π_1} and D_{π_2} are the domains of π_1 and π_2, respectively. The function f must satisfy the following two conditions:

- f is computable in polynomial time.
- For every instance $I \in Y_{\pi_1}$, we have:

$$I \in Y_{\pi_1} \iff f(I) \in Y_{\pi_2}$$

where Y_{π_1} and Y_{π_2} are the sets of instances for which the answer to π_1 and π_2 is "Yes", respectively.

3.2.3 Variants of the SAT Problem

The Boolean Satisfiability Problem (SAT) has several important variants that extend its complexity and applicability. These variants modify the structure of the SAT problem or introduce additional constraints, making them useful for different real-world applications.

➢ **k-SAT**:

In k-SAT, each clause in the CNF formula contains k literals. The complexity of the problem depends on k:

– **2-SAT**: (each clause has 2 literals) is solvable in polynomial time.
Example:

$$C_1 = (x_1 \vee x_2)$$
$$C_2 = (x_1 \vee \neg x_2)$$
$$C_3 = (x_1 \vee x_3)$$

Solution: A potential solution to this instance is: $\{x_1 = 1; x_2 = 1; x_3 = 1\}$.

– **3-SAT**: (each clause has 3 literals) is NP-complete, meaning it is computationally hard.
Example:

$$C_1 = (x_1 \vee x_2 \vee x_3)$$
$$C_2 = (x_1 \vee \neg x_2 \vee \neg x_3)$$
$$C_3 = (\neg x_1 \vee x_2 \vee x_3)$$

Solution: A potential solution to this instance is: $\{x_1 = 1; x_2 = 1; x_3 = 1\}$.

For all $k \geq 3$, the k-SAT problem remains NP-complete.

➢ **Max-SAT**:

Max-SAT is an optimization problem, where instead of determining whether a formula is fully satisfiable, the goal is to find an assignment that satisfies the maximum number of clauses. The problem is defined as follows:

– **Instance:** A set of m clauses C_i formed from n variables and an integer k.
– **Question:** Given the formula $\phi = C_1, C_2, \ldots, C_m$, is there an interpretation that satisfies a maximum number of clauses?

➢ **Max-w-SAT**:

Max-w-SAT is a variant of Max-SAT in which each clause C_i is assigned a strictly positive weight p_i. The objective is to maximize the total weight of satisfied clauses.

3.2.4 Real-World Applications of Boolean Satisfiability (SAT)

The Boolean Satisfiability Problem (SAT) is not just a theoretical concept but has extensive applications in real-world computational problems like:

- **Hardware and Software Verification:**
  SAT solvers are widely used in verifying digital circuits and detecting errors in software programs. By encoding circuit logic as Boolean formulas, engineers can check whether a design meets specifications.

- **Scheduling Problems:**
  Efficient scheduling is crucial in universities, airlines, and project management. SAT solvers can help optimise resource allocation by ensuring constraints are met.

- **Artificial Intelligence and Machine Learning:**
  A robot navigating a maze can use a SAT solver to determine the optimal path. Each possible movement is modelled as a Boolean variable, with constraints ensuring valid moves. The solver finds a sequence of steps leading to the goal without breaking any rules.

- **Cryptanalysis:**
  SAT can be applied in cryptography by representing encryption algorithms as SAT instances. This approach helps identify vulnerabilities or recover cryptographic keys, particularly in analysing weakened encryption schemes.

- **Puzzle Solving:**
  SAT solvers can be used to encode puzzle constraints, allowing for automated puzzle generation and solution verification in games like Sudoku or logic puzzles.

3.3 SAT Solvers

Being the first problem to be proven NP-complete, SAT became the reference for NP-complete class problems. A solution to SAT would have major implications for all the problems in this class. Since 1960, an important number of SAT solvers have been developed to contribute to the resolution of various complex problems. This solvers are divided into two main categories depending on whether they rely on complete or incomplete methods. Complete methods aim to determine an exact and effective solution. As the explore the entire search space, these solvers are capable of either finding a correct solution or proving that the instance has no solution. In contrast, incomplete methods cannot always find a correct solution or prove the existence of one. They often generate approximate solutions

3.3.1 Complete Methods

Complete solvers use exact algorithms to find a solution that completely satisfies the problem. They systematically explore the set of possibilities by building a tree structure connecting the elements of the solution, thus guaranteeing a correct result or proving the unsatisfactoriness of the problem.

3.3.1.1 The Davis-Putnam-Logemann-Loveland algorithm (DPLL)

The DPLL (Davis-Putnam-Logemann-Loveland) algorithm, introduced in 1962, is an improvement of the Davis-Putnam (DP) [32] and is one of the most widely used complete SAT solvers. It

explores a search tree where each node represents a recursive call and each leaf corresponds to a satisfiable solution or a contradiction. Based on chronological backtracking, it detects conflicts via the presence of empty clauses and goes back up the tree until a valid alternative is found. Two essential procedures characterize it: unit propagation, which assigns a value to literals in unit clauses to avoid inconsistencies, and pure literal elimination, which directly assigns a value to variables appearing under a single polarity. These simplifications optimize the resolution by reducing the search space and improving the efficiency of the process Logemann [32]. Three essential procedures characterize the process:

- **DPLL Algorithm:** A recursive approach that assigns values to variables and systematically explores possible solutions. When a contradiction arises, it backtracks to try different assignments.

Algorithm 4: DPLL (Davis-Putnam-Logemann-Loveland) Algorithm

Input: A formula ϕ in CNF
Output: `sat` if ϕ is satisfiable; `unsat` otherwise

if $\phi = \emptyset$ **then**
    return `sat`
$\phi \leftarrow$ `UnitPropagation`(ϕ);
if ϕ *contains an empty clause* **then**
    return `unsat`
Select a variable ι in ϕ using a heuristic;
if *DPLL*($\phi \wedge \iota$) *returns* `sat` **then**
    return `sat`
if *DPLL*($\phi \wedge \neg\iota$) *returns* `sat` **then**
    return `sat`
return `unsat`

- **Unit Propagation:** If a clause contains only one unassigned literal, that literal must be true for the formula to remain satisfiable. This step ensures that necessary assignments are made, simplifying the formula.

Algorithm 5: UnitPropagation Algorithm

Input: A formula ϕ in CNF
Output: Simplified formula ϕ

while *there is no empty clause and a unit clause* $C \in \phi$ *exists* **do**
    Consider C a unit clause and ι its unaffected literal;
    $\phi \leftarrow$ `Simplify`(ϕ, ι);
return ϕ

- **Simplify:** This step removes pure literals (literals that appear only as positive or only as negative in the formula). Assigning these literals their natural values reduces the number of clauses, making the problem easier to solve.

Algorithm 6: Simplify Algorithm

Input: A formula ϕ in CNF, a literal ι
Output: Simplified formula ϕ

```
foreach clause C ∈ φ do
    if ι ∈ C then
        φ ← φ \ {C};
    else if ῑ ∈ C then
        C ← C \ {ῑ};
return φ
```

Many improvements and variants of the DPLL algorithm have been proposed. Among them, approaches based on clause learning from conflicts (CDCL)

3.3.1.2 Conflict-Driven Clause Learning algorithm(CDCL))

The CDCL procedure was introduced by Marques-Silva and Sakallah [33] and improved by Moskewicz et al. [34]. It is an extension of the DPLL procedure. This method attempts to take advantage of conflict analysis to learn new clauses and perform non-chronological backtracking. The CDCL algorithm has two major advantages over the DPLL algorithm: (i) Backjumping allows to eliminate a section of the search tree that has no solution and (ii) clause learning helps to avoid examining sub-search trees, which have been shown to be unsolvable in a previous conflict analysis.

Algorithm 7: CDCL-Based Simplify Algorithm

Input: A formula ϕ in CNF
Output: CDCL(ϕ) result: `True` (SAT) or `False` (UNSAT)

```
V_aff ← ∅;
depth ← 0;
while true do
    foreach ι ∈ V_aff do
        φ ← φ \ {ι};
    if there exists an empty clause in φ then
        if depth = 0 then
            return False;
        C ← conflict clause inferred;
        ι ← literal ∈ C affecting conflict depth;
        depth ← max (depth(v))   ∀v ∈ C \ {ι};
        V_aff ← V_aff \ {v | depth(v) > depth};
        V_aff ← V_aff ∪ {ι};
        φ ← φ ∪ {C};
    else
        if V_aff = V then
            return True;
        Choose a literal ι from an unassigned variable;
        V_aff ← V_aff ∪ {ι};
        depth ← depth + 1;
```

Nowadays, a plethora of modern SAT solvers are fundamentally based on the DPLL and its extension CDCL.

3.3.2 Incomplete Methods

When problem instances grow larger and more complex, complete approaches are no longer able to handle them, leading to combinatorial explosion and large computational time overruns. Because of this, researchers have created new methods for looking for partial solutions that try to strike a balance between computational time and the quality of the final solution (effectiveness vs efficiency). These algorithms are stochastic and heuristic-driven, looking for the best solution rather than searching the whole search space.These solvers are typically based on metaheuristic algorithms high level strategies that guide underlying heuristics using randomness and problem-specific knowledge to efficiently explore the solution space. Metaheuristics such as evolutionary algorithms, swarm intelligence, and local search are discussed in detail in the previous chapter, which lays the foundation for their application to SAT solving.

3.4 Conclusion

This chapter explores the SAT problem, a fundamental challenge in computational complexity and the first decision problem proven to be NP-complete. Due to its significance, extensive research has been conducted to develop efficient solutions, given the potential impact of solving SAT on all NP-complete problems.

The most commonly used SAT solvers fall into two main categories. Complete algorithms, such as DPLL, are designed to guarantee finding an exact solution if one exists or prove that the problem is unsatisfiable. On the other hand, incomplete algorithms, often based on metaheuristic approaches, aim to quickly find approximate solutions without guaranteeing optimality or proving the inexistence of a solution.

The aim of this work is to contribute to the resolution of NP-complete problems, and particularly the SAT problem, through the exploration of hybrid strategies integrating metaheuristic algorithms and data mining methods.

Part III

Contribution and Implementation

CHAPTER 4

ADAPTING AND ENHANCING EHO ALGORITHM

4.1 Introduction

The Boolean Satisfiability Problem is one of the most critical and fundamental problems in computer science. As solving it enables the resolution of a wide range of real-world challenges. Indeed, many complex computational tasks can be transformed into SAT instances, making SAT solving a powerful and universal approach.
As a result, numerous solvers and especially metaheuristic algorithms have been proposed to address this problem.
In this context, we chose to explore and adapt the Elephant Herding Optimization algorithm for solving SAT effectively.

In this chapter, an adapted version, to the binary nature of the SAT problem, of the Elephant Herding Optimiztion algorithm is proposed. The limitations of this basic version are then presented and an improved version is introduced.

4.2 Implementation of Sat Solver Using Basic EHO

In this section, the implementation process of the proposed SAT solver based on the Elephant Herding Optimization algorithm is presented. The objective is to adapt the basic EHO, originally designed for continuous optimization problems, to handle the binary problems specifically to the SAT problem.

This implementation is structured around two main components:

- The representation and preprocessing of CNF formulas.
- The adaptation of the EHO algorithm for binary problem-solving.

The integration of both parts is illustrated in the global architecture below (Figure 4.1).

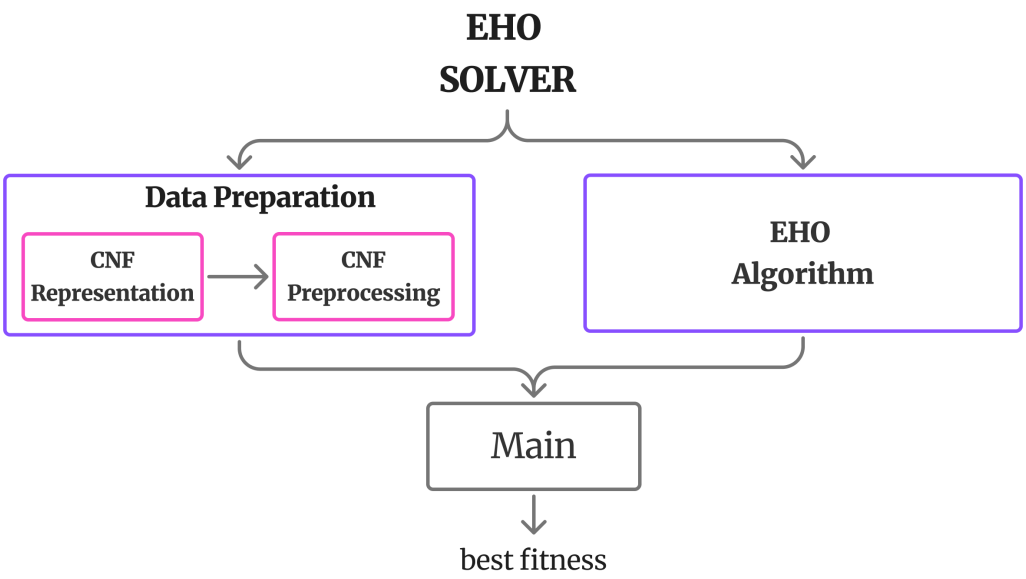

Figure 4.1: Global Architecture of the EHO-based SAT Solver

4.2.1 Binary Representation and CNF Preprocessing

The first stage in developing the proposed system or any computational system is preparing the dataset. Once an appropriate benchmark dataset is selected, it is essential to analyze its structure and identify potential issues that could affect performance.
In this work, Conjunctive Normal Form (CNF) files that define SAT instances are considered.
To better understand the structure, content and variability of these benchmarks, scrolling through a wide range of SAT benchmarks of different size is essential. This exploration helps determine the most suitable data structure for representing CNF formulas efficiently.

The adopted data structure, in this work, is a list-of-lists because of the significant difference in the number of variables and clauses from one instance to another. This flexible and dynamic representation ensures that the representation remains adaptable across different instances and supports efficient clause manipulation during preprocessing and optimization.

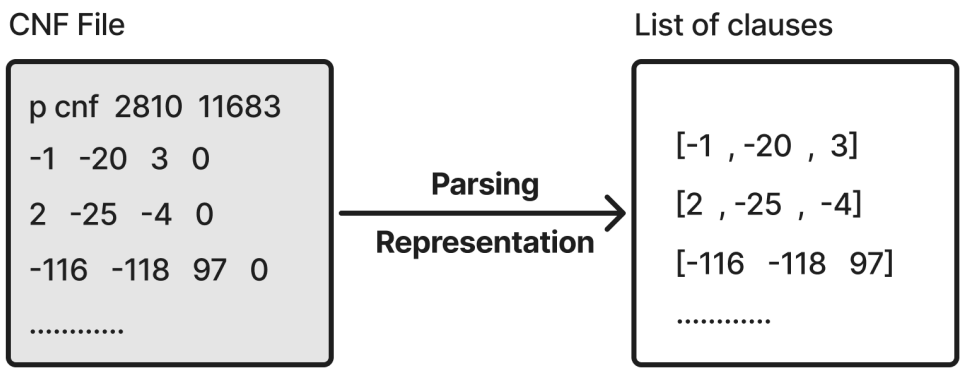

Figure 4.2: Parsing and Representing Clauses from a CNF File

Figure 4.2 illustrates the transformation of raw CNF input into a structured list-of-lists format through the parsing process

After structuring the CNF clauses, the preprocessing phase is performed. This stage aims to refine the dataset by removing redundant or unnecessary information to ensure the optimization process operates on a clean and efficient clauses base.
While in general, SAT benchmarks do not require extensive preprocessing since they consist of simple numeric literals with no complex numerical data. The observations of the benchmarks used in our project motivated applying a lightweight but meaningful preprocessing step, including:

- **Remove tautological clauses:**

  A tautological clause contains both a literal and its negation, for example [X1, ¬X1, X3]. Such clauses are always true and only add unnecessary complexity to the benchmark. Removing them helps speed up optimization by focusing on meaningful constraints only.

- **Remove duplicate clauses:**

  Duplicate clauses are repeated copies of the same clause, such as having [X2, ¬X3, X4] multiple times. These clauses add no new information but increase the problem size and processing time.
  Removing duplicates reduces redundancy and makes the optimization more efficient without losing any constraints.

- **Remove included clauses:**

  When a clause fully contains another smaller clause, the larger one can be removed without affecting satisfiability.
  Consider,for example, the clauses [X1 , X2] and [X1], satisfying the second clause guarantees the satisfaction of the first one. The clause [X1 , X2] can then be removed because [X1] is more restrictive.
  Keeping the smaller clause preserves or strengthens the constraints, simplifying the problem for the solver.

4.2.2 Adapting Elephant Herding Optimization Algorithm for Solving SAT Problem

In this section, an adapted version of the EHO algorithm is proposed. The original EHO is modified in order to operate in a binary search space and incorporates SAT-specific components such as clause-based fitness evaluation and Update operation.
The workflow of the proposed method is described below (The Figure 4.3), outlining each step of the adaptation of EHO to efficiently solve the SAT problem.

1. **Population Initialization and Clan Division**

   This first step involves generating a diverse initial population of elephants randomly, where each elephant represents a potential solution to the SAT problem.
   In this representation, the length of a solution, represented by a binary vector corresponds to the number of Boolean variables in the CNF formula. Each bit in the vector is initialized randomly with either 0 (false) or 1 (true), indicating a truth assignment to a variable.

   Example 4.1. *Let us consider a SAT instance with* 6 *variables:* $x_1, x_2, \ldots, x_6$. *An example of an elephant (solution) could be:*

$$E_1 = [1,\ 0,\ 1,\ 1,\ 0,\ 0]$$

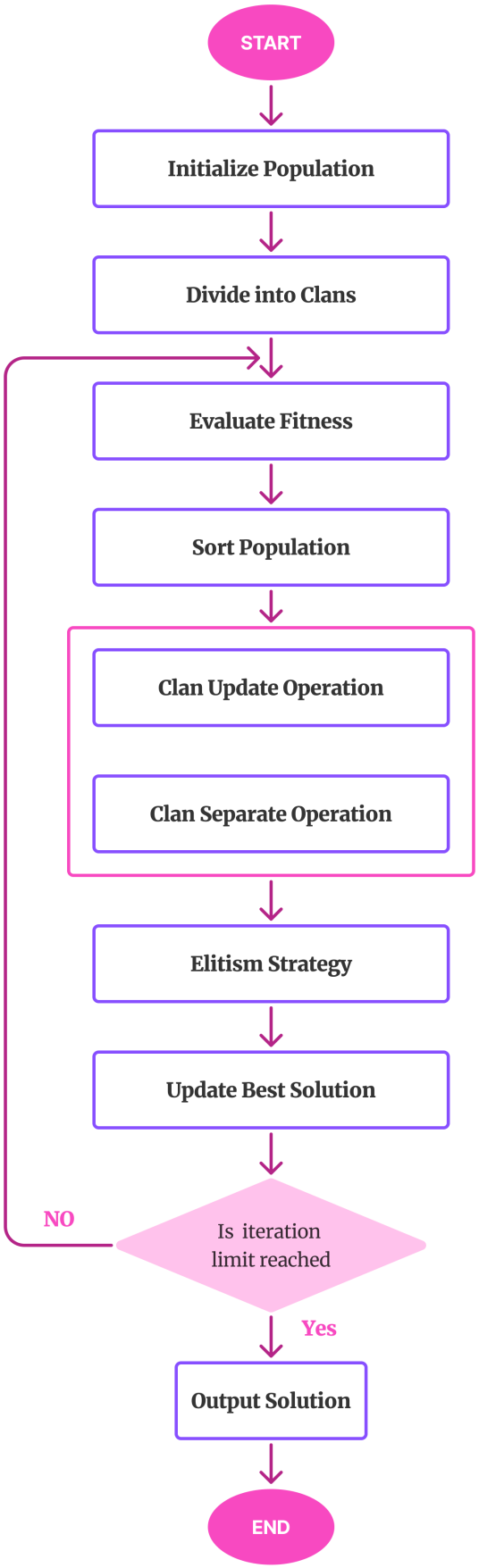

Figure 4.3: Implementation Workflow of the EHO Algorithm

Once the population is generated, the entire population of elephants is divided into a predefined number of clans using sequential partitioning, where each clan receives a contiguous segment of the population. Each clan is composed of a fixed number of individuals (elephants). This division allows the algorithm to explore multiple regions of the search space simultaneously, enhancing both the diversity and the convergence capabilities of the algorithm.

Example 4.2. *Let's consider a population of 12 elephants and devided into 3 clans. Using sequential partitioning, the resulting clans are as follows:*

- ***Clan 1*** *receives elephants at indices 0 to 3* $\rightarrow$ *[e_0, e_1, e_2, e_3]*
- ***Clan 2*** *receives elephants at indices 4 to 7* $\rightarrow$ *[e_4, e_5, e_6, e_7]*
- ***Clan 3*** *receives elephants at indices 8 to 11* $\rightarrow$ *[e_8, e_9, e_{10}, e_{11}]*

This method ensures that all elephants are distributed equally among the clans. Each clan evolves independently in the following steps of the algorithm.

2. **Fitness Evaluation**

The fitness evaluation function plays a critical role in guiding the optimization process of any metaheuristic algorithm. It serves as the objective metric by which the quality of candidate solutions is measured and compared. This function varies depending on the nature of the problem being addressed.
In the case of the MAX-SAT problem, the fitness function is designed to reflect how well a solution satisfies the given Boolean formula. Specifically, the number of satisfied clauses is to be maximized. Each solution (elephant) is evaluated based on the number of clauses it satisfies. The more satisfied clauses, the higher the fitness score.

Formally, the fitness of a solution S is given by:

$$\text{Fitness}(S) = \sum_{i=1}^{m} \text{sat}(C_i, S) \tag{4.1}$$

where:

- S is a candidate solution ,
- C_i is the i^{th} clause in the CNF formula,
- m is the total number of clauses,
- $\text{sat}(C_i, S)$ is a function that returns 1 if clause C_i is satisfied by solution S, and 0 otherwise.

Example 4.3. *Let's consider the following CNF formula with 3 variables and 4 clauses:*

$$F = (x_1 \vee \neg x_2) \wedge (\neg x_1 \vee x_3) \wedge (x_2 \vee \neg x_3) \wedge (\neg x_1 \vee \neg x_2 \vee x_3)$$

Let a candidate solution be $S = [1, 0, 1]$

The fitness of solution S is calculated by checking how many clauses are satisfied under this assignment, as represented in the Figure 4.4.

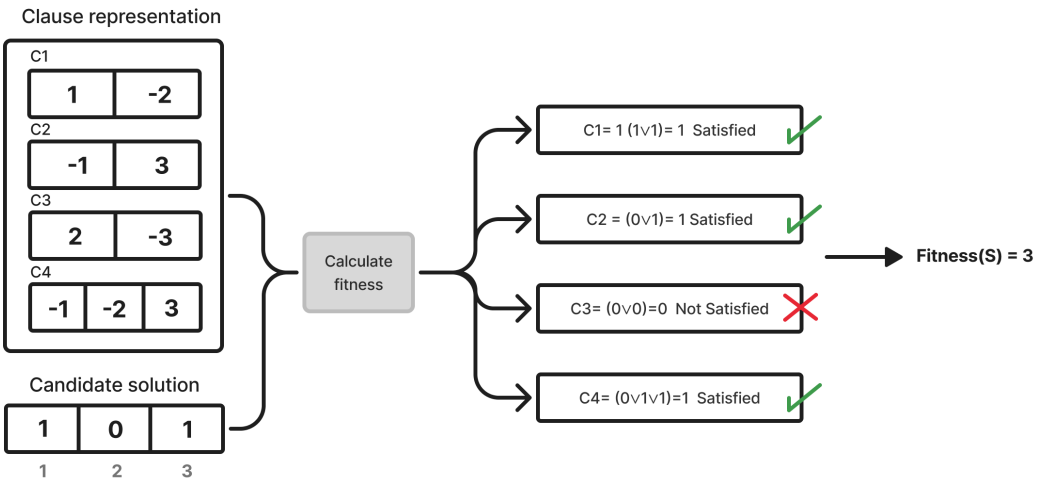

Figure 4.4: Example of Fitness Evaluation in MAX-SAT

3. Clan Update Operator

The clan update operator is a crucial mechanism in the Elephant Herding Optimization algorithm. It ensures a balance between exploitation of good solutions and exploration of new areas in the solution space, thus helping to maintain population diversity and avoid premature convergence.
Each clan is composed of multiple elephants (candidate solutions), and the evolution of each individual depends on two major influences:

- The matriarch, which is the best fitness within a clan. It represents the most promising solution discovered by the group and influences the update process of the non-matriarch elephants.
- The clan center, which is the average position (bitwise) of all elephants in the clan. It reflects the overall search direction and general behavior of the clan and influences the update of the matriarch.

Figure 4.5 illustrate the updating position of the elephant within a clan. This update depends on the nature of the elephant, that is, whether it is a matriarch or not.

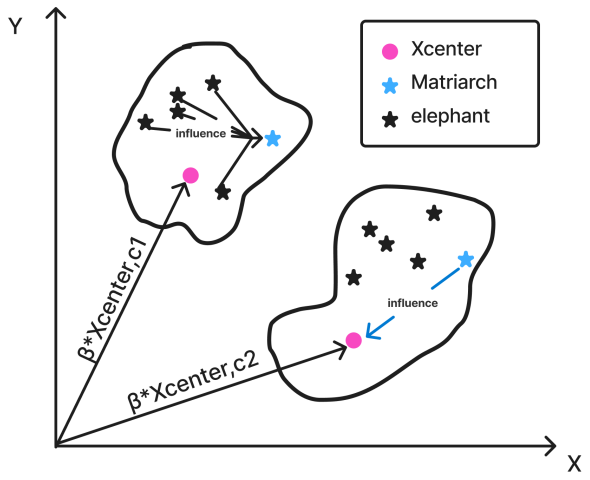

Figure 4.5: Illustration of the Conceptual Influence in Update Operator

In the SAT context, the clan center is computed by averaging each bit position across all elephants in the clan and rounding the result to 0 or 1 to maintain the binary format (Example 4.4). This process produces a representative binary vector that captures the dominant traits of the clan. The center is then used to influence the matriarch's update.

Example 4.4. *Let's consider a simple example of a clan composed of 3 elephants:*

$$E = \begin{bmatrix} 1 & 0 & 1 & 0 & 1 \\ 1 & 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 & 1 \end{bmatrix}$$

We calculate the center of this clan as shown in Figure 4.6:

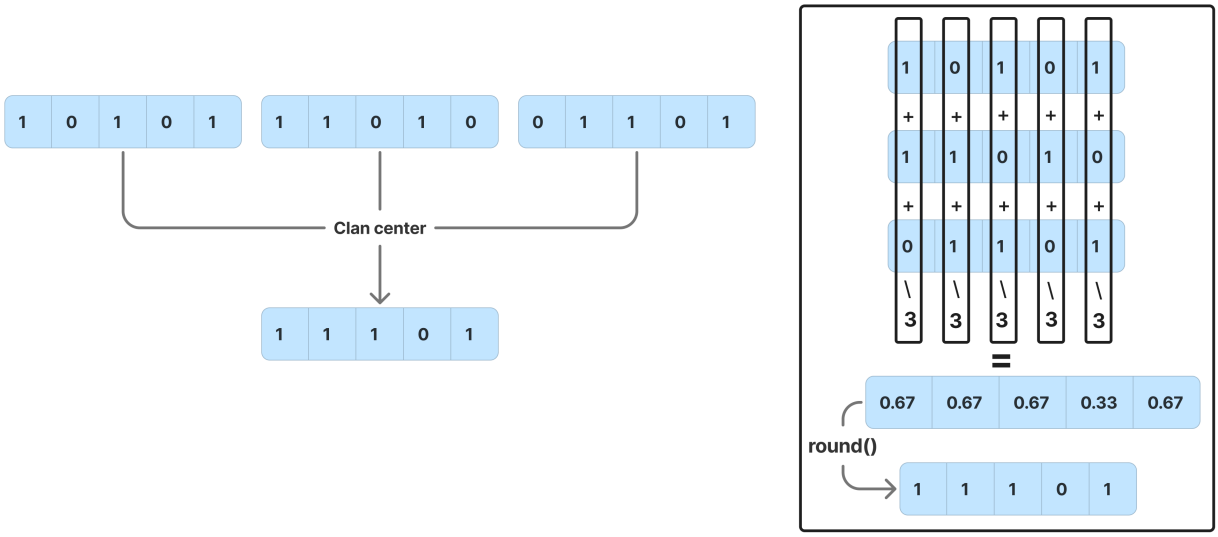

Figure 4.6: Illustration of the clan calculation process

The update process is carried out as follows;

- ⇨ For the **matriarch**, the position is updated by moving toward the *clan center*, guided by a control parameter β, which determines the degree of influence from the center. For each bit in the binary solution, a weighted combination of the current bit and the corresponding bit in the clan center is computed. This value is then compared to a random threshold to introduce stochasticity and preserve diversity.

  The update rule is defined as:

$$\text{new\_bit} = \begin{cases} 1, & \text{if } (1-\beta) \cdot \text{elephant}[i] + \beta \cdot \text{center}[i] > \text{random()} \\ 0, & \text{otherwise} \end{cases} \tag{4.2}$$

  This update allows the matriarch to slightly adjust her position in the direction of the general trend of the clan, without drastically altering her high-quality solution.

➪ For the **other elephants** in the clan, the update process is guided by the *matriarch* and controlled by a parameter α, which determines the extent to which an elephant follows the best individual in the clan. For each bit in the solution, a weighted combination of the current elephant's bit and the corresponding bit of the matriarch is computed. A random threshold is then used to decide the new bit value.

The update rule is given by:

$$\text{new\_bit} = \begin{cases} 1, & \text{if } (1-\alpha) \cdot \text{elephant}[i] + \alpha \cdot \text{matriarch}[i] > \text{random()} \\ 0, & \text{otherwise} \end{cases} \tag{4.3}$$

This mechanism allows the clan members to progressively align with the strongest solution, while maintaining variation necessary for exploration.

4. **Clan Separating Operator**

In the context of SAT problem, the separating operator identifies, for each clan, the worst elephant,(the individual with the *lowest fitness value*). This elephant is then replaced by a new randomly generated solution.
This operation helps introduce diversity into the population and prevents premature convergence by encouraging the exploration of new areas in the search space.

5. **Apply Elitism Strategy**

To further enhance the quality of solutions over generations, an elitism strategy is applied.
In each generation, the top-k best individuals are preserved before any updates. After applying the update and separating operators to all clans, the k worst individuals in the new population are identified and replaced by the previously saved elite solutions. This ensures that the best found solutions are not lost due to the stochastic nature of the algorithm.
Reinserting the elites ensures the maintain of high-quality candidates in the search space, helps preserve progress across generations, and promotes faster convergence toward optimal or near-optimal solutions.

These steps, except the population initialization step, represent one iteration of the EHO process. The best solution of all the population is considered as best solution. This solution is returned if it satisfies the problem, otherwise the process of updating-separating continue until reaching the maximum number of iteration. In such case, the best solution across all generations is returned.

All these steps are summarized in the following pseudo-code (8);

Algorithm 8: Adaptive Elephant Herding Optimization for SAT

Input : CNF clauses, population size N, number of clans c, max generations $t_{\max}$, learning rates α, β, elitism count k

Output: Best solution x_{best}

Initialize population of N elephants (binary vectors of length n)
Divide population into c clans of size N/c randomly; Evaluate fitness of all elephants (using Formula 4.1)
Sort population by descending fitness
Save top-k elephants as elites
while $t < t_{max}$ **do**
    for *each clan* $clan_i$ *from 1 to* c **do**
        `/* Applying Update Operator */`
        Identify matriarch M with best fitness in $clan_i$
        Compute clan center $C =$ mean of elephants in $clan_i$ (rounded to binary)
        for *each elephant* e_j *in* $clan_i$ **do**
            if e_j *is matriarch* **then**
                Update e_j using Formula 4.2
            else
                Update e_j using Formula 4.3
        `/* Applying Separation Operator */`
        Replace worst elephant in $clan_i$ with new random binary vector
    `/* Applying Elitism Strategy */`
    Replace worst k elephants with saved k elites
    $t \leftarrow t + 1$
return x_{best}

Complexity: In the worst-case scenario, the algorithm evaluates and updates all N elephants in each of the $t_{\max}$ generations. For each elephant, operations such as fitness evaluation and binary vector updates are performed, which have a cost proportional to the vector length n. Sorting the population and computing clan centers are also involved, but their cost is dominated by the per-generation processing of the full population.

Hence, the overall time complexity of the adapted EHO algorithm is:

$$\mathcal{O}(t_{\max} \cdot N \cdot n)$$

4.3 Improved Basic EHO

A crucial criterion while evaluating the effectiveness of a metaheuristic is its ability to maintain an equilibrium between exploration and exploitation. Nevertheless, applying the basic adapted version of EHO indicates a major issue. In fact, the fitness value remained nearly constant over many iterations. This stagnation, or in some cases near-stagnation, indicates that the algorithm was struggling to escape local optima. Observations suggest that the population lacks sufficient diversity, and that the standard update mechanisms were not effective enough in driving the search towards better regions of the solution space.

To cope with the stagnation, two enhancement strategies were proposed and implemented. The first proposition is a mutation mechanism that deals with stagnation and is incorporated to inject diversity when the best fitness remains unchanged over several iterations. The second proposed

improvement is a redivision strategy to restructure clans during near-stagnation phases were the best fitness improves very slowly over iterations.

4.3.1 Integration of a mutation operator (IEHO-M)

To solve the stagnation problem and enhance the exploration capabilities of the population, a mutation strategy is integrated to the EHO algorithm. This strategy prevent premature convergence during the optimization process.
The integrated mutation process is structured as follows:

- The mutation is applied only when the algorithm detects stagnation (based to the stagnation threshold), meaning the best solution hasn't improved over a predefined number of generations. This suggests the population may be stuck in a local optimum. Mutation is then triggered to introduce diversity and help the search escape this situation.
- Once stagnation is detected, the mutation process is activated to counter premature convergence. It proceeds by examining each variable (bit) in a solution and applying a probabilistic rule defined by the mutation rate to decide whether that variable should be altered. This controlled randomness increases the chances of escaping local optima and allows the algorithm to explore previously unreached areas of the search space.
- The mutation process is applied to all individuals in the population except the top k best ones. This helps keep good solutions safe while letting the rest explore new possibilities.

Example 4.5. *Let's consider a population composed of 3 individuals represented by binary vectors of dimension 5:*

- *Individual 0: [1, 1, 1, 1, 1] (Elite)*
- *Individual 1: [0, 1, 0, 0, 1] (Mutated individual 1)*
- *Individual 2: [1, 0, 1, 0, 0] (Mutated individual 2)*

Assuming a high mutation rate for this example = 0.5 (meaning each variable has a 50% chance of being modified) .

The mutation process is illustrated in the following figure (4.7):

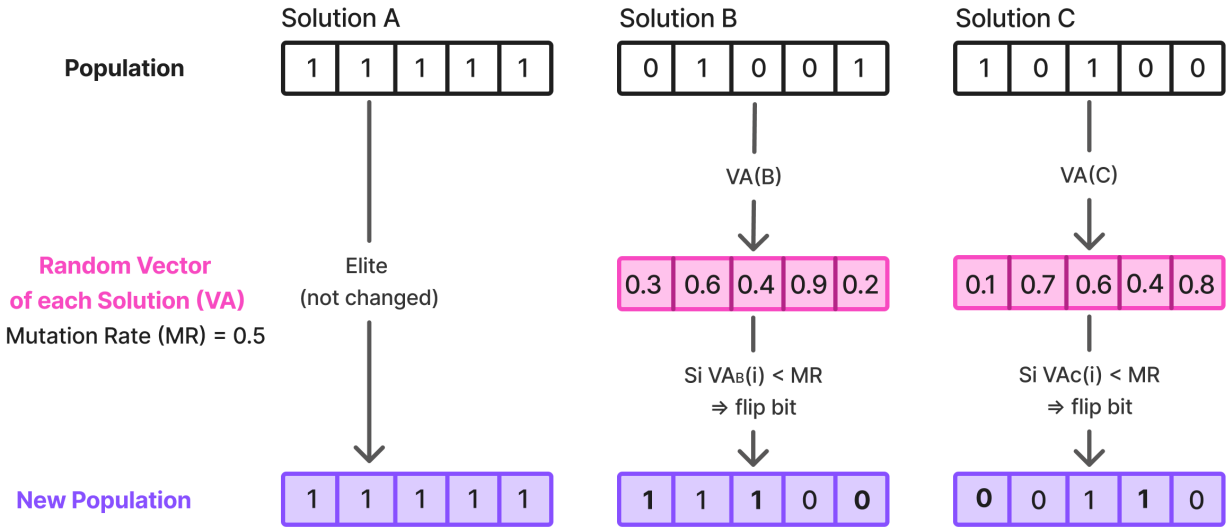

Figure 4.7: Illustration of the mutation process

All the procces of this variant are summarized in the following pseudocode (9);

Algorithm 9: Improved Elephant Herding Optimization with Conditional Mutation

Input : CNF clauses, population size N, number of clans c, max generations $t_{\max}$, learning rates α, β, elitism count k, mutation rate m, stagnation threshold s

Output: Best solution x_{best}

Initialize population of N elephants (binary vectors of length n)
Divide population into c clans of size N/c randomly
Evaluate fitness of all elephants (using Formula 4.1)
Sort population by descending fitness
Save top-k elephants as elites
while $t < t_{max}$ **do**
    for *each clan* $clan_i$ *from 1 to* c **do**
        /* Apply **Clan Update Operator** to $clan_i$ */
        /* Apply **Separation Operator** to $clan_i$ */
    /* Applying **Elitism Strategy** to replace worst k elephants with saved elites */
    /* Detecting Stagnation */
    if *no improvement* **then**
        Increment stagnation_counter
    else
        Reset stagnation_counter $\leftarrow 0$
    /* Apply Mutation if Stagnation Persists */
    if *stagnation_counter* $\geq s$ **then**
        for *each non-elite elephant* e_j **do**
            for *each bit* x_i *in* e_j **do**
                Flip x_i with probability m
        Reset stagnation_counter $\leftarrow 0$
    $t \leftarrow t + 1$
n **return** x_{best}

Complexity: In the worst-case scenario, the algorithm performs update and separation operations for all N elephants across $t_{\max}$ generations. If stagnation occurs for s consecutive generations, mutation is applied to each bit of every non-elite elephant, costing $\mathcal{O}(N \cdot n)$. Since this may happen multiple times, it adds additional overhead compared to the standard EHO.
Nevertheless, the dominant cost remains the per-generation processing of the population and potential mutation.
Thus, the overall time complexity is:

$$\mathcal{O}(t_{\max} \cdot N \cdot n)$$

4.3.2 Integration of Re-division Strategy (IEHO-RD)

As another way to address the stagnation problem besides using mutation, a re-division strategy is applied. This approach modifies the population structure to promote diversity and renew exploration potential.

The process works as follows:

- The re-division strategy is activated when stagnation detected over a predefined number of generations . This signals that the current population structure may be limiting exploration.
- After the strategy is activated, the entire population is randomly shuffled and then repartitioned into new clans. This breaks the existing clan structures, allowing new interactions between individuals that were not previously in the same group.

This restructuring encourages new group dynamics and exploration of different regions in the search space, increasing the chance of finding better solutions while avoiding repetitive patterns and restoring effective search behavior.

> **NB:**The re-division process follows the same method used in the initial clan formation. The key difference is that it occurs dynamically during the search, only when no improvement is observed for several consecutive generations.

All the procces of this variant are summarized in the following pseudocode (10);

Algorithm 10: Improved Elephant Herding Optimization with Redivision Strategy

Input : CNF clauses, population size N, number of clans c, max generations $t_{\max}$, learning rates α, β, elitism count k, mutation rate m, stagnation threshold s
Output: Best solution x_{best}

Initialize population of N elephants (binary vectors of length n)
Divide population into c clans of size N/c randomly
Evaluate fitness of all elephants (using Formula 4.1)
Sort population by descending fitness
Save top-k elephants as elites
while $t < t_{max}$ **do**
    for *each clan* $clan_i$ *from 1 to* c **do**
        /* Apply **Clan Update Operator** to $clan_i$ */
        /* Apply **Separation Operator** to $clan_i$ */
    /* Applying **Elitism Strategy** to replace worst k elephants with saved elites */
    /* Detecting Stagnation */
    if *no improvement* **then**
        Increment stagnation_counter
    else
        Reset stagnation_counter $\leftarrow 0$
    /* Apply Re-division if Stagnation Persists */
    if *stagnation_counter* $\geq s$ **then**
        Shuffle population randomly
        Re-divide into new clans of size N/c randomly; Reset stagnation_counter $\leftarrow 0$
    $t \leftarrow t + 1$
return x_{best}

Complexity: In the worst-case scenario, the algorithm processes all N elephants at each generation for both the update and separation operations, repeated over $t_{\max}$ generations. When stagnation is detected, the population is shuffled and re-divided into clans, which costs $\mathcal{O}(N)$.

Since shuffling and re-clustering are linear in N, their impact remains negligible compared to the per-generation operations, which dominate the complexity. Even if this re-division were applied in every generation, the total cost would become $\mathcal{O}(t_{\max} \cdot N \cdot n + t_{\max} \cdot N)$, which simplifies to $\mathcal{O}(t_{\max} \cdot N \cdot (n+1))$, and asymptotically remains:

$$\mathcal{O}(t_{\max} \cdot N \cdot n)$$

4.3.3 Combined Strategy: Mutation and Re-division (IEHO-MRD)

In population-based metaheuristics, stagnation can frequently occur during the search process.. When applying the adapted EHO algorithm, it appeared in two forms:

- Full stagnation, when the best solution does not improve at all for several consecutive generations.
- Near-stagnation, when the improvement is very slight or too slow, indicating possible convergence to a local optimum.

In the previously proposed strategy, we addressed full stagnation by using mutation and redivision operators independently. However, we now choose to combine these two operators to tackle both forms of stagnation full and near stagnation within a unified mechanism.

- ❖ **Mutation to resolve full stagnation:** Mutation introduces randomness and diversity into the population, helping escape from flat regions or repeated solutions where no progress is made across generations. This disrupts premature convergence and encourages exploration.
- ❖ **Redivision to resolve near-stagnation:** Redividing the population reorganizes individuals into new clans or groups, allowing for better information sharing and fresh dynamics. This stimulates gradual improvements when progress slows down, helping the algorithm escape local optima.

By applying this combined strategy, both full stagnation and near-stagnation are effectively mitigated through enhanced exploration and diversity.

All the procces of this variant are summarized in the following pseudocode (11);

Algorithm 11: Improved Elephant Herding Optimization with Combined Mutation and Re-division Strategy

Input : CNF clauses, population size N, number of clans c, max generations $t_{\max}$, learning rates α, β, elitism count k, mutation rate m, stagnation threshold s, window size w,slow progress threshold $\delta_{\min}$

Output: Best solution x_{best}

Initialize population of N elephants (binary vectors of length n)
Divide population into c clans of size N/c randomly
Evaluate fitness of all elephants (using Formula 4.1)
Sort population by descending fitness
Save top-k elephants as elites
while $t < t_{max}$ **do**
    for *each clan* $clan_i$ *from 1 to* c **do**
        /* Apply **Clan Update Operator** to $clan_i$ */
        /* Apply **Separation Operator** to $clan_i$ */
    /* Applying **Elitism Strategy** to replace worst k elephants with saved elites */
    /* Detect Full Stagnation */
    if *best fitness did not improve* **then**
        Increment $stagnation\_counter$
    else
        Reset $stagnation\_counter \leftarrow 0$
    /* Apply Mutation (Full Stagnation) */
    if $stagnation\_counter \geq s$ **then**
        for *each non-elite elephant* e_j **do**
            for *each bit* x_i *in* e_j **do**
                Flip x_i with probability m
        Reset $stagnation\_counter \leftarrow 0$
    /* Detect Near-Stagnation (Slow Progress) */
    if $t \mod w == 0$ ***and*** $t \geq w$ **then**
        $\Delta \leftarrow$ fitness_trace[t] − fitness_trace[$t - w + 1$]
    /* Apply Re-division (Near-Stagnation) */
    if $\Delta < \delta_{\min}$ **then**
        Shuffle population randomly
        Re-divide into new clans of size N/c randomly
    $t \leftarrow t + 1$
return x_{best}

Complexity: In the worst-case scenario, the algorithm processes all N elephants at each generation for both the update and separation operations, repeated over $t_{\max}$ generations. When full stagnation is detected, mutation is applied to all non-elite elephants, each with n bits, costing $\mathcal{O}(N \cdot n)$. Additionally, when near-stagnation is detected, the population is shuffled and re-divided into clans, which costs $\mathcal{O}(N)$.

Since both mutation and re-clustering are linear or linearithmic in nature, their impact remains

negligible compared to the dominant per-generation operations. Even if mutation and re-division were applied in every generation, the total cost would become $\mathcal{O}(t_{\text{max}} \cdot N \cdot n + t_{\text{max}} \cdot N)$, which simplifies to $\mathcal{O}(t_{\text{max}} \cdot N \cdot (n+1))$, and asymptotically remains:

$$\mathcal{O}(t_{\text{max}} \cdot N \cdot n)$$

4.4 Conclusion

Throughout this chapter, we presented the adaptation of Elephant Herding Optimization for solving the SAT problem by making it compatible with binary CNF clauses. We did not stop just at adapting the algorithm but also addressed its challenges by introducing mutation and population re-division to enhance diversity and performance. These improvements significantly boosted EHO's effectiveness on SAT problems. They also lay the foundation for a hybrid with other techniques, which will be explored in the next chapter.

CHAPTER 5

HYBRIDIZATION OF EHO WITH DATA MINING TECHNIQUES

5.1 Introduction

The Elephant Herding Optimization algorithm is a nature-inspired metaheuristic that mimics the social behavior of elephant herds. It organizes the population into clans led by matriarchs and employs strategies to balance exploration and exploitation of the search space. While effective, its standard version has elements that can be further refined to enhance performance.
To improve the overall performance of EHO, an hybridizing with data mining clustering techniques is proposed, specifically K-Means and DBSCAN. Throughout this chapter, the motivation behind this hybridization is presented. Followed by a description of both k-means and DBSCAN approaches

5.2 K-Means-Based EHO (EHO-Kmeans)

In the standard EHO algorithm, clans are formed through random partitioning of the population, without accounting for individual similarity or fitness. This uninformed division often results in weak intra-clan collaboration, inefficient local search behavior, and limited exploration of the solution space.

To address these limitations, the integration of the K-means clustering algorithm into the EHO framework is proposed. Instead of random division, individuals are grouped into clans based on their solution similarity. This clustering mechanism enhances the search dynamics of EHO by forming semantically meaningful groups, which promotes efficient local exploitation within each clan and encourages global diversity across clans.
As expecting result, the algorithm should become more effective at avoiding premature convergence and explore the search space more thoroughly.

5.2.1 Methodology of the Hybrid EHO-K-means Approach

The hybrid EHO-K-means approach consists on embedding K-means clustering into the clan division step. This integration replaces the original random division with a structured, similarity-based partitioning of the population. Figure 5.1 exhibits the hybridization process which involves

several key steps from cluster initialization to iterative refinement ensuring that individuals are assigned to the most appropriate clans.

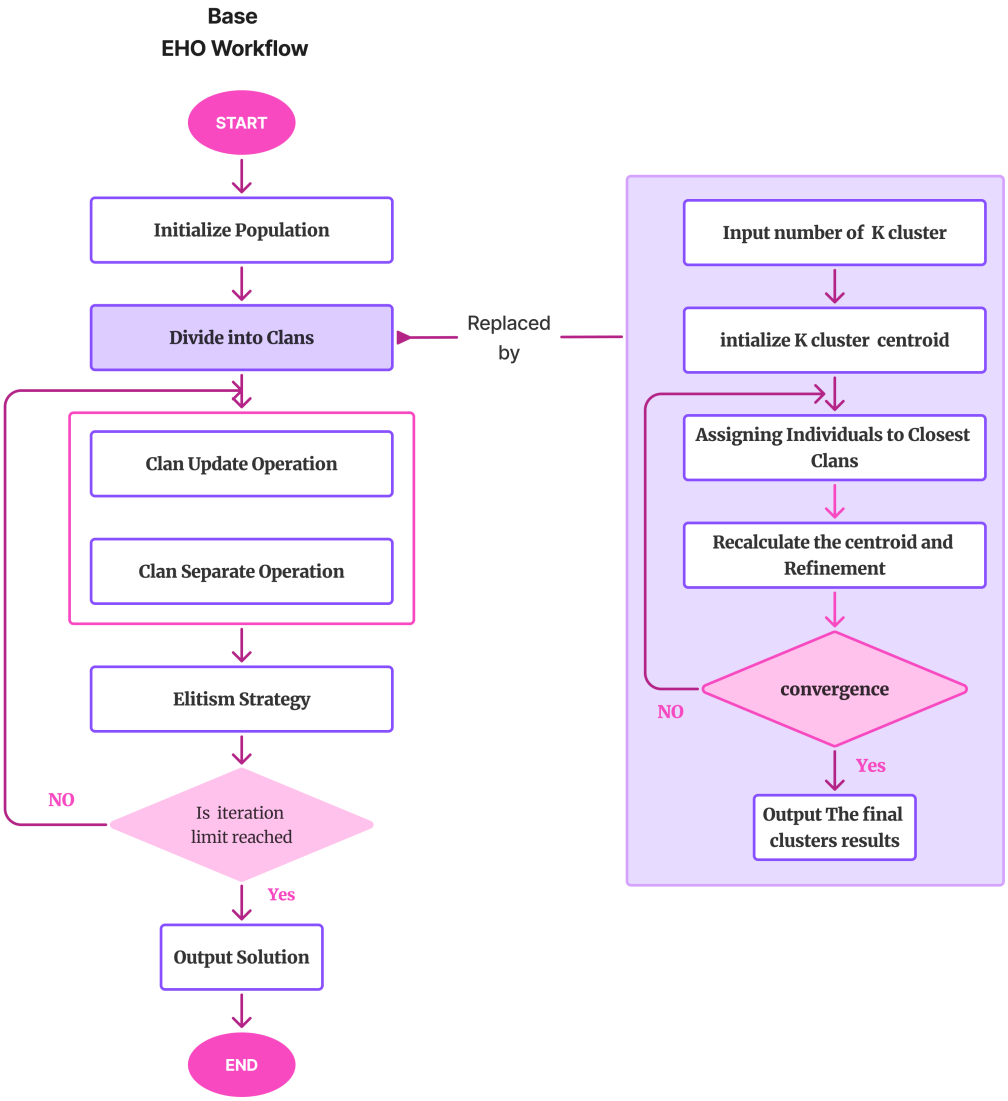

Figure 5.1: Implementation Workflow of K-Means-Based EHO

The process of dividing the population using k-means follows these steps:

1. **Clan Centroid Initialization**

   The process begins by selecting k initial centroids, where k is the desired number of clans. These centroids act as the initial leaders around whom the rest of the population will group.

   Example 5.1. *Let's consider a population of 7 individuals*

   $$\{10101, 01100, 11101, 00010, 10111, 01010, 11001\}$$

   The number of clans (clusters) is set to $k = 2$.

   Initial centroids are randomly selected:

   - ***Centroid 1**: Individual 1* $\rightarrow$ *[1, 0, 1, 0, 1]*

- ***Centroid 2***: *Individual 4 → [0, 0, 0, 1, 0]*

2. **Assigning Individuals to Closest Clans**

Centroids being set, each member of the population is paired with the closest centroid to create the first clans. This assignment determines the degree of similarity between two solutions binary vectors based on the Hamming distance, that quantifies the positions at which the binary vectors representations of these solutions diverge. counts the number of bit positions where they differ (Equation 5.1). The lower the Hamming distance, the more similar the two solutions vectors are.
After calculating the distance to each centroid, each individual is paired with the centroid that has the smallest distance, forming k initial clusters.

$$\text{Hamming Distance}(A, B) = \sum_{i=1}^{n} [A_i \neq B_i] \tag{5.1}$$

Example 5.2. *Consider two binary solutions:*

$$A = 10101 \quad \textit{and} \quad B = 11001$$

The Hamming distance between A and B is computed as we illustrated in following Figure 5.2.

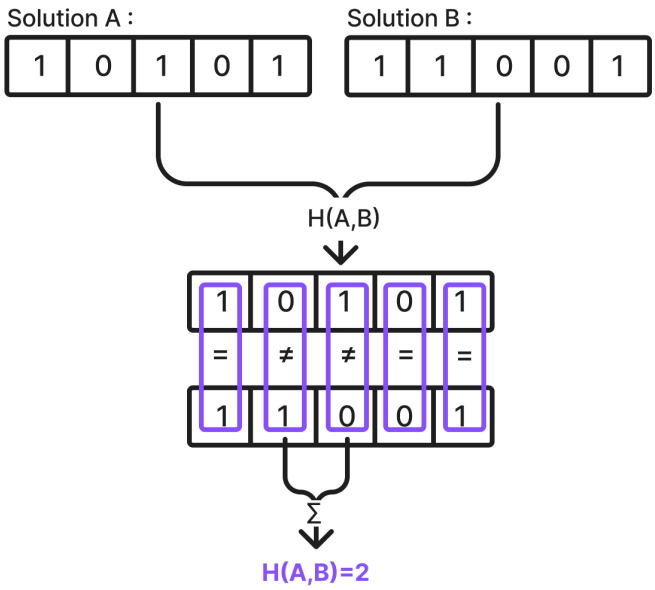

Figure 5.2: Illustration of Hamming Distance Computation

NB: The choice of using the Hamming distance in the SAT problem context is influenced by its binary nature. It represents a natural and efficient way to measure the structural similarity between binary solutions.

3. **Recalculate the centroid and Refinement**

After assigning individuals to the closest clans, the centroid of each clan is recalculated as the mean value of the cluster, to better represent its current members. This step follows the same process as the initial calculation of the clan center within the EHO algorithm (Figure 4.6).

4. **Convergence of Clan Structures**

   The process of calculating distance, assigning solutions to the closest cluster or clan and recalculating the centroids is repeated until individuals or solutions stabilization, that is, convergence of clustering.

All these steps are summarized in the following pseudo-code (12).

Algorithm 12: Elephant Herding Optimization with K-Means Clustering for SAT

Input : CNF clauses, population size N, number of clans c, max generations $t_{\max}$, learning rates α, β, elitism count k, max K-Means iterations max_kmeans_iter

Output: Best solution x_{best}

Initialize population of N elephants (binary vectors of length n)
Evaluate fitness of all elephants (using Formula 4.1)
Apply K-Means clustering to divide population into c clans
Sort population by descending fitness
Save top-k elephants as elites
$t \leftarrow 0$
while $t < t_{max}$ **do**
    for *each clan* $clan_i$ *from 1 to* c **do**
        /* Applying `Clan Update Operator` to $clan_i$ */
        /* Applying `Separation Operator` to $clan_i$ */
    /* Rebuild Clans Using K-Means */
    Merge all elephants into a new population
    Randomly select c elephants as initial centroids
    for $iter = 1$ **to** max_kmeans_iter **do**
        foreach *elephant* e_j *in population* **do**
            foreach *centroid* C_i **do**
                Compute Hamming distance $d(e_j, C_i)$ using Formula 5.1
            Assign e_j to cluster with nearest centroid
        foreach *cluster* $cluster_i$ **do**
            for $k = 1$ **to** n **do**
                Count number of 1s in bit position k
                if *majority is 1* **then**
                    $C_i[k] \leftarrow 1$
                else
                    $C_i[k] \leftarrow 0$
        if *centroids unchanged* **then**
            break
    /* Applying `Elitism Strategy` to replace worst k elephants with saved elites */
    $t \leftarrow t + 1$
return x_{best}

Complexity: In this variant, the algorithm processes all N elephants at each generation through the clan update and separation operators, resulting in a cost of $\mathcal{O}(N \cdot n)$ per generation. However, unlike the standard variant, this version re-applies K-Means clustering at every generation to reassign elephants to c clans.

Each K-Means iteration involves computing the Hamming distance between every elephant and all c centroids, costing $\mathcal{O}(N \cdot c \cdot n)$. Updating centroids requires $\mathcal{O}(N \cdot n)$ operations (to compute the majority bit per dimension across clusters). If we run the clustering for max_kmeans_iter iterations, the total cost of K-Means becomes $\mathcal{O}(max\_kmeans\_iter \cdot N \cdot c \cdot n)$ per generation.

Therefore, the total time complexity over all generations is:

$$\mathcal{O}\left(t_{\max} \cdot [N \cdot n + max\_kmeans\_iter \cdot N \cdot c \cdot n]\right) = \mathcal{O}\left(t_{\max} \cdot N \cdot n \cdot [1 + max\_kmeans\_iter \cdot c]\right)$$

Asymptotically, the complexity remains polynomial in population size N and chromosome length n, with an additional multiplicative factor from max_kmeans_iter and number of clans c.

5.2.2 Illustrative Example of Kmeans-Based EHO

To better understand the hybridization of the Kmeans-Based EHO algorithm, consider the following example representing a population of elephants

$$\{000, 100, 110, 011, 101, 111\}$$

with $k = 2$.
The steps of this clustering process are illustrated in the figure below 5.3:

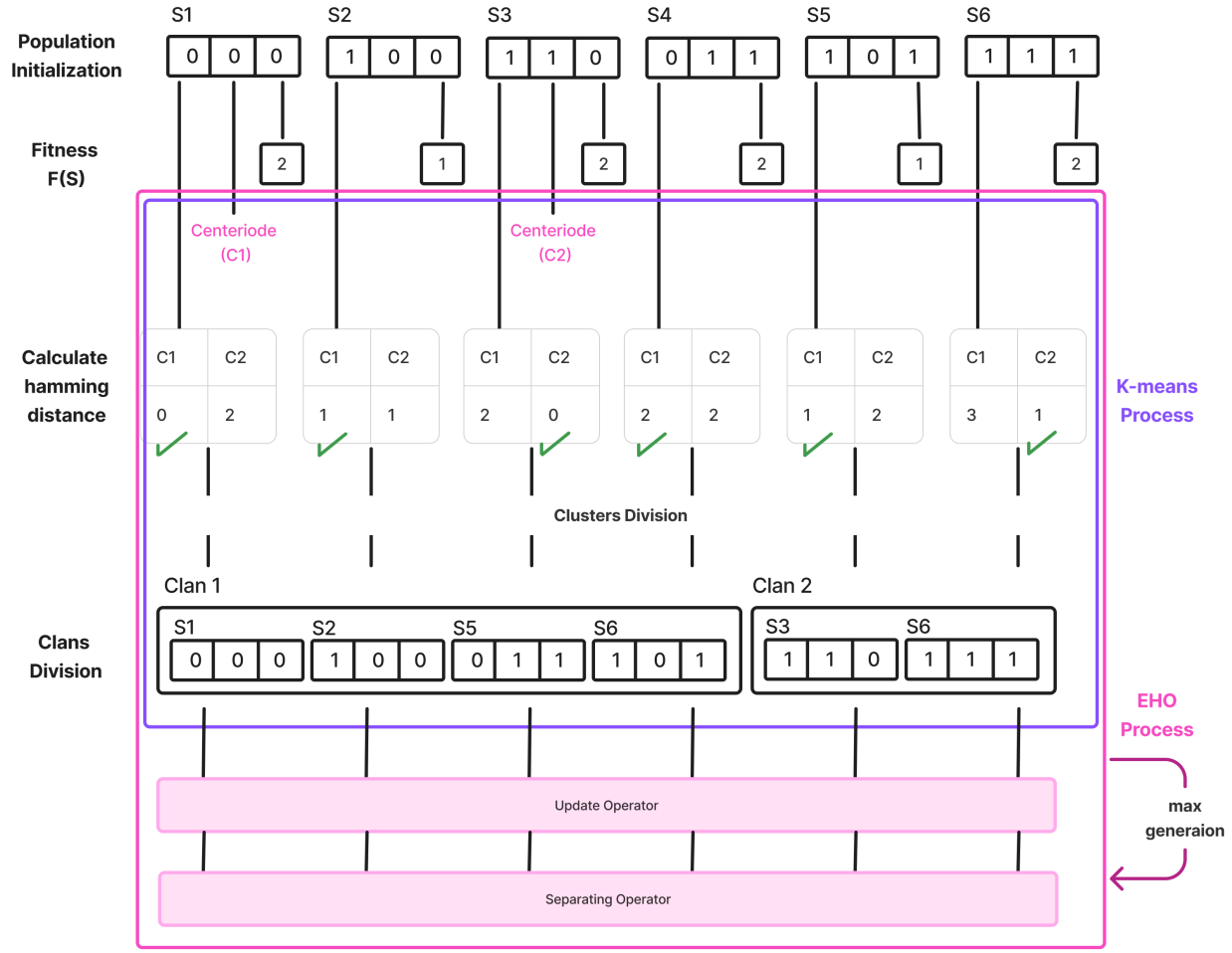

Figure 5.3: Illustrative Example of Kmeans-Based EHO

Note: In this example, the ``*Recalculate the centroid*'' step is not included; we assume that the clustering process is performed only once.

5.3 DBSCAN-Based EHO (EHO-DBSCAN)

The Elephant Herding Optimization (EHO) algorithm is inspired by the natural social behavior of elephant herds. In this metaheuristic, the population of solutions (elephants) is divided into several clans. Each clan evolves independently under the guidance of a leader (often the matriarch), while some elephants are periodically separated from their clans and replaced to maintain diversity.

On the other hand, DBSCAN is a powerful clustering algorithm that groups data points based on local density. rather than predefined cluster numbers. It identifies core points that have a dense neighborhood, border points on the edges of clusters, and outliers that do not belong to any cluster due to their sparse surroundings. This method is especially effective in detecting arbitrarily shaped clusters and handling noise within datasets.

This conceptual similarity between clan-based division of EHO and density-based clustering in DBSCAN inspired our hybridization approach. Moreover, the randomly or uniformly division of clans within the tradition method of EHO overlooks the underlying structure of the solution space and may not effectively utilize outlier solutions. These limitations led us to propose the integration of DBSCAN into the EHO framework to guide a more adaptive and informed division of clans, along with the effective handling and exploitation of outliers, thereby enhancing performance in solving the SAT problem.

5.3.1 Methodology of the Hybrid EHO-DBSCAN Approach

Since both the EHO and DBSCAN algorithms share a common principle of dividing the population into clans or clusters and handling outliers, careful consideration is given to effectively integrate their strengths. Obervation is that DBSCAN performs cluster division and removes outliers at the beginning before exploiting each cluster independently. Therefore, there is no need to modify or replace the update and separating operators in EHO, as the clustering is now more accurate and adaptive from the start.
The Elitism Strategy in EHO, which replaces the worst individuals in the entire population with the top-performing ones, is replaced with an outliers handling mechanism that better utilizes those isolated solutions, since both strategies share the goal of maintaining solution quality and diversity. Building on these observations, the core of the proposed hybridization lies in enhancing the clan division step, which is now driven by a DBSCAN-inspired clustering strategy. As shown in Figure 5.4, this integration focuses on two essential and complementary components DBSCAN-based clan division and outliers handling which together improve population structuring and preserve diversity from the outset.

The inspired DBSCAN clan division is presented along with an effective approach to outlier handling.

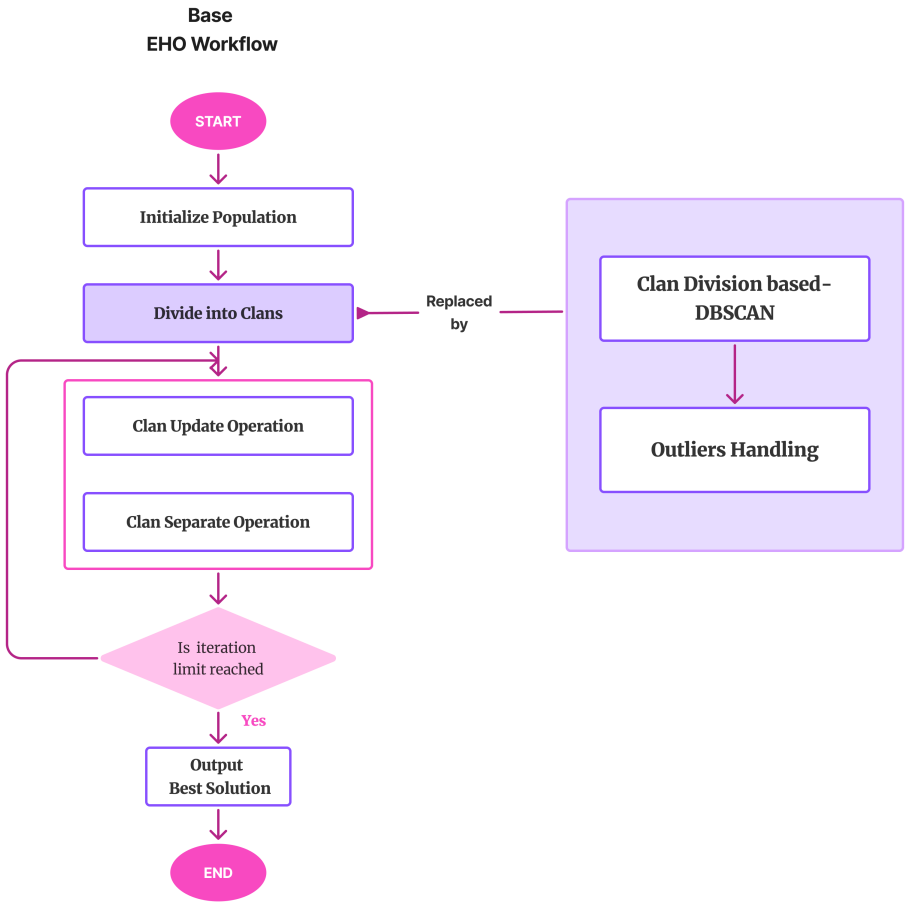

Figure 5.4: Hybridization of Clan Division in EHO Using DBSCAN and Outlier Handling

5.3.1.1 Clan Division Based on DBSCAN-Inspired Strategy

The first key component of the hybrid EHO-DBSCAN approach is the DBSCAN-based clan division, which replaces the conventional random or KMeans-based clustering methods used in EHO. This step ensures that individuals in the initial population are grouped into meaningful clans based on similarity and density in their structure—measured using the Hamming distance.

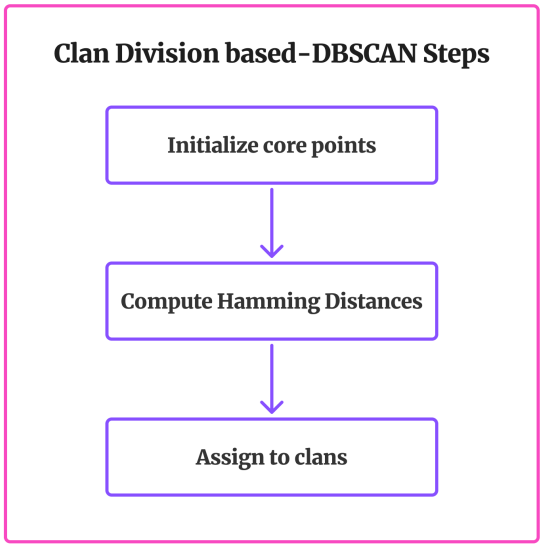

Figure 5.5: Clan Division based-DBSCAN Steps

Figure 5.5 illustrates the division process unfolds in three major steps:

1. **Initialization of Core Points**

   Core points are randomly selected from the population and serve as provisional clan centers. The number of core points is a tunable parameter that controls the number of formed clans, helping to flexibly structure the population, preserve diversity, and reduce the risk of premature convergence.

2. **Computation of Hamming Distances**

   Each individual in the population is then compared to the initialized core points using Hamming distance, a metric well-suited to binary representations like our problem (SAT). The Hamming distance between two binary vectors is the count of positions at which the bits differ.

3. **Clan Assignment**

   Instead of assigning individuals to the closest core point like in k-means, our approach uses a fixed distance threshold (ε) to determine if an individual is close enough to a core point to join its clan. This ε-based assignment creates more flexible and natural groupings, where individuals are clustered only if they share a sufficient level of similarity, enhancing intra-clan cohesion and supporting more focused exploration.

To clarify how the Hamming distance is computed and used for clan assignment in our DBSCAN-based strategy, consider the following example 5.3:

Example 5.3. *Consider two binary solutions:*

- $A = [1, 0, 1, 1, 0]$
- $B = [1, 1, 0, 1, 0]$

=> The Hamming distance (H) In this case:

$$H(A,B) = 2 \quad \textit{(positions 2 and 3 differ)}$$

=>To enable comparison with the ε threshold, a normalization of this distance is applied by the dimensionality of the solution (here number of variables $= 5$*):*

$$\textit{Normalized } H(A,B) = \frac{2}{5} = 0.4$$

If $\varepsilon = 0.5$ *=> H(A,B)*$= 0.4 < 0.5$*, meaning the second solution is considered sufficiently close to the first (core point) and can be assigned to the same clan.*

5.3.1.2 Outliers Handling and Integration

During the clan division process, outlier individuals are identified. Instead of discarding them, a mutation was applied to these outliers to enhance their potential and then redistribute them across clans. This ensures that all individuals remain active in the optimization process.
Figure 5.6 resumes the followed steps of outliers handling process.

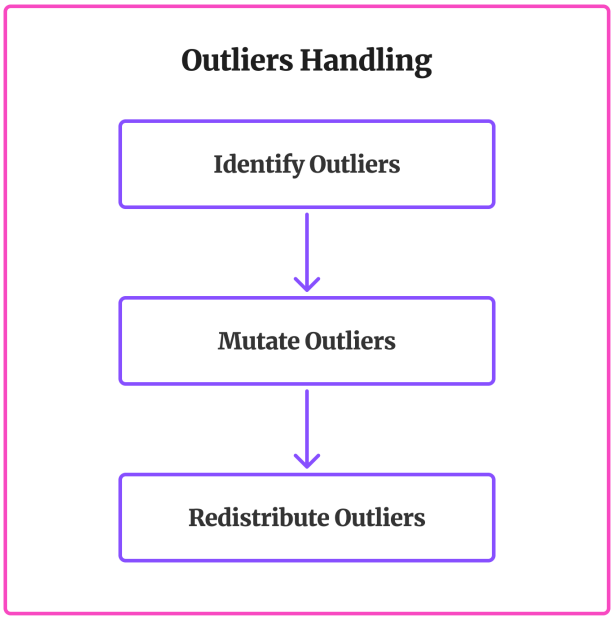

Figure 5.6: Outliers Handling Steps

The process follows three main steps:

1. **Identification of Outliers**

   After assigning solutions to clans based on similarity, each individual is evaluated to determine whether it is close enough to at least one core solution. This is done using a predefined threshold that measures how similar two solutions are. If a solution does not meet this threshold with any of the cores, it is considered an outlier and added to a separate list for special handling.

2. **Mutation of Outliers**

   After identifying outliers, slight random mutations is applied to their structure to increase diversity while preserving their core characteristics. This enhances their chances of integrating into existing clans or exploring new promising areas in the solution space.

3. **Redistribution of Mutated Outliers**

   After mutation, outliers are redistributed intelligently to maintain balance among clans. Filling smaller clans those with fewer than a minimum number of individuals by assigning them mutated outliers is prioritized. This helps ensure that all clans meet a baseline size for effective exploration. the remaining outliers are distributed randomly across the clans to preserve diversity.

The following example is proposed to simplify the understanding of the outliers handling process.:

Example 5.4. *Let's consider the following initial population of solutions:*

$$\{101010, 001110, 111000, 110011, 000111, 111111, 010101, 100100\}$$

and the initial parameters include:

$$\textit{Number of core points} = 2, \quad \textit{MinPts (minimum clan size)} = 4$$

Selected core points:

$$\{101010, 000111\}$$

The resulting clan division :

$$\begin{cases} Clan\ 1 : \{101010, 100100\} \\ Clan\ 2 : \{000111, 001110\} \\ Outliers : \{111000, 110011, 010101, 111111\} \end{cases}$$

After mutation of outliers:

$$\{111001, 110111, 010100, 111110\}$$

Redistribution:

$$\begin{cases} Clan\ 1\ (size < MinPts) \rightarrow \{111001, 110111\} \\ Clan\ 2\ (size < MinPts) \rightarrow \{010100, 111110\} \end{cases}$$

5.3.2 Illustrative Example of DBSCAN-Based EHO

To illustrate the DBSCAN-Based EHO algorithm, consider a population of 9 binary solutions of length 6:

$$S = \{S1=101010,\ S2=101110,\ S3=100010,\ S4=000111,\ S5=001111,\ S6=000011,\ S7=111000,\ S8=110000,\ S9=111100\}$$

The initial parameters are set as follows:

- number of core points = 2
- $\epsilon = 0.33$ (normalized Hamming distance (H))
- minPts = 4

All the process of clan division, outlier detection, mutation, and redistribution is illustrated in Figure (5.7).

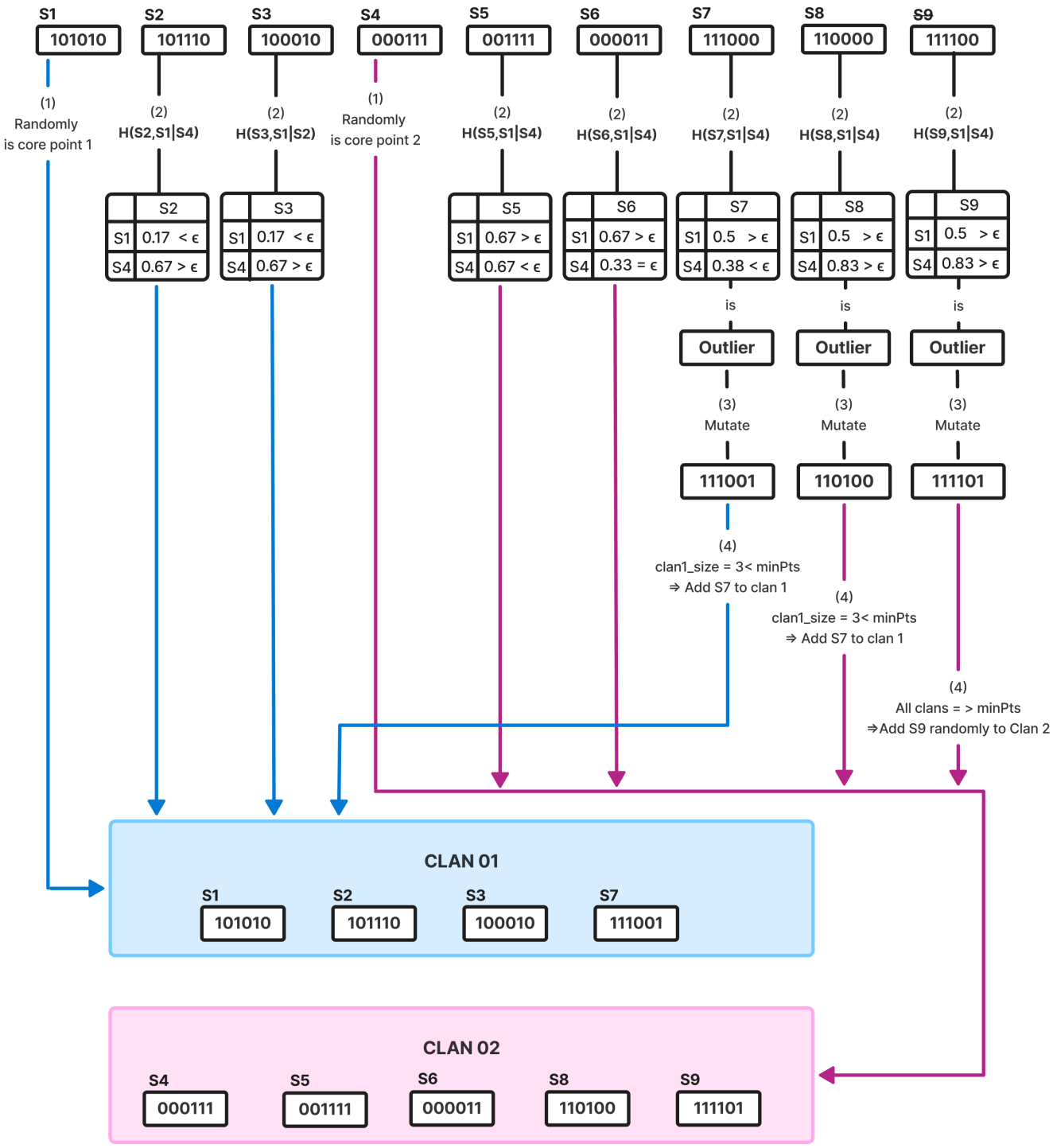

Figure 5.7: Illustrative Example of DBSCAN-Based EHO Process

The following pseudocode outlines the main steps of the hybrid EHO-DBSCAN algorithm (13);

Algorithm 13: Elephant Herding Optimization with DBSCAN-Based Clustering for SAT

Input : CNF clauses, population size N, max generations t_{max}, learning rates α, β, DBSCAN parameters ϵ, minPts, number of core points c_p

Output: Best solution x_{best}

Initialize population of N elephants (binary vectors of length n)
Evaluate fitness of all elephants (using Formula 1)
Apply **DBSCAN-based mechanism** in clan divison step
Sort population by descending fitness
Save top-k elephants as elites
$t \leftarrow 0$
while $t < t_{max}$ **do**
    for *each clan* $clan_i$ **do**
        Apply **Clan Update Operator** to $clan_i$
        Apply **Separation Operator** to $clan_i$
    Merge all elephants into a new population
    `/* Clan division using DBSCAN mechanism */`
    Copy the population: $P_{copy} \leftarrow P$
    Randomly select c_p core points from P_{copy}:
    Initialize empty clans: $clans \leftarrow \{[], [], \ldots, []\}$ (one for each core)
    Initialize empty list of outliers: $outliers \leftarrow \{\}$
    foreach *individual* $x \in P$ **do**
        $assigned \leftarrow false$
        foreach *core point* $core_j \in C$ **do**
            if *HammingDistance*$(x, core_j) \leq \epsilon$ **then**
                Add x to $clans[j]$
                $assigned \leftarrow true$
                break
        if *not* $assigned$ **then**
            Add x to $outliers$
    Sort $clans$ by descending size
    `/* Outlier handling */`
    Mutate outliers with small bit flip probability ($outliers_{mutated}$)
    $idx \leftarrow 0$; $num \leftarrow$ size of $outliers_{mutated}$
    foreach *clan* $clan_k$ *in* $clans$ **do**
        while *length*$(clan_k) < minPts$ *and* $idx < num$ **do**
            Add $outliers_{mutated}[idx]$ to $clan_k$
            $idx \leftarrow idx + 1$
    if $idx < num$ **then**
        for $i \leftarrow idx$ **to** $num - 1$ **do**
            Randomly select clan $clan_r$ and add $outliers_{mutated}[i]$ to it
    Return $clans$
    $t \leftarrow t + 1$
return x_{best}

Complexity: In this variant, the algorithm processes all N elephants at each generation using both the clan update and separation operators, which together require $\mathcal{O}(N \cdot n)$ operations per generation, where n is the chromosome length.

The distinctive feature of this variant is the DBSCAN-inspired clan division. At each generation:

- c_p core points are selected randomly from the population: $\mathcal{O}(c_p)$
- Each of the N individuals is compared to all c_p core points using Hamming distance: $\mathcal{O}(N \cdot c_p \cdot n)$
- Sorting clans by size: $\mathcal{O}(c_p \cdot \log c_p)$
- Outlier handling and mutation (with constant probability bit flips): $\mathcal{O}(N \cdot n)$
- Redistributing outliers among clans: $\mathcal{O}(N)$

Combining these, the total cost per generation becomes:

$$\mathcal{O}\left(N \cdot n + N \cdot c_p \cdot n + c_p \cdot \log c_p + N\right) = \mathcal{O}(N \cdot c_p \cdot n)$$

since $N \cdot c_p \cdot n$ dominates for large N and n.

Over all t_{max} generations, the total time complexity is:

$$\mathcal{O}(t_{\text{max}} \cdot N \cdot c_p \cdot n)$$

5.4 Conclusion

Throughout this chapter, the limitations of the Elephant Herding Optimization algorithm were identified and addressed through the proposition of a hybridization with data mining and especially clustering techniques.

The K-means clustering, was first proposed for hybridization, with the aim of forming semantically meaningful clans facilitating efficient local exploitation within the clans while promoting global diversity across them.

As second proposition, DBSCAN algorithm is proposed for both clan division and outlier detection. On one hand, the clans division is carried out using DBSCAN inspired approach. On the other hand, outliers are identified and slightly mutated to intelligently enhance population diversity.

The following chapter is dedicated to experimental validation of the proposed approaches.

CHAPTER 6

EXPERIMENTS AND RESULTS

6.1 Introduction

This chapter presents a comprehensive experimental evaluation of the proposed EHO-based SAT solver variants.
After describing the test environment, selecting benchmarks, and introducing evaluation metrics, the performance of the base EHO and its improved versions are assessed. All the proposed improvement strategies of EHO, which aim to reduce stagnation and enhance search efficiency including the mutation operator, re-division strategy, and their combination, are first experimentally validated. The second stage of experimental process focuses on examining hybrid approaches that integrate data mining techniques K-Means and DBSCAN clustering into the base EHO model to achieve better exploration and exploitation of the solution space.

6.2 Experimental Setup

To achieve and validate this project, a precise experimental setup is required, including the test environment, benchmark datasets, and evaluation metrics. This section presents these components to ensure a thorough and objective performance analysis.

6.2.1 Test Environment

To ensure a comprehensive evaluation of the proposed methods, the experiments were conducted on two different computing environments which configuration are resumed in Table 6.1.

First Machine	**Second Machine**
• Processor: AMD Ryzen 5 4500U with Radeon Graphics @ 2.38 GHz • RAM: 32 GB • Disk: 512 GB SSD • Operating System: Windows 11 Pro	• Processor: AMD Ryzen 5 PRO 5650U with Radeon Graphics @ 2.30 GHz • RAM: 16 GB • Disk: 286 GB SSD • Operating System: Windows 11 Pro

Table 6.1: Hardware Configuration of Used Machines

For the software environment, on a set of essential tools and platforms was relied upon :

- **Programming Language**

  The chosen programming language is Python due to its simplicity, readability, and the richness of its scientific ecosystem. These characteristics made it particularly suitable for implementing our custom metaheuristic algorithms from scratch.

- **Core Libraries**

  The implementation was based on core Python libraries such as `random`, `numpy`, `collections`, and `time`. These libraries support key operations including stochastic behavior, numerical computations, data structure management, and runtime tracking.

- **Development Environment**

  Development and debugging were conducted using Visual Studio Code, while Google Colab provided a convenient cloud-based platform for rapid testing and experimentation.

- **Data Analysis and Tabulation:**

  Microsoft Excel was used for organizing experimental results, performing tabular analysis, and creating summary charts to support interpretation and reporting.

- **Redaction and Documentation**

  Overleaf (LaTeX) was employed for typesetting and writing the thesis, ensuring a high-quality academic format. Additionally, Notion was used to organize research notes and manage project planning in a collaborative and structured manner.

- **Diagrams and Illustrations**

  Figma was chosen for designing workflow diagrams and creating visual representations of the methodology, which significantly aided in clarifying and communicating complex processes throughout the project.

6.2.2 Benchmark Description

To evaluate the performance and robustness of the proposed EHO-based SAT solver and its improvements, two types of SAT benchmarks were considered:

➤ **Small benchmark:**

To examine the behavior of all the proposed improvement and approaches on small-scale problems generated within a constrained search space, multiple tests were conducted on the Uniform Random-3-SAT benchmark specifically the "uf200-860". Although this benchmark, originally constructed for decision problems, however, it can be used within an optimization context such as Max-SAT.

➤ **Medium and Large benchmark:**

A set of instances from the Bounded Model Checking BMC, which includes complex SAT encoding derived from hardware verification were selected as medium and large benchmarks.

The set of benchmarks contain 13 instances which description is presented in Table 6.2.
Due to high computational cost, only three representative instances (bmc-ibm-1 ,bmc-ibm-2,bmc-ibm-7) were tested.

File name	Number of variables	Number of clauses
bmc-ibm-1.cnf	9686	55870
bmc-ibm-2.cnf	2810	11683
bmc-ibm-3.cnf	14930	72106
bmc-ibm-4.cnf	28161	139716
bmc-ibm-5.cnf	9396	41207
bmc-ibm-6.cnf	51639	368352
bmc-ibm-7.cnf	8710	39774
bmc-ibm-10.cnf	59056	323700
bmc-ibm-11.cnf	32109	150027
bmc-ibm-12.cnf	39598	194778
bmc-ibm-13.cnf	13215	65728
bmc-galileo-8.cnf	58074	294821
bmc-galileo-9.cnf	63624	326999

Table 6.2: SATLIB benchmark instance characteristics

As discussed in Section 4.2.1, a lightweight preprocessing step were applied to the selected SAT benchmarks to reduce redundancy and enhance solving efficiency. Although SAT instances are generally simple, our benchmarks contained some tautological, duplicate, or included clauses. Table 6.3 summarizes the impact of this preprocessing on the three selected BMC instances by showing the reduction in clauses count, including the number of tautologies, duplicate clauses, and subsumed clauses.

Instance	Clauses Before	Clauses After	Removed Clauses	Tautologies	Duplicate clauses	Removed clauses
bmc-ibm-1	55870	54682	1188	15	193	980
bmc-ibm-2	11683	10561	1122	16	301	805
bmc-ibm-7	39774	37388	2386	28	170	2188

Table 6.3: Impact of preprocessing on BMC benchmark instances

To validate the preprocessing step, a random solution was generated and evaluated on both the original and preprocessed versions of an instance.
Figures 6.1 and 6.2 present a comparative study of the satisfiability rate and execution time before and after preprocessing, averaged over 10 independent runs.

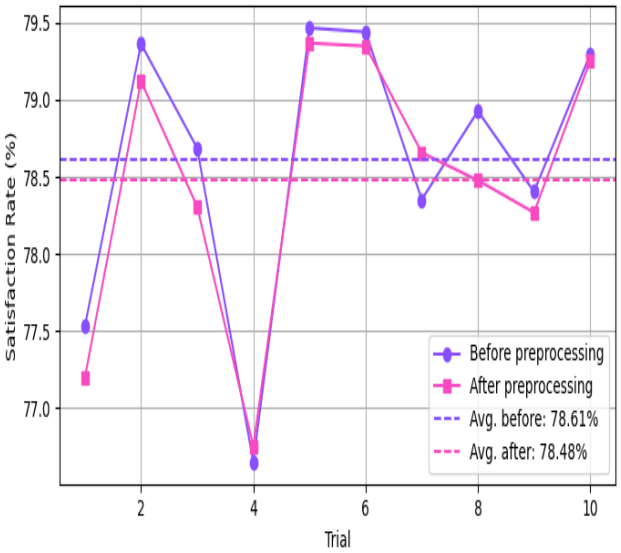

Figure 6.1: Satisfiability rate before and after preprocessing over 10 runs

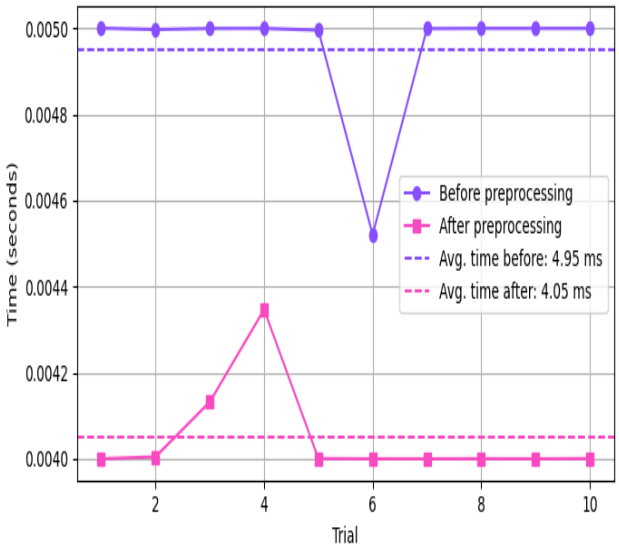

Figure 6.2: Execution time before and after preprocessing over 10 runs

From the figures [6.1, 6.2], satisfiability rates appear slightly higher before preprocessing, which is explained by the removal of included clauses.
For instance, removing a clause like $[X_1 \vee X_2]$ when $[X_1]$ exists reduces redundancy but also lowers the chance of satisfying all clauses with random solutions.
However, execution time is clearly lower after preprocessing, making the solving process faster and more efficient.
This confirms that the preprocessing step is valuable, especially for large benchmark instances where time is a key factor.

6.2.3 Evaluation Metrics

According to the context of metaheuristics, the main four performance metrics adapted to the EHO algorithm to solve SAT problems are presented as follows:

✓ **Exploration and Exploitation**

This reflects the algorithm's ability to balance between discovering new areas in the search space (exploration) and refining existing good solutions (exploitation). Population diversity and solution variations across generations Were monitored.

✓ **Effectiveness**

Represents the ability of the algorithm to find valid or optimal solutions. It is estimated via the success rate (percentage of satisfiable results) and the best fitness achieved.

✓ **Efficiency**

Evaluates the computational cost of the algorithm to reach a solution. It is estimated based on execution time and the number of generations required to converge.

✓ **Stagnation**

In SAT problems, stagnation happens when the algorithm stops improving over several generations—the best fitness value stays the same for multiple iterations. This usually means the search is stuck in a local optimum or has lost its ability to explore.

To measure this, the Stagnation Rate is proposed and is, based on observed patterns in the sequence of best fitness values: (1) many values remain unchanged across iterations, and (2) some values repeat frequently.

$$\text{Stagnation Rate (SR)} = \frac{\text{max\_rep} + \text{rep\_value\_count}}{\text{total\_gen}} \tag{6.1}$$

Where:

- `max_rep` is the maximum number of times a single fitness value occurs,
- `rep_value_count` is the number of fitness values that occur more than once,
- `total_gen` is the total number of generations.

This formula 6.1 provides a simple yet informative measure of how much the algorithm stagnates during its run.

Note: These metrics were either directly computed or inferred through observations, given their interdependence.

6.3 Results and Analysis

This section presents a detailed evaluation of the adapted EHO-based SAT solver and its various improved versions.
The performance analysis is first conducted on the basic adapted EHO and its improvement including mutation, re-division, and their combinations , followed by that of the proposed hybrid models using K-Means and DBSCAN clustering. The results provide insights into the effectiveness and improvements achieved across different benchmarks.

6.3.1 Performance of the Adapted EHO-Based SAT Solver

To evaluate the efficiency of adapted EHO-based solver for the SAT problem, the process began with testing its baseline behavior and identifying potential limitations.

The EHO algorithm involves several key parameters that govern its search dynamics. In this initial phase, the focus is, first, placed on two core parameters: α and β, which are used in the update operator to control the position updates of elephants.

From the results presented in Table 6.3, observations show that the tuning of α and β significantly influences the performance of the solver. No single combination consistently outperforms others across all benchmark instances, underscoring the sensitivity of the algorithm to these parameters.

Lower values generally promote more stable convergence by maintaining a balance between exploration and exploitation, whereas higher values tend to increase stagnation and reduce reliability. Overall, a well-balanced setting of α and β leads to more effective performance by allowing the algorithm to explore the solution space efficiently while avoiding premature convergence.

Parameter's		UF200-02 (200 variable - 860 clausse)				bmc-ibm-2 (2810 variable - 11683 clausse)			
α	β	Best Fitness	Satisfiability Rate (%)	Stagnation Rate (%)	Balance Rate	Best Fitness	Satisfiability Rate (%)	Stagnation Rate (%)	Balance Rate
0.1	0.1	811 (2)	93.56 (2)	46.34 (2)	0.894 (1)	8881 (3)	84.09 (3)	37.62 (3)	0.7055 (2)
	0.3	803 (7)	92.95 (8)	**46.14 (1)**	0.6703(2)	8837 (7)	83.86 (7)	**29.7 (1)**	0.728 (1)
	0.5	**798 (8)**	**92.49 (9)**	48.12 (3)	0.4759 (4)	**8795 (9)**	**83.23 (9)**	35.64 (2)	0.4557 (8)
0.3	0.1	**813 (1)**	**93.84 (1)**	86.34 (8)	0.5098 (3)	8862 (6)	83.91 (6)	70.30 (6)	0.4832 (7)
	0.3	809 (3)	93.47 (4)	78.81 (4)	0.4648 (6)	**8911** (1)	**84.38** (1)	68.32 (4)	0.5874 (3)
	0.5	807 (4)	93.3 (6)	84.95 (6)	0.3266 (8)	8877 (4)	84.05 (4)	71.29 (7)	0.5232 (6)
0.5	0.1	804 (6)	93.14 (7)	**87.13 (9)**	0.2407 (9)	8829 (8)	83.6 (8)	**83.17 (9)**	0.2684 (9)
	0.3	811 (2)	93.53 (3)	80.59 (5)	0.4652 (5)	8864 (5)	83.93 (3)	63.37 (5)	0.5635 (4)
	0.5	806 (5)	93.4 (5)	85.74 (7)	0.354 (7)	8904 (2)	84.31 (2)	74.26 (8)	0.5527 (5)

Figure 6.3: Impact of α and β Parameter Tuning on Solver Performance

The analysis in Table 6.3 was conducted with fixed parameters: number of clans = 20, population size = 300, max generations = 100, and elitism k = 5, while varying α and β.

After analyzing the individual impact of the parameters α and β in Table 6.3, the attention is now directed toward two additional key parameters which are the population size and the clan number.

Table 6.4 exhibits the influence of dynamic configurations of population size and number of clans, while maintaining adaptive (dynamic) settings for α and β. The aim is to understand how varying the core structural components of the EHO algorithm affects its performance across several benchmark instances, including small-scale SAT problems (UF200) and medium-scale instances (BMC).

The Key observations and findings from Table 6.4:

- **Smaller number of clans** leads to improved performance, as it strengthens intra-clan cooperation and enhances convergence toward optimal solutions.
- **Larger population size** consistently improve the Satisfiability Rate , although it comes with a noticeable increase in computational time.
- **The best overall performance** is achieved with a large population (300) and a small number of clans (5), indicating an effective balance between exploration and exploitation.
- As population size increases, execution time increases as well. This highlights a clear tradeoff between solution quality and time complexity.
- While α and β showed inconsistent effects in static tuning (Table 6.3), their dynamic adjustment in this phase improves stability and search efficiency, especially when paired with larger populations. This confirms their complementary role in refining performance.

From this analysis,it is concluded that carefully controlling the core parameters of the EHO algorithm such as: population size, number of clans, and the dynamic adjustment of α and β has a substantial impact on its ability to reach optimal solutions efficiently.
The experimental results clearly show that larger population sizes combined with fewer clans lead to superior performance, while adaptive tuning of α and β further supports convergence. These insights highlight the necessity of parameter sensitivity analysis when applying EHO to complex combinatorial problems such as SAT.

			Small										Medium					
			uf200-01			uf200-010			uf200-050			uf200-095	bmc-ibm-1			bmc-ibm-2		
Population Size	Clan Size	Alpha	Beta	Best Fitness	SR (%)	Time (s)	Best Fitness	SR (%)	Time (s)	Best Fitness	SR (%)	Time (s)	Best Fitness	SR (%)	Time (s)	Score	SR (%)	Time (s)	Score	SR (%)	Time (s)
100	5	0.1	0.1	813	94.53	58.55	812	94.42	59.31	817	95	59.47	809	94.07	61.68	44703	81.77	3602.65	8939	84.63	649.4
			0.5	805	93.6	58.68	807	93.84	59.28	801	93.14	59.53	803	93.37	61.94	44107	80.68	3583.58	8855	83.86	648.77
		0.5	0.1	814	94.65	59.18	810	94.19	58.6	804	93.49	60.44	807	93.84	62.34	44189	80.83	3591.47	8812	83.44	651.05
			0.5	812	94.42	64.36	799	92.91	58.97	803	93.37	60.18	809	94.07	62.71	44208	80.87	3582.34	8861	83.91	648.97
	10	0.1	0.1	807	93.84	63.97	800	93.02	59.88	801	93.14	59.93	803	93.37	63.36	44200	80.86	3854.32	8878	84.08	668.99
			0.5	791	91.98	64.16	792	92.09	58.49	791	91.98	60.17	791	91.98	62.8	43785	80.08	3871.98	8700	82.38	667.9
		0.5	0.1	804	93.49	56.76	799	92.91	62.72	801	93.14	67.92	801	93.14	62.91	44283	80.99	4262.56	8811	83.43	672.99
			0.5	804	93.49	56.56	806	93.72	62.2	805	93.6	67.71	805	93.6	62.99	44218	80.9	3853.04	8845	83.72	671.65
	20	0.1	0.1	796	92.56	56.71	796	92.56	63.09	796	92.56	69.19	799	92.91	62.98	43958	80.39	4244.53	8741	82.76	682.06
			0.5	795	92.44	58.21	789	91.74	63.75	794	92.33	69.18	793	92.21	68.09	43628	79.77	4236.51	8687	82.26	679.98
		0.5	0.1	803	93.37	58.85	798	92.79	63.96	802	93.26	69.78	803	93.37	68.58	44130	80.73	3073.18	8859	83.89	682.44
			0.5	804	93.49	59	798	92.79	63.94	802	93.26	68.4	800	93.02	68.85	44005	80.48	3074.56	8768	83.01	682.78
300	5	0.1	0.1	826	96.05	184.76	825	95.93	208.8	829	96.4	202.63	827	96.16	199.14	45643	83.47	12089.69	9123	86.36	2243.81
			0.5	831	96.63	186.17	830	96.51	208.88	831	96.63	201.89	829	96.4	197.45	45303	82.86	12097.42	9095	86.09	2237.43
		0.5	0.1	814	94.65	185.77	818	95.12	209.67	813	94.53	203.15	815	94.77	199.34	44659	81.68	13583.64	8949	84.73	2240.49
			0.5	818	95.12	194.43	818	95.12	206.84	816	94.88	205.87	817	95	227.88	44651	81.66	13550.95	8952	84.76	2240.99
	10	0.1	0.1	816	94.88	196.13	821	95.47	208.71	822	95.58	203.7	823	95.7	226.99	45009	82.34	12140.35	9023	85.41	2101.63
			0.5	815	94.77	195.33	819	95.23	205.74	814	94.65	205.57	816	94.88	226.58	44591	81.55	13575.61	8990	85.11	2109.35
		0.5	0.1	814	94.65	192.48	809	94.07	208.73	808	93.95	199.74	809	94.07	207.87	44428	81.26	11712.46	8942	84.69	2104.85
			0.5	812	94.42	191.98	810	94.19	206.35	816	94.88	200.24	809	94.07	206.47	44423	81.25	11723.16	9014	85.29	2104.59
	20	0.1	0.1	810	94.19	192.34	811	94.3	207.08	816	94.88	201.39	813	94.53	208.18	44440	81.28	11761.47	8903	84.31	2123.77
			0.5	804	93.49	194.98	800	93.02	209.24	801	93.14	213.39	802	93.26	231	44087	80.64	12615	8793	83.25	2123.8
		0.5	0.1	807	93.84	196.15	810	94.19	209.39	803	93.37	212.41	802	93.26	231.88	44626	81.63	12649.27	8843	83.67	2130.42
			0.5	812	94.42	195.48	803	93.37	206.48	802	93.26	213.35	805	93.6	231.56	44551	81.48	12654.11	8891	84.16	2129.41

Figure 6.4: General Turing Parameters Result Table

The analysis in Table 6.4 was conducted with fixed parameters: max generations=100, elitism size =5 (Top-k elitism)

Recommended Parameter Setting: The most effective configuration observed includes a population size of 300, 5–10 clans, $\alpha \approx 0.1$, and β between 0.1 and 0.5. This setup will be used in subsequent tests. However, results remain context-dependent, as no single combination guarantees optimal performance across all instances.

After several tests of the base EHO algorithm, a recurring issue of stagnation or near-stagnation were observed (Table 6.3). As a remainder, stagnation refers to situations where the fitness value remains unchanged or progresses extremely slowly over a large number of generations.
To illustrate this phenomenon, Figure 6.5 presents the fitness progression on a given benchmark instance, where prolonged phases of stagnation can be clearly identified.

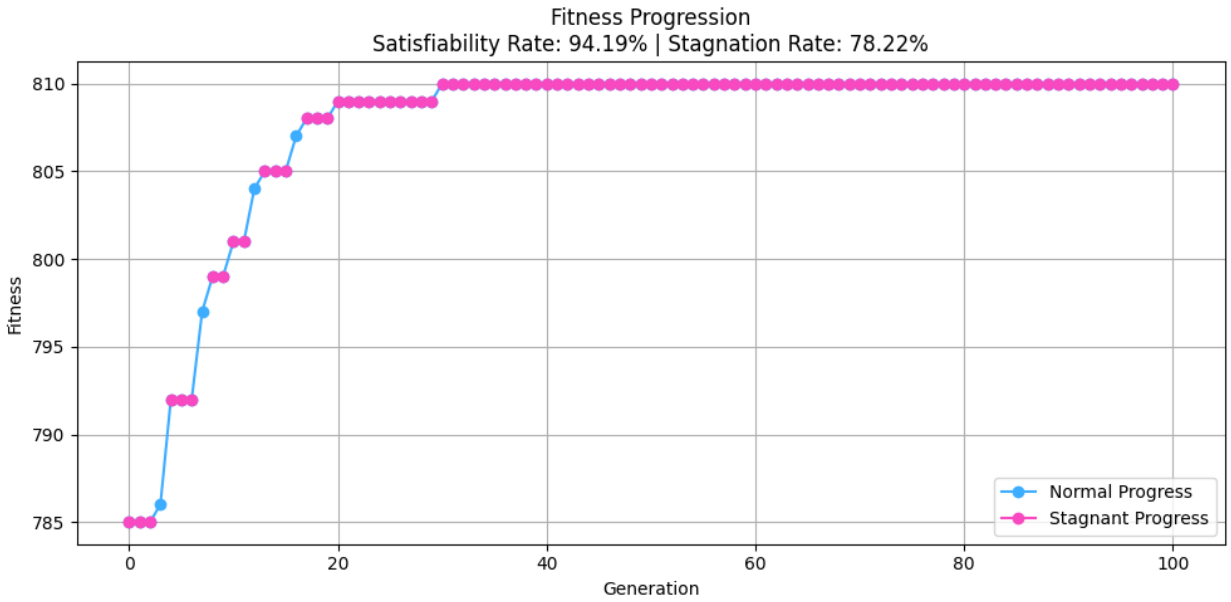

Figure 6.5: Fitness progression showing stagnation behavior

As observed, previously, in Table 6.3, the choice of parameters particularly α and β plays a crucial role in stagnation behavior: their valuesdetermine whether they exacerbate or reduce stagnation.
By measuring the stagnation rate across small and medium benchmark instances, it was further observed (figure 6.6) that stagnation or near stagnation tends to decrease as the benchmark size increases likely due to greater diversity and a broader search space.

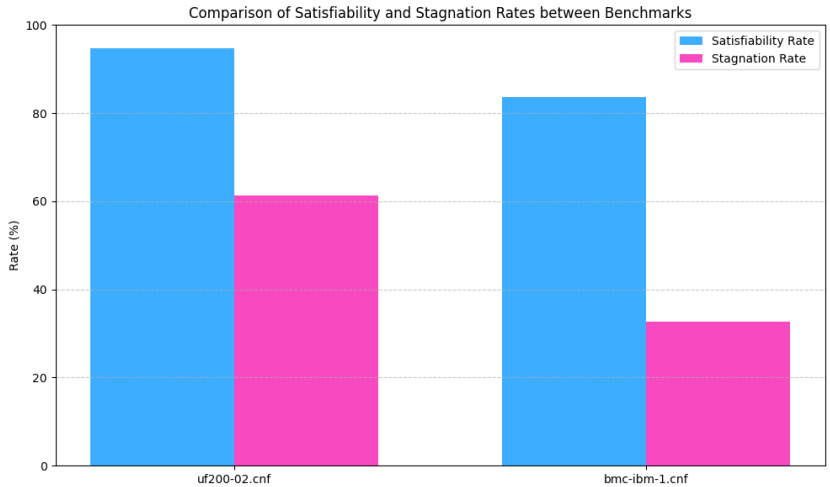

Figure 6.6: Comparison of satisfiability and stagnation rates in small and medium benchmarks

Finally, to better illustrate the efficiency of the adapted EHO-based SAT solver, additional experiments were conducted with 200 iterations. Table 6.4presents a summary of the solver's performance across various benchmark instances.

Benchmark Size	Instances	Attempts Number	Highest Fitness	Average Satisfiability Rate	Average Stagnation Rate	Average Execution Time
Small	uf20-02.cnf	5	823	95.37%	67.26%	125.28 s
	uf20-050.cnf	5	924	95.42%	69.95%	135.64 s
	uf20-095.cnf	5	820	95.02%	68.06%	123.74 s
Medium	bmc-ibm-2.cnf	1	9037	85.57%	36.82%	1701.8054 s
	bmc-ibm-7.cnf	1	31023	82.98%	29.85%	5116.6968 s
	bmc-ibm-1.cnf	1	45138	82.55%	24.38%	8706.0522 s

Table 6.4: Performance metrics of benchmark instances

6.3.2 Evaluation of the Improved Base EHO Model

This section evaluates the improvements made to the base EHO model through various strategies such as mutation, re-division, and their combination. This analysis explores how the proposed enhancements address issues like stagnation and improve overall performance on SAT benchmarks.

6.3.2.1 Evaluation of EHO with Mutation Operator

As previously discussed, the mutation operator was introduced into the EHO algorithm to enhance diversity and escape stagnation. This operator randomly perturbs certain elephant positions during the iteration process, thereby increasing exploration in the search space.
Here, the impact of the mutation rate introduced as a new parameter on the algorithm's behavior is analyzed. And specifically, the influence of the variation of this rate on satisfiability, stagnation, and execution time across benchmark instances. Table 6.5 summarizes the results obtained by applying different mutation rates on selected SAT problems under fixed initial parameters.

Instance	Mutation Rate	Satisfiability Rate (%)	Stagnation Rate (%)	Execution Time (s)
`uf200-050.cnf`	0.01	96.05 ↓	31.68 ↑	123.7624
	0.05	94.65 ↓	38.61 ↑	127.9361
	0.1	93.84 ↓	48.5 ↑	127.6842
	0.2	93.02 ↓	80.39 ↑	127.5776
	0.3	92.79 ↓	92.08 ↑	128.8517
	0.5	92.09 ↓	97.03 ↑	129.7167
`bmc-ibm2.cnf`	0.01	85.76 ↓	27.72 ↑	1219.5584
	0.05	84.88 ↓	51.49 ↑	956.8103
	0.1	84.17 ↓	62.38 ↑	1012.9934
	0.2	83.97 ↓	72.28 ↑	1296.1470
	0.3	82.91 ↓	87.13 ↑	1526.7370
	0.5	82.37 ↓	88.37 ↑	1533.6174

Table 6.5: Impact of Mutation Rate on Satisfiability, Stagnation, and Execution Time

The Evaluation in Table 6.5 conducted over 100 iterations with n_clans=10, population_size=300, α=0.1, β=0.3, elitism_k=4.

It is observed, from the analysis of Table 6.5, that a mutation rate of 0.01 yields the best balance, achieving high satisfiability and low stagnation. This rate, meaning only 1% of elephant positions are perturbed per iteration, ensures sufficient diversity without disrupting convergence. Higher rates degrade performance by introducing excessive randomness.

After selecting the optimal parameter combination and the best mutation rate from the previous section, the conducted evaluation of performance of the improved algorithm over 200 iterations is summarized in Table 6.6.

Benchmark Size	**Instances**	**Attempts Number**	**Highest Fitness**	**Average Satisfiability Rate**	**Average Stagnation Rate**	**Average Execution Time**
Small	uf20-02.cnf	5	836	96.99%	27.56%	244.33 s
	uf20-050.cnf	5	838	97.05%	30.75%	124.39 s
	uf20-095.cnf	5	836	96.91%	30.15%	116.32 s
Medium	bmc-ibm-2.cnf	1	9206	87.15%	25.37%	1398.1492 s
	bmc-ibm-7.cnf	1	31490	84.22%	25.87%	4943.6126 s
	bmc-ibm-1.cnf	1	45714	83.60%	32.34%	7626.3785 s

Table 6.6: Performance Evaluation of the Improved EHO with Mutation

The Evaluation in Table 6.6 conducted over 200 iterations with n_clans=10, population_size=300, alpha=0.1, beta=0.3, elitism_k=4, stagnation_threshold=3, mutation_rate=0.01.

Note: The stagnation threshold refers to the number of consecutive iterations during which the fitness value does not improve. It is used to detect when the search process is no longer making progress. The value of this threshold is chosen according to the benchmark size: for large benchmarks, stagnation tends to occur less frequently, a smaller threshold value is then used to detect stagnation earlier and maintain search efficiency.

6.3.2.2 Evaluation of EHO with Re-Division Strategy

Although the mutation operator is effective in many situations adding variation and helping the search escape local optima, it can sometimes fall short. In such cases, despite multiple changes, the algorithm remains trapped in the same region of the search space, unable to discover better solutions.

To better address stagnation, an alternative approach, namely re-division strategy is proposed.and is executed when a predefined stagnation threshold (X Stagnation) is exceeded. At that point, the algorithm triggers a full re-division of the population into new clans. This process redistributes candidate solutions across the search space, encouraging exploration of fresh and unexplored areas. It effectively resets the search and boosts the chances of finding improved results.

The table (6.7) shows the performance of EHO using the Re-Division strategy.

Benchmark Size	Instances	Attempts Number	Highest Fitness	Average Satisfiability Rate	Average Stagnation Rate	Average Execution Time
Small	uf20-02.cnf	5	841	97.79%	49.75%	287.7 s
	uf20-050.cnf	5	837	97.33%	56.72%	289.11 s
	uf20-095.cnf	5	839	97.56%	43.28%	349.02 s
Medium	bmc-ibm-2.cnf	1	9393	88.94%	27.86%	1677.93 s
	bmc-ibm-7.cnf	1	32113	85.89%	25.37%	7388.0961 s
	bmc-ibm-1.cnf	1	46925	85.81%	27.36%	8867.4629 s

Table 6.7: Performance Evaluation of the Improved EHO with Re-Division

Tests were run over 200 iterations with parameters: n_clans=10, population_size=300, α=0.1, β=0.3, elitism_k=4, and stagnation_threshold=3. These settings match those used in earlier and upcoming experiments for consistency.

6.3.2.3 Evaluation of Combined Mutation and Re-Division

As explained in section (4.3.3), mutation is applied when strict stagnation occurs. Additionally, re-division of clans is triggered when the fitness improvement over a window of X generations is below a threshold K, indicating slow progress. This combined strategy helps maintain diversity and escape local optima. The test results of this combined algorithm are summarized in Table 6.8.

Benchmark Size	Instances	Attempts Number	Highest Fitness	Average Satisfiability Rate	Average Stagnation Rate	Average Execution Time
Small	uf20-02.cnf	5	839	97.56%	20.89%	305.06 s
	uf20-050.cnf	5	840	97.67%	10.94%	307.93 s
	uf20-095.cnf	5	842	97.91%	12.94%	303.75 s
Medium	bmc-ibm-2.cnf	1	9370	88.72%	8.46%	2273.524 s
	bmc-ibm-7.cnf	1	32181	86.07%	10.07%	4963.8787 s
	bmc-ibm-1.cnf	1	46799	85.58%	10.45%	10006.7322 s

Table 6.8: Performance Evaluation of the Improved EHO with Combined mutation and Re-Division

6.3.2.4 Comparative Analysis of Improved Variants

In this section, a comparative analysis of the base EHO algorithm and its proposed improvements designed to mitigate the stagnation problem is performed, to observe whether these enhancements lead to better performance and effectively resolve the stagnation issue.

Figure (6.7) illustrates the evolution of the best fitness over 200 iterations for the base EHO and its improved variants tested on the medium benchmark. This curve allows us to visually assess how each variant progresses toward optimal solutions.

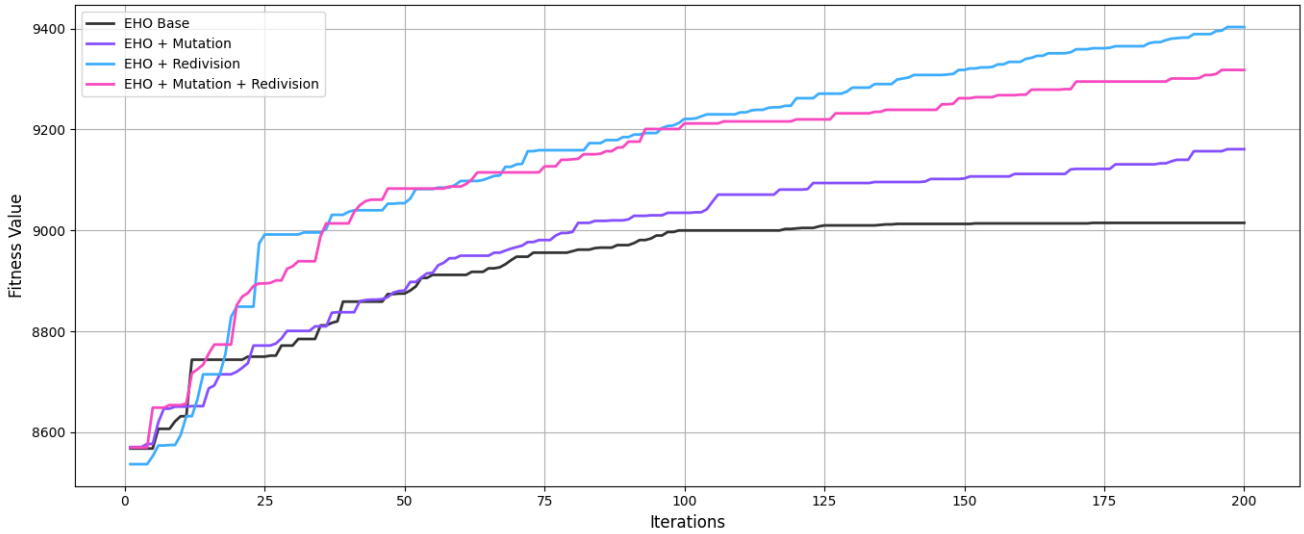

Figure 6.7: Fitness Convergence Comparison of EHO Variants

Based on Tables [6.4 , 6.6 , 6.7 , 6.8], which summarize the performance evaluation of the base EHO and its improved variants, a comparative study focusing on key performance metrics is conducted. Specifically, Figures (6.8) and (6.9) present the satisfiability instances, allowing us to observe how effectively each variant solves the SAT instances. To assess the ability of each approach to overcome stagnation, Figure (6.10) shows the stagnation rates across instances. Finally, execution time is compared to evaluate the computational cost introduced by the proposed (figure 6.11).

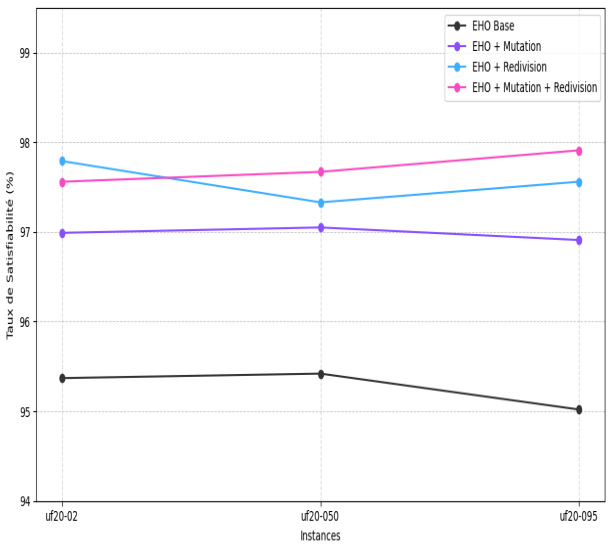

Figure 6.8: Comparison of satisfiability rates on small benchmark instances

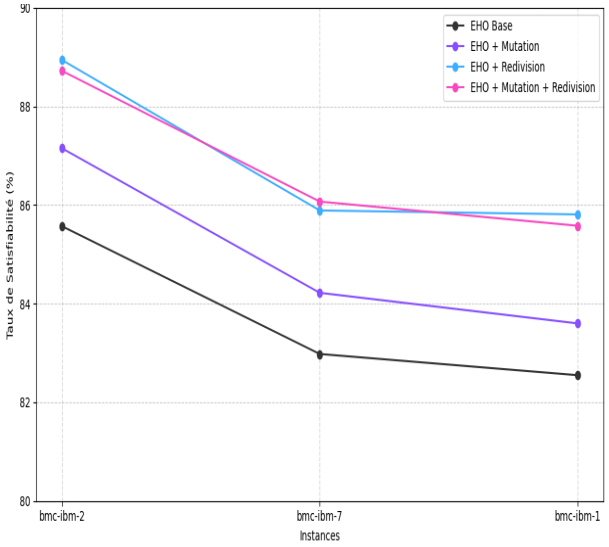

Figure 6.9: Comparison of satisfiability rates on medium benchmark instances

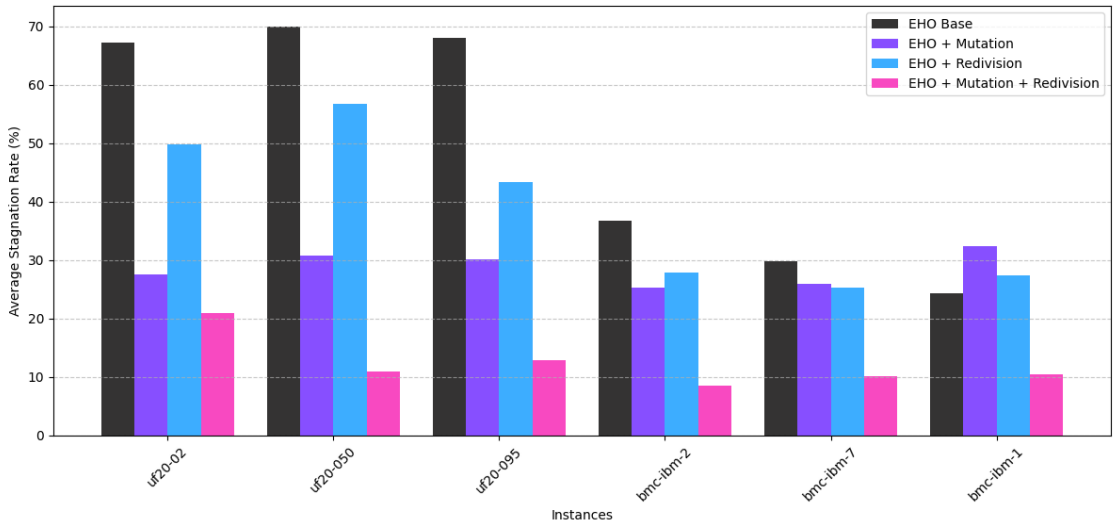

Figure 6.10: Average stagnation rate per instance across EHO algorithm variants

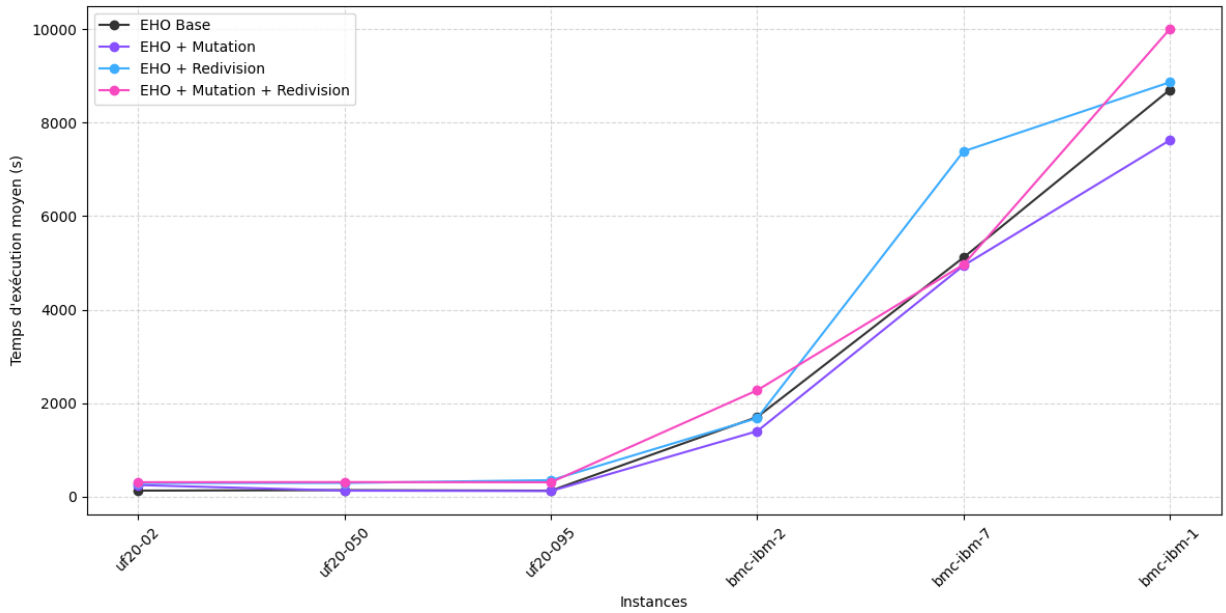

Figure 6.11: Average execution time for each EHO variant across all instances tested

By observing the results from Figures 6.7 to 6.11, as well as Tables [6.4, 6.6, 6.7, 6.8], it is evident that the improved variants of the EHO algorithm exhibit contrasting performances depending on the metrics considered. The base algorithm shows relatively slow convergence with a high stagnation rate, indicating difficulty in escaping local optima. The addition of the mutation operator yields a slight improvement in the satisfiability rate, but its effect remains limited in the face of persistent stagnation. On the other hand, the re-division strategy alone significantly improves the satisfiability rate while maintaining a reasonable execution time, suggesting better exploration of the search space. The combination of both approaches mutation and re-division proves to be the most effective in breaking stagnation, as shown by the significant reduction in stagnation rate in Figure 6.10. Regarding execution time (Figure 6.11), it is observed that the combined version results in the highest time across several tested instances, whereas the mutation only variant is generally the fastest. This difference is explained by the increased complexity of the combined version, which integrates two diversification mechanisms, while mutation alone applies a simple, low-cost modification.

6.3.3 Evaluation of the Hybridization with K-Means Clustering

As presented previously, the implementation of hybridizing EHO with K-Means, the core idea is to group the population into structurally similar clans using K-Means clustering based on Hamming distance. This allows the EHO algorithm to operate on more coherent subgroups, improving both exploration and exploitation.
At each generation, the population is reclustered through multiple `max_iters` steps. This iterative reclustering helps stabilize clan formation and adapt the search process. This motivated us to experiment with different values of `max_iters` to evaluate whether this step plays a significant role in solving the SAT problem effectively. Table (6.9) exhibits the results of this experiments. This is what is shown in the following table :

Instance	max_iters Value	Satisfiability Rate (%)	Stagnation Rate (%)	Execution Time (s)	Score (SR/ET)
uf200-02.cnf	1	98.49	29.85	135.0488	**0.7293**
	3	97.91	27.36	174.3747	0.5615
	5	98.37	11.44	161.9254	0.6073
	10	97.67	22.89	170.743	0.5722
	15	97.26	16.42	170.9746	0.569
bmc-ibm-2.cnf	1	90.29	3.48	1530.0687	**0.059**
	3	89.56	2.99	1777.839	0.0504
	5	89.33	6.47	1880.8006	0.0475
	10	90.01	3.48	1947.07775	0.0462
	15	89.28	2.49	1946.3381	0.0459

Table 6.9: Evaluating the Impact of max_iters in the EHO-KMeans Algorithm

From Table 6.9, it is observed we observe that performing the clustering step only once per generation in the EHO-KMeans algorithm is sufficient, since repeating the clustering multiple times does not significantly improve the satisfiability rate. However, doing so increases the execution time considerably, as we illustrated in these two figures (6.12 and 6.13).

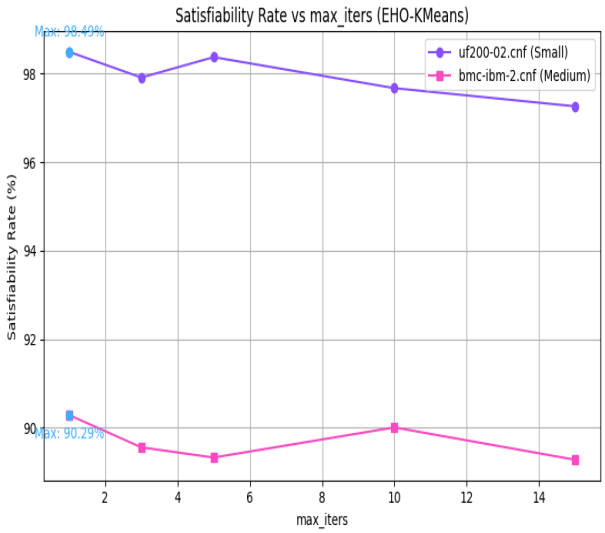

Figure 6.12: Effect of max_iters on Satisfiability Rate in EHO-KMeans Algorithm

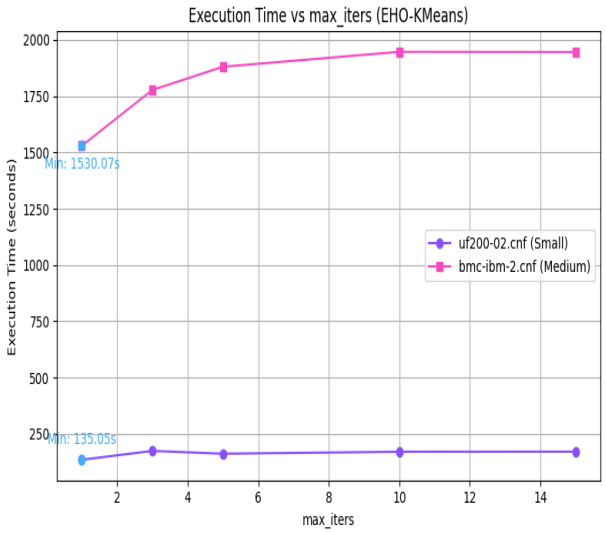

Figure 6.13: Effect of max_iters on Execution Time in EHO-KMeans Algorithm

After analyzing the impact of different max_iters values, evaluation considers now we evaluate the performance of the EHO-KMeans algorithm over 200 iterations, as summarized in Table 6.10.

Benchmark Size	Instances	Attempts Number	Highest Fitness	Average Satisfiability Rate	Average Stagnation Rate	Average Execution Time
Small	uf20-02.cnf	5	845	98.00%	21.39%	173.95 s
	uf20-050.cnf	5	847	97.70%	27.06%	171.18 s
	uf20-095.cnf	5	845	97.49%	22.99%	170.49 s
Medium	bmc-ibm-2.cnf	1	9506	90.01%	3.48%	1947.08 s
	bmc-ibm-7.cnf	1	32380	86.61%	3.48%	6667.13 s
	bmc-ibm-1.cnf	1	47407	86.70%	3.48%	9425.12 s

Table 6.10: Performance Evaluation for the EHO-KMeans algorithm

6.3.4 Evaluation of the Hybridization with DBSCAN Clustering

As we discussed previously in the implementation of hybridizing EHO with DBSCAN, As observed previously, the clustering process plays a crucial role in how the population is structured and evolves. One key addition introduced by this hybridization is the notion of outliers, which are individuals that are not assigned to any cluster and handled separately.

In DBSCAN clustering, the division of clans is primarily based on the spatial proximity of individuals, which is controlled by the parameter epsilon (ε). This parameter defines the maximum distance between two points for one to be considered as in the neighborhood of the other. Thus, epsilon critically influences the number and size of clusters formed, the number of outliers detected, and consequently the behavior of the entire algorithm. This motivated us to conduct tests on the algorithm's performance with different epsilon values in population of 300 solutions, the results of these experiments are presented in Table (6.11).

Benchmark Type	Epsilon	Attempts Number	Satisfiability Rate	Average Outliers Number	Stagnation Rate	Execution Time
Small	0.1	3	97.52%	189 ↓	44.78%	90.99 s
	0.2	3	97.83%	137 ↓	52.07%	107.65 s
	0.3	3	97.56%	106 ↓	65.51%	105.28 s
	0.4	3	97.29%	61 ↓	57.38%	99.27 s
Medium	0.1	1	88.52%	179 ↓	17.91%	1066.79 s
	0.2	1	90.85%	136 ↓	15.92%	1060.86 s
	0.3	1	90.44%	97 ↓	17.41%	1165.70 s
	0.4	1	90.38%	70 ↓	13.93%	1219.01 s

Table 6.11: Performance results for different epsilon values in the hybrid EHO-DBSCAN algorithm.

As shown in Table 6.11, varying epsilon does not significantly affect satisfiability rate or execution time, which remain stable. However, epsilon clearly influences the number of outliers, affecting how individuals are grouped into clans. This impacts on the population structure and optimization dynamics. To better understand this behavior, Figure 6.14 illustrates how the number of outliers evolves during the iterations, providing insight into the clustering dynamics over time.

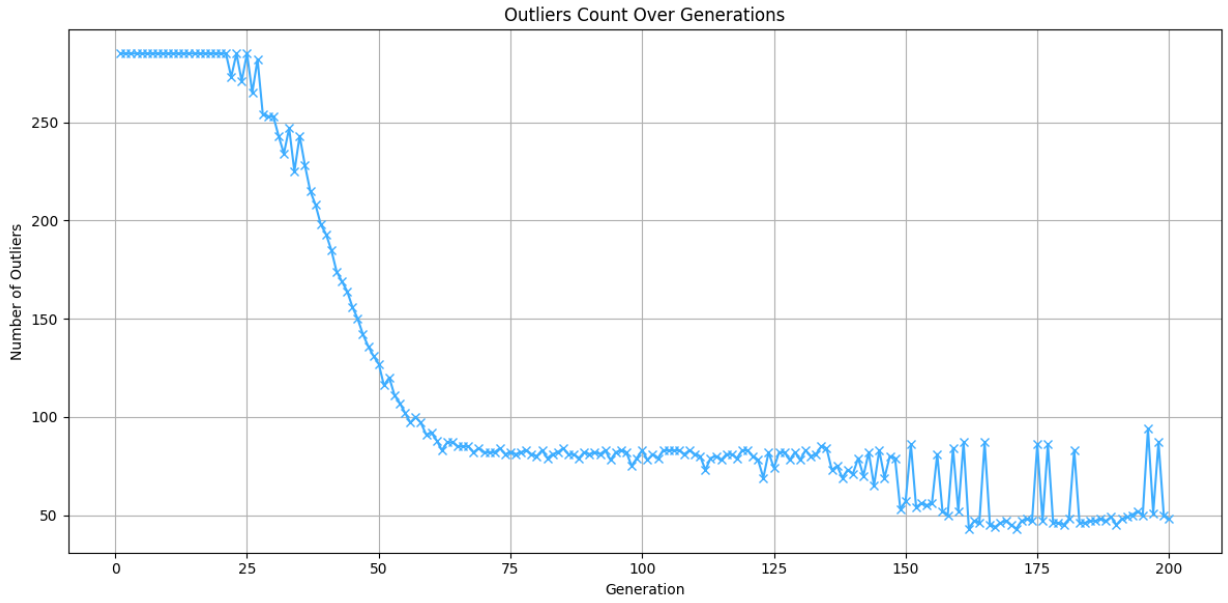

Figure 6.14: Variation of Outliers Count Across Generations

The results shown in Figure (6.14) were obtained using the following parameter settings: population size = 300, maximum generations = 200, $\alpha = 0.1$, $\beta = 0.3$, $\varepsilon = 0.25$, core points count = 15, and MinPts = 2.

Figure 6.14 shows a gradual decrease in the number of outliers throughout the iterations, which indicates the effectiveness of the clan division and outlier detection process in the hybrid EHO+DBSCAN algorithm. This decrease also reflects how outliers are efficiently exploited to inject new diversity into the clans, contributing to maintaining exploration capability during the optimization process.

Finally, to better illustrate the performance of the hybrid EHO-DBSCAN algorithm, we tested it over 200 iterations, with the results summarized in Table X.

Benchmark Size	**Instance**	**Attempts Number**	**Highest Fitness**	**Average Satisfiability Rate**	**Average Stagnation Rate**	**Average Execution Time**
Small	uf20-02.cnf	5	846	97.67%	66.07%	89.68 s
	uf20-050.cnf	5	844	97.79%	63.28%	73.61 s
	uf20-095.cnf	5	848	97.77%	65.07%	75.55 s
Medium	bmc-ibm-2.cnf	1	9638	91.26%	11.44%	1078.26 s
	bmc-ibm-7.cnf	1	33699	90.13%	10.95%	3809.94 s
	bmc-ibm-1.cnf	1	47631	87.11%	5.47%	5482.95 s

Table 6.12: Performance Evaluation of the Hybrid EHO-DBSCAN Algorithm

6.4 Global Evaluation of the Built EHO-Based Variants

After testing all the implementation variants across several parameters and evaluating them on multiple small and medium SAT benchmarks, we present a global comparison of the EHO-based

methods is presented and illustrated . This evaluation is illustrated using three curves showing the comparison of satisfiability (Figure 6.17), stagnation (Figure 6.16), and execution time (Figure 6.16) across all EHO variants on SAT benchmarks.

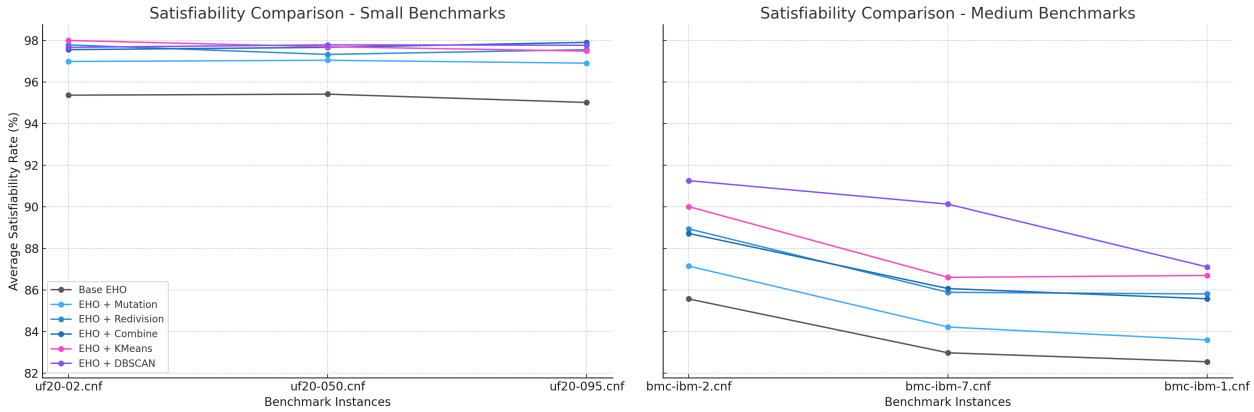

Figure 6.15: Comparison of Average Satisfiability Rates Across EHO Variants on SAT Benchmarks

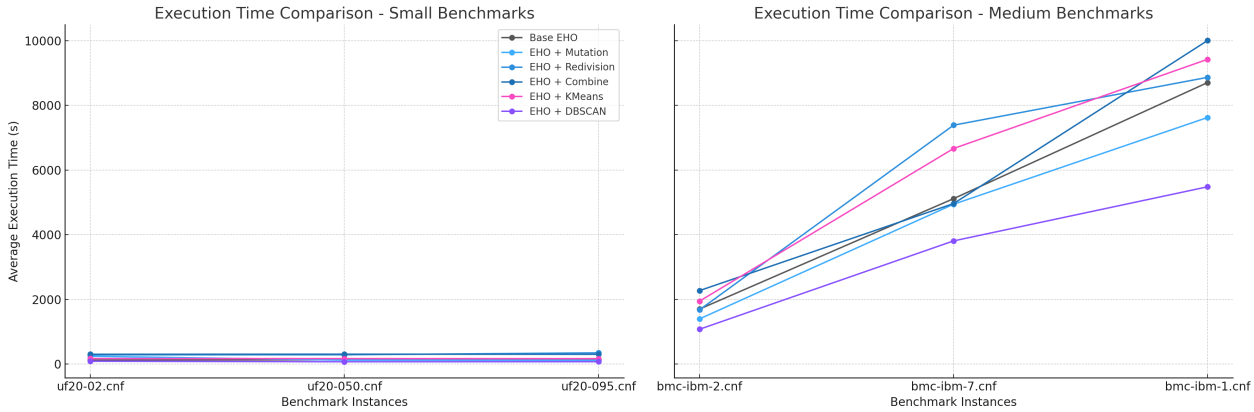

Figure 6.16: Comparison of Average Execution time Across EHO Variants on SAT Benchmarks

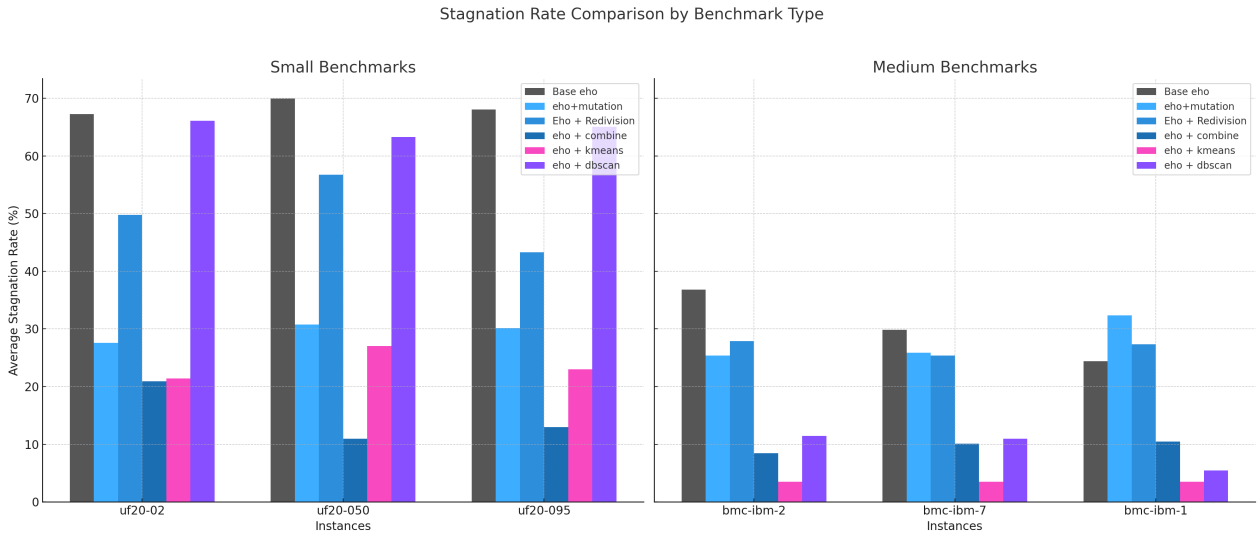

Figure 6.17: Comparison of Average Stagnation Rates Across EHO Variants on SAT Benchmarks

It is observed that the base EHO variant provides an acceptable level of satisfiability but exhibits high stagnation, particularly in small benchmarks. This is due to the limited number of potential solutions in smaller instances, which increases the likelihood of early stagnation. In contrast, medium or larger benchmarks offer more solution diversity, allowing for continuous exploration.

To address the stagnation issue, improved variants based on mutation, redivision, and a combined approach are introduced. These enhancements significantly reduce stagnation and improve satisfiability. Among them, the combined strategy achieves the most balanced performance across all criteria.

The proposed We also designed hybrid EHO variants using KMeans and DBSCAN . These are built on the base EHO and are capable of detecting stagnation during the search process. When stagnation is identified, the system switches directly to one of the improved EHO variants to maintain solution quality. As shown in the figures (6.15, 6.16, 6.17), EHO + DBSCAN emerges as the most effective variant, offering high satisfiability, very low stagnation, and efficient execution time across both small and medium benchmarks. We observe that Data mining techniques provide strong performance when hybridized with EHO, as they enhance both exploration and exploitation during the search process.

6.5 Conclusion

Throughout this chapter, a detailed experimental evaluation was carried out to assess the performance of the adapted EHO algorithm and its various improved versions across a series of SAT benchmark instances. The study began by testing the basic EHO adaptation for binary SAT problems, which, despite demonstrating promising overall performance, revealed a major limitation: the tendency toward stagnation during the search process.

To address this issue, three improvement strategies were explored mutation, re-division, and a combination of both. Each variant showed noticeable improvement over the base EHO model. Among them, the mutation-enhanced version achieved faster execution times, while the re-division and combined approaches delivered higher satisfiability rates and better overall exploration.

To further strengthen the proposed approach, hybridization with clustering-based data mining techniques was introduced. The K-Means hybrid significantly improved solution quality by enabling more coherent clan formations, though at the cost of higher execution time. In contrast, the DBSCAN-based variant not only achieved the highest satisfiability rate but also demonstrated efficient execution time, making it the most effective and balanced EHO variant among those tested. These results confirm the effectiveness and potential of combining metaheuristic search with data-driven strategies in efficiently solving complex combinatorial problems.

Part IV

Conclusion and Perspectives

GENERAL CONCLUSION

This manuscript presented a comprehensive study on adapting and enhancing the Elephant Herding Optimization algorithm to effectively solve the Boolean Satisfiability Problem (SAT), a challenging and well-known NP-complete problem. The core objective was to build an intelligent metaheuristic-based solver that balances exploration and exploitation while addressing the specific constraints of SAT. The work began with the adaptation of the standard EHO to a binary context, enabling it to operate effectively on CNF-encoded Boolean instances by introducing suitable solution representation and fitness evaluation based on satisfied clauses.

During initial evaluations, it became clear that the adapted EHO suffered from stagnation where the population converged prematurely and failed to explore new regions of the solution space. To mitigate this, a series of targeted improvements including the integration of a mutation operator and a re-division strategy were proposed. These enhancements significantly reduced stagnation rates and improved satisfiability scores across benchmark instances, confirming the effectiveness of lightweight structural modifications before exploring more advanced hybrid approaches.

Building on the solid foundation of an adapted and improved EHO, the research advanced toward hybridization with clustering techniques from data mining, namely K-Means and DBSCAN. These techniques were selected due to their relevance to the EHO structure, particularly in replacing EHO's random clan division mechanism with more meaningful, similarity-based clustering.

The K-Means hybridization was implemented by integrating its clustering mechanism into the clan division step of EHO. Before integrating K-Means into EHO, its mechanism was adapted to suit the SAT problem. One of the most essential adjustments crucial for similarity-based clustering was replacing the standard Euclidean distance, which is unsuitable for binary representations, with Hamming distance to better reflect differences between binary solutions. This change ensured that similarity between binary solutions was accurately measured. As a result, the modified clustering produced more coherent clans, enhancing intra-clan collaboration and enabling more effective local exploration, which ultimately led to improved results and more optimal solutions compared to the original and simply improved EHO variants.

The DBSCAN-based hybridization extended the enhancement by addressing both clustering structure and outlier treatment. Inspired by DBSCAN’s capacity to detect dense regions and separate outliers commonly discarded in traditional clustering, the approach to preserve and utilize these outlier solutions within the EHO framework was adapted. Before integration, DBSCAN was tailored for SAT by substituting the standard Euclidean distance with Hamming distance and retaining its

core parameters to suit the binary nature of the problem. For handling outliers, the original elitism strategy of EHO (which replaces the worst individual with the best) was modified to instead improve the identified outliers through mutation and reintroduce them into the population. This approach preserved potentially valuable solutions, reinforced diversity, and maintained exploration. As a result, this proposed solution yielded the best performance among all tested variants, combining high effectiveness with reduced execution time.

These findings confirm the strong potential of the EHO algorithm for solving complex combinatorial problems like SAT, especially when enhanced through data-driven hybridization. This work shows that carefully designed adaptations and smart integrations can significantly improve metaheuristic performance. The improved EHO variants represent a promising foundation for developing more robust and adaptive solvers. They pave the way for building more efficient and scalable tools capable of addressing increasingly complex optimization challenges.

Perspectives

Through this work, the results have shown that enhancing the EHO algorithm whether by improving its internal structure or combining it with other techniques significantly boosted its performance in solving the SAT problem. These improvements led to better solution quality and faster convergence, while effectively reducing stagnation and achieving a more balanced search between exploration and exploitation. Based on these promising outcomes, future work will aim to explore the integration of EHO with other search algorithms or to design more intelligent hybrid versions, including nature-inspired methods and learning-based strategies. The idea of combining EHO with quantum computing also appears to be a promising path for speeding up and improving optimization performance.

Since the SAT problem is a fundamental model for many types of combinatorial optimization problems, generalizing the enhanced algorithm and applying it to other problem domains would be a logical next step toward building robust and adaptable optimization solutions across various application contexts.

Part V

Bibliography

BIBLIOGRAPHY

[1] Joyce Jackson. ``Data mining; a conceptual overview.'' In: *Communications of the Association for Information Systems* 8.1 (2002), p. 19.

[2] Pang-Ning Tan, Michael Steinbach, and Vipin Kumar. *Introduction to Data Mining*. Harlow, Essex, England: Pearson Education Limited, 2014. isbn: 978-1-292-02615-2.

[3] J. Han, J. Pei, and M. Kamber. *Data mining: Concepts and techniques*. Woltham: Elsevier, 2011.

[4] Coderbyte. *Introduction to Data Preprocessing in Data Mining*. Accessed: 2025-02-22. 2025. url: https://medium.com/coderbyte/introduction-to-data-preprocessing-in-data-mining-87f5134ef923.

[5] Sherin Moussa, Dina Fawzy, and Nagwa Badr. ``The Evolution of Data Mining Techniques to Big Data Analytics: An Extensive Study with Application to Renewable Energy Data Analytics.'' In: *Asian Journal of Applied Sciences* 4 (June 2016).

[6] Dipti N Punjani and Kishor Atkotiya. ``A Comprehensive Study of Various Classification Techniques in Medical Application using Data Mining.'' In: (2018).

[7] Archika Jain et al. ``A Review: Data Mining Classification Techniques.'' In: Apr. 2022. doi: 10.1109/ICIEM54221.2022.9853036.

[8] Tan Jun-shan, He Wei, and Qing Yan. ``Application of Genetic Algorithm in Data Mining.'' In: *2009 First International Workshop on Education Technology and Computer Science*. Vol. 2. 2009, pp. 353–356. doi: 10.1109/ETCS.2009.340.

[9] Samia Mandour et al. ``Data Mining Problems Optimization by using Metaheuristic Algorithms: A Survey.'' In: *Multicriteria Algorithms with Applications* 4 (2024), pp. 28–52.

[10] Ivan Bruha. ``Pre-and post-processing in machine learning and data mining.'' In: *Advanced course on artificial intelligence*. Springer, 1999, pp. 258–266.

[11] Ivan Bruha and A Famili. ``Postprocessing in machine learning and data mining.'' In: *ACM SIGKDD Explorations Newsletter* 2.2 (2000), pp. 110–114.

[12] Complex Systems AI. *Combinatorial Optimization*. Accessed: 2025-05-02. n.d. url: https://complex-systems-ai.com/en/combinatorial-optimization-2/.

[13] Saman Almufti et al. ``Overview of Metaheuristic Algorithms.'' In: *Polaris Global Journal of Scholarly Research and Trends* 2 (Apr. 2023), pp. 10–32. doi: 10.58429/pgjsrt.v2n2a144.

[14] Trinav Bhattacharyya et al. ``Mayfly in Harmony: A New Hybrid Meta-Heuristic Feature Selection Algorithm.'' In: *IEEE Access* 8 (Nov. 2020). doi: 10.1109/ACCESS.2020.3031718.

[15] Gai-Ge Wang et al. ``A new metaheuristic optimisation algorithm motivated by elephant herding behaviour." In: *International Journal of Bio-Inspired Computation* 8.6 (2016), pp. 394–409.

[16] Gai-Ge Wang, Suash Deb, and Leandro dos S Coelho. ``Elephant herding optimization." In: *2015 3rd international symposium on computational and business intelligence (ISCBI)*. IEEE. 2015, pp. 1–5.

[17] PhD Assistance. *Algorithm Tips: EHO for Complex Problems - PhD Assistance --- phdassistance.com*. https://shorturl.at/aBGNP. [Accessed 26-03-2024]. November 19, 2019.

[18] Monalisa Nayak et al. ``Elephant herding optimization technique based neural network for cancer prediction." In: *Informatics in Medicine Unlocked* 21 (2020), p. 100445. issn: 2352-9148. doi: https://doi.org/10.1016/j.imu.2020.100445. url: https://www.sciencedirect.com/science/article/pii/S2352914820305955.

[19] Huseyin Hakli. ``BinEHO: a new binary variant based on elephant herding optimization algorithm." In: *Neural Computing and Applications* 32.22 (2020), pp. 16971–16991.

[20] Yuxian Duan et al. ``Gradient-based elephant herding optimization for cluster analysis." In: *Applied Intelligence* 52.10 (2022), pp. 11606–11637.

[21] Mohammed Sannef, EL-Hachemi Guerrout, and Ramdane Mahiou. ``État de l'art sur les méthodes d'optimisation de l'élevage d'éléphants." PhD thesis. Nov. 2021. doi: 10.13140/RG.2.2.22824.49924.

[22] Wei Li and Gai-Ge Wang. ``Improved elephant herding optimization using opposition-based learning and K-means clustering to solve numerical optimization problems." In: *Journal of Ambient Intelligence and Humanized Computing* 14.3 (2023), pp. 1753–1784.

[23] Mostafa A Elhosseini et al. ``On the performance improvement of elephant herding optimization algorithm." In: *Knowledge-Based Systems* 166 (2019), pp. 58–70.

[24] Wei Li, Gai-Ge Wang, and Amir H Alavi. ``Learning-based elephant herding optimization algorithm for solving numerical optimization problems." In: *Knowledge-Based Systems* 195 (2020), p. 105675.

[25] K Shankar et al. ``An efficient image encryption scheme based on signcryption technique with adaptive elephant herding optimization." In: *Cybersecurity and Secure Information Systems: Challenges and Solutions in Smart Environments* (2019), pp. 31–42.

[26] Pushpendra Singh et al. ``Hybrid elephant herding and particle swarm optimizations for optimal DG integration in distribution networks." In: *Electric power components and systems* 48.6-7 (2020), pp. 727–741.

[27] Rasool Bukhsh et al. ``Appliances scheduling using hybrid scheme of genetic algorithm and elephant herd optimization for residential demand response." In: *2018 32nd International Conference on Advanced Information Networking and Applications Workshops (WAINA)*. IEEE. 2018, pp. 210–217.

[28] Tansel Dokeroglu et al. ``A survey on new generation metaheuristic algorithms." In: *Computers & Industrial Engineering* 137 (2019), p. 106040.

[29] Wei Li and Gai-Ge Wang. ``Improved elephant herding optimization using opposition-based learning and K-means clustering to solve numerical optimization problems." In: *Journal of Ambient Intelligence and Humanized Computing* 14.3 (2023), pp. 1753–1784.

[30] Zeynab Hoseini et al. ``A new enhanced hybrid grey wolf optimizer (GWO) combined with elephant herding optimization (EHO) algorithm for engineering optimization." In: *Journal of Soft Computing in Civil Engineering* 6.4 (2022), pp. 1–42.

[31] Stephen A. Cook. ``The complexity of theorem-proving procedures." In: *Proceedings of the third annual ACM symposium on Theory of computing*. ACM. 1971, pp. 151–158. doi: 10.1145/800157.805047.

[32] Martin Davis, George Logemann, and Donald Loveland. ``A machine program for theorem-proving." In: *Communications of the ACM* 5.7 (1962), pp. 394–397.

[33] Joao Marques-Silva and Karem Sakallah. ``Conflict analysis in search algorithms for satisfiability." In: (1996).

[34] Lintao Zhang et al. ``Efficient conflict driven learning in a boolean satisfiability solver." In: *IEEE/ACM International Conference on Computer Aided Design. ICCAD 2001. IEEE/ACM Digest of Technical Papers (Cat. No. 01CH37281)*. IEEE. 2001, pp. 279–285.

[35] Inês Lynce and Joël Ouaknine. ``Sudoku as a SAT Problem." In: *AI&M*. Citeseer. 2006.

[36] Fred Glover. ``Future paths for integer programming and links to artificial intelligence." In: *Computers & operations research* 13.5 (1986), pp. 533–549.

[37] Kenneth Sörensen and Fred Glover. ``Metaheuristics." In: *Encyclopedia of operations research and management science* 62 (2013), pp. 960–970.

[38] Holger H Hoos and Thomas Stützle. ``Propositional satisfiability and constraint satisfaction." In: *Stochastic local search: Foundations and applications. Elsevier* (2004).

[39] Olivier Goudet et al. ``Emergence of new local search algorithms with neuro-evolution." In: *European Conference on Evolutionary Computation in Combinatorial Optimization (Part of EvoStar)*. Springer. 2024, pp. 33–48.

[40] Kenneth Alan De Jong. *An analysis of the behavior of a class of genetic adaptive systems.* University of Michigan, 1975.

[41] Davis Lawrence. ``Handbook of genetic algorithms." In: *Van Nostrand Reinhold* (1991).

[42] Stephen Cook. ``The p versus np problem. the millennium prize problems, 87-104." In: *American Mathematical Society* (2006).

[43] Celina MH De Figueiredo. ``The P versus NP--complete dichotomy of some challenging problems in graph theory." In: *Discrete Applied Mathematics* 160.18 (2012), pp. 2681–2693.

[44] Michael R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. Vol. 29. New York: W. H. Freeman and Company, 2002.

[45] Evelyne Lutton. ``Darwinisme artificiel: une vue d'ensemble." In: *Traitement du signal* 22.4 (2004).