$N°$ d'ordre : 212/2024-C/IN

République Algérienne Démocratique et Populaire
Ministère de L'Enseignement Supérieur et de la Recherche Scientifique
Université des Sciences et de la Technologie HOUARI BOUMEDIENE

Faculté d'Informatique

Thèse de Doctorat

Présentée pour l'obtention du **grade** de **DOCTEUR**

En : INFORMATIQUE

Spécialité : Intelligence Artificielle

Par : KALI ALI Selma

Sujet

Approches métaheuristiques et optimisation pour l'apprentissage automatique : Application à la résolution de problèmes complexes.

Soutenue publiquement, le 26/09/2024, devant le jury composé de :

Mme	DRIAS Habiba	Professeur	à l'USTHB/FIN	Présidente
Mme.	BOUGHACI Dalila	Professeur	à l'USTHB/FIN	Directrice de thèse
M.	REZOUG Abdellah	MCA	à l'UMBB	Examinateur
M.	BOULIF Menouar	Professeur	à l'UMBB	Examinateur
M.	BABA ALI Ahmed Riadh	Professeur	à l'USTHB/FIN	Examinateur
M.	BOUKRA Abdelmadjid	Professeur	à l'USTHB/FIN	Examinateur

$N°$ d'ordre : 212/2024-C/IN

People's Democratic Republic of Algeria
Ministry of Higher Education and Scientific Research
University of Science and Technology HOUARI BOUMEDIENE

Faculty of Computer Science

جامعة هواري بومدين
للعلوم والتكنولوجيا
U S T H B

Doctoral Thesis

Presented to obtain the **degree** of **doctor**

In : COMPUTER SCIENCE

SPECIALITY : Artificial Intelligence

By : KALI ALI Selma

Subject

Metaheuristic approaches and optimization for machine learning: Application to complex problem solving.

Publicly defended, on 26/09/2024, before a jury composed of :

Mrs.	DRIAS Habiba	Professor	at USTHB/FIN	President
Mrs.	BOUGHACI Dalila	Professor	at USTHB/FIN	Thesis supervisor
Mr.	REZOUG Abdellah	MCA	at UMBB	Examiner
Mr.	BOULIF Menouar	Professor	at UMBB	Examiner
Mr.	BABA ALI Ahmed Riadh	Professor	at USTHB/FIN	Examiner
Mr.	BOUKRA Abdelmadjid	Professor	at USTHB/FIN	Examiner

To my dear and loving parents and
To my beloved family members
In memory of my loved ones, may Allah rest their souls,

Acknowledgements

First and foremost, all praise belongs to Allah the Almighty, the Most Gracious and the Most Merciful, for his guidance, help, and blessings given to me during my studies and in completing this thesis.

I would like to express my deepest gratitude and sincere thanks to my supervisor, Prof. Dalila BOUGHACI, for her great trust, confidence, and motivation, for all the continuous scientific and administrative support, and her contribution to this work.

I would also like to address the jury president Prof. Habiba DRIAS and the examiners Prof. Menouar BOULIF, Prof. Abdelmadjid BOUKRA, Prof. Ahmed Riadh BABA ALI, and Dr. Abdellah REZOUG for honoring us with their evaluation of our dissertation and for the effort and time they spent reading and correcting this thesis.

In addition, my most incredible gratitude and appreciation are addressed to Prof. Belaid BENHAMOU for warmly hosting me during my internship and for his invaluable advice. This experience was truly enriching and incredibly beneficial to me.

Of course, I will not forget to convey my sincere appreciation to my dear friends and colleagues - Imene, Lydia, Khadidja, Anfel, Naila, Farida, Widad, Maissa, Celia, and Nawel - whose invaluable support has been instrumental throughout this journey. Furthermore, I express my gratitude to all my fellow PhD students with whom I shared the PhD students' room. The time we spent together and our insightful discussions were enriching and contributed significantly to my personal and academic growth.

With profound sincerity, love, and faith, I extend my heartfelt gratitude to my family for their encouragement and prayers throughout my life. To my beloved parents, thank you for your unconditional love and support; I hope that my accomplishments have made you proud. I am deeply grateful to my dear siblings, grandmothers, and every member of my family for believing in me. Had it not been for all your prayers and benedictions, were it not for your sincere love and help, I would never have completed this work.

Abstract

Machine learning (ML) and metaheuristics, two prominent fields within artificial intelligence, have witnessed remarkable progress in recent years, enabling them to address a wide range of applications. This progress has given rise to a new discipline that combines these two fields, paving the way for innovative approaches to tackle complex problems by leveraging their complementary strengths. The methods proposed within this emerging discipline can be broadly categorized into two categories: approaches that employ metaheuristics to enhance the performance of ML models and approaches that utilize ML techniques to improve the search processes of metaheuristic algorithms.

In this thesis, we proposed approaches that fall into both categories of utilizing the synergy between ML and metaheuristics. Firstly, we introduced a new optimization algorithm called the hybrid GWO-MVO algorithm, which combines the strengths of the Grey Wolf Optimizer (GWO) and Multi-Verse Optimizer (MVO) metaheuristics. We applied the GWO-MVO algorithm to optimize dropout probabilities for Convolutional Neural Networks (CNNs) and tune hyperparameters for Extreme Learning Machines (ELMs). Inspired by the success of these approaches, we then proposed a new optimization approach called qGWO-CNN, which integrates quantum computing principles into the GWO algorithm for CNN hyperparameter optimization.
Subsequently, we explored the practical implications of integrating ML techniques into metaheuristic algorithms for solving Combinatorial Optimization Problems (COPs). In this context, we utilized the Logistic Regression (LR) model in the initialization phase of the Simulated Annealing (SA) algorithm to improve the search process in binary COPs, specifically the Multidimensional Knapsack Problem and the Subset Sum Problem. In the latest contribution, we proposed two new algorithms, DGWO and NFP-DGWO, to address the Winner Determination Problem (WDP). DGWO is an adapted version of the original GWO metaheuristic that can handle WDP's discrete and combinatorial nature. Meanwhile, NFP-DGWO incorporates a learning mechanism that utilizes infrequent patterns via the FP-max ML algorithm to improve solution diversity and prevent stagnation in local optima.

Keywords: Metaheuristics, Machine Learning, Optimization, Combinatorial Optimization Problem.

Résumé

L'apprentissage automatique (ML) et les métaheuristiques, deux domaines importants de l'intelligence artificielle, ont connu des progrès remarquables ces dernières années, ce qui leur a permis d'aborder un large éventail d'applications. Ces progrès ont donné naissance à une nouvelle discipline qui combine ces deux domaines, ouvrant la voie à des approches innovantes pour résoudre des problèmes complexes en tirant parti de leurs forces complémentaires. Les méthodes proposées dans le cadre de cette discipline émergente peuvent être classées en deux catégories : les approches qui utilisent les métaheuristiques pour améliorer les performances des modèles ML et les approches qui utilisent les techniques ML pour optimiser les processus de recherche des algorithmes métaheuristiques.

Dans cette thèse, nous avons proposé des approches qui s'inscrivent dans les deux catégories d'utilisation de la synergie entre la ML et les métaheuristiques. Tout d'abord, nous avons introduit un nouvel algorithme d'optimisation hybride appelé GWO-MVO, qui combine les forces des métaheuristiques Grey Wolf Optimizer (GWO) et Multi-Verse Optimizer (MVO). Nous avons appliqué l'algorithme GWO-MVO pour optimiser les probabilités de dropout pour les réseaux neuronaux convolutifs (CNNs) et régler les hyperparamètres pour les Extreme Learning Machines (ELMs). Inspirés par le succès de ces approches, nous avons ensuite proposé une nouvelle approche d'optimisation appelée qGWO-CNN, qui intègre les principes de l'informatique quantique dans l'algorithme GWO pour l'optimisation des hyperparamètres des CNN.
Par la suite, nous avons exploré les implications pratiques de l'intégration des techniques de ML dans les algorithmes métaheuristiques pour résoudre les problèmes d'optimisation combinatoire (COPs). Dans ce contexte, nous avons utilisé le modèle de régression logistique (LR) dans la phase d'initialisation de l'algorithme de recuit simulé (SA) pour améliorer le processus de recherche dans les problèmes d'optimisation binaires, en particulier le problème du sac à dos multidimensionnel et le problème de la somme de sous-ensembles. Dans notre dernière contribution, nous avons proposé deux nouveaux algorithmes, DGWO et NFP-DGWO, pour résoudre le problème de la détermination du gagnant (WDP). DGWO est une version adaptée de la métaheuristique GWO originale qui peut gérer la nature discrète et combinatoire du WDP. Par ailleurs, NFP-DGWO intègre un mécanisme d'apprentissage qui utilise des modèles peu fréquents via l'algorithme FP-max de ML afin d'améliorer la diversité des solutions et d'éviter la stagnation dans les optima locaux.

Mots-clés: Métaheuristique, Apprentissage Automatique, Optimisation, Problème d'Optimisation Combinatoire.

ملخص

شهد التعلُم الآلي (ML) والنُهج الميتاهوريستية (Metaheuristics) ، وهما مجالان بارزان في الذكاء الاصطناعي، تقدماً ملحوظاً في السنوات الأخيرة، مما مكنهما من معالجة مجموعة واسعة من التطبيقات. وقد أدى هذا التقدم إلى نشوء تخصص جديد يجمع بين هذين المجالين، مما يمهد الطريق أمام مناهج مبتكرة لمعالجة المشاكل المعقدة من خلال الاستفادة من نقاط القوة التكميلية لكل منهما. يمكن تصنيف الأساليب المقترحة في هذا المجال الناشئ بشكل عام إلى فئتين: الأساليب التي تستخدم أساليب النُهج الميتاهوريستية لتحسين أداء نماذج التعلم الآلي والأساليب التي تستخدم تقنيات التعلم الآلي لتعزيز عمليات البحث في خوارزميات النُهج الميتاهوريستية .

في هذه الأطروحة، اقترحنا مناهج تندرج ضمن فئتي الاستفادة من التآزر بين التعلم الآلي واستراتيجيات النُهج الميتاهوريستية . أولاً، قدمنا خوارزمية تحسين جديدة تسمى خوارزمية GWO- MVO الهجينة، والتي تجمع بين نقاط القوة في Grey Wolf Optimizer (GWO) و Multi Verse Optimizer (MVO) . لقد طبقنا خوارزمية GWO-MVO لتحسين احتمالات التسرب للشبكات العصبونية الالتفافية (CNNs) وضبط المعاملات الفائقة لآلات التعلم القصوى (ELMs) . استلهامًا من نجاح هذه الأساليب، اقترحنا بعد ذلك نهجًا جديدًا للتحسين يسمى qGWOCNN ، والذي يدمج مبادئ الحوسبة الكمية في خوارزمية GWO لضبط المعلمات الفائقة ل CNNs .

بعد ذلك، استكشفنا التأثيرات العملية لدمج تقنيات تعلّم الآلة في خوارزميات النُهج الميتاهوريستية لحل مسائل الاستمثال التوافقي (COPs) . في هذا السياق، استخدمنا نموذج الانحدار اللوجستي (LR) في مرحلة التهيئة لخوارزمية محاكاة عملية التلدين (SA) لتحسين عملية البحث في Multidimensional Knapsack Problem و Subset Sum Problem . في المساهمة الأخيرة، اقترحنا خوارزميتين جديدتين، DGWO و NFP-DGWO ، لمعالجة مشكلة تحديد الفائز (WDP) . DGWO هي نسخة معدّلة من خوارزمية GWO الأصلية التي يمكنها التعامل مع طبيعة WDP . وفي الوقت نفسه، تتضمن NFP-DGWO آلية تعلم تستخدم أنماطًا نادرة عبر خوارزمية FP-max لتحسين تنوع الحلول ومنع الركود في الحل الأمثل المحلي.

الكلمات المفتاحية: النُهج الميتاهوريستية ، التعلم الآلي، التحسين الأمثل، مسائل الاستمثال التوافقي.

Contents

List of Figures

List of Tables

List of Algorithms

Acronyms

AI Artificial intelligence

ANN Artificial Neural Network

BA Bat Algorithm

BARD Bayesian Automatic Relevance Determination

AQ Adaptive Query

ASP Algorithm Selection Problem

CAE Convolutional Autoencoder

CNN Convolutional Neural network

COP Combinatorial Optimization Problem

CV Cross-Validation

DE Differential Evolution

DGWO Discrete Grey Wolf Optimizer

DL Deep Learning

DNN Deep Neural Network

EA Evolutionary Algorithm

ELM Extreme Learning Machine

FA Firefly Algorithm

FP-growth Frequent Pattern growth

GNN Graph neural network

GPR Gaussian Process Regression

GWO Grey Wolf Optimizer

HS Harmony Search

k-NN k-Nearest Neighbors

k-NNR k-Nearest Neighbors Regression

KP Knapsack Problem

LEM Learnable Evolution Model

LR Logistic Regression

MKP Multidimensional Knapsack Problem

ML Machine Learning

MLE Maximum Likelihood Estimation

MLP Multi-Layer Perceptron

MOP Multi-Objective Optimization

MVO Multi-Verse Optimizer

PN Pointer Network

PSO Particle Swarm Optimization

QAP Quadratic Assignment Problem

QC Quantum Computing

QPSO Quantum-behaved Particle Swarm Optimization

RFR Random Forest Regression

RL Reinforcement Learning

RNN Recurrent Neural Network

SA Simulated Annealing

SC Soft Computing

SI Swarm Intelligence

SLS Stochastic Local Search

SOP Single Objective Optimization

SSP Subset Sum Problem

SVM Support Vector Machine

SVR Support Vector Regression

TSP Travelling Salesman Problem

VNS Variable Neighborhood Search

WDP Winner Determination Problem

Introduction

Research motivations and objectives

Machine Learning (ML), a branch of Artificial Intelligence (AI), specializes in creating models and algorithms that can learn and predict outcomes without explicit programming. This data-driven approach streamlines decision-making processes and reduces human effort across various applications [8]. One of the significant advantages of ML is its capacity to detect intricate patterns and relationships in complex and high-dimensional datasets. ML algorithms thrive in identifying patterns that may be challenging or impossible for humans to discern manually, leading to valuable insights and solutions. Moreover, ML models possess exceptional flexibility, continually enhancing their performance as new data emerges. This adaptive learning capability makes ML ideal for environments where the underlying patterns or relationships within the data evolve over time, guaranteeing the models' accuracy and relevance. Thanks to its versatility, ML has found broad applications in various fields, such as natural language processing, computer vision, and many others, unlocking innovative solutions and propelling technological advancements.

Although ML offers numerous advantages, it also poses a number of challenges that must be addressed [48]. One of the most crucial issues is the quality and availability of data. Poor quality or insufficient data can lead to inaccurate models and subpar performance. Additionally, determining the optimal architecture and hyperparameters can be a daunting task due to the extensive search space and the lack of a universally effective method. Typically, this process involves experimenting with techniques such as trial-and-error, grid searches, or more advanced approaches like random search or Bayesian optimization. However, these methods can be time-consuming and computationally demanding. Moreover, the selection of architecture and hyperparameters can be heavily influenced by the problem domain, data characteristics, and desired trade-offs between performance metrics. Furthermore, developing complex ML models

can be resource-intensive, requiring powerful hardware and significant computational resources that may not be accessible or affordable for some organizations.

Metaheuristics, on the other hand, are a class of approximate optimization algorithms designed to find high-quality solutions to complex problems by intelligently guiding the search process. These algorithms draw inspiration from natural phenomena. Metaheuristics are particularly valuable for problems with large search spaces, specifically Combinatorial Optimization Problems (COPs), where traditional optimization techniques may struggle to find optimal or near-optimal solutions within reasonable computational time. Recently, metaheuristics have emerged as important tools for solving challenging optimization problems across various domains. These algorithms have proven their versatility and robustness in tackling various optimization challenges, making them indispensable in modern problem-solving frameworks.

Throughout the iterative search process, metaheuristics generate a significant amount of data that has the potential to contain valuable insights. This data may reveal the characteristics of both good and bad solutions, the efficacy of different operators at different stages, and the priority of search operators. Despite this, traditional metaheuristics fail to extract knowledge from this data [53].

The synergy between ML and metaheuristics has opened up new avenues for solving complex problems by leveraging the complementary strengths of these two AI search fields. Metaheuristics can efficiently explore vast search spaces and find high-quality solutions, which can then be utilized to train ML models or refine their parameters. Conversely, ML algorithms can learn patterns and relationships from data, providing valuable insights and knowledge that can be seamlessly incorporated into metaheuristic optimization algorithms.

This thesis aims to investigate the potential of combining ML and metaheuristics to optimize hyperparameters for ML models and improve the performance of metaheuristic algorithms when solving COPs. While the synergy between ML and metaheuristics remains an underexplored and poorly studied area, this work introduces several innovative methods to efficiently determine optimal ML hyperparameters and leverages ML techniques to enhance metaheuristic algorithms. Each proposed approach is thoroughly assessed for effectiveness and potential contributions to this emerging field through rigorous evaluation and comparison with state-of-the-art methods. Ultimately, this exploration seeks to advance the understanding and application of ML and metaheuristics in synergy, paving the way for more powerful and effective solutions to real-world problems.

Contributions

The approaches proposed in this thesis can be classified into two distinct categories: (1) employing metaheuristic techniques for optimizing the hyperparameters of ML models, and (2) improving metaheuristic algorithms by using ML models. The main contributions of this research work are as follows:

1. We proposed an innovative hybrid metaheuristic algorithm, GWO-MVO, that seamlessly merges the Grey Wolf Optimizer (GWO) and the Multi-Verse Optimizer (MVO) to estimate appropriate dropout probability rates for convolutional neural networks (CNNs). Extensive experimentation has demonstrated that GWO-MVO is exceptionally effective, surpassing the baseline GWO. Additionally, our algorithm performs comparably to other state-of-the-art methods on benchmark image classification datasets.

2. Expanding on the success of our initial contribution, we have extended the GWO-MVO algorithm to address more intricate ML hyperparameter optimization problems. Our focus was on boosting the performance of Extreme Learning Machine (ELM) networks for image classification tasks by addressing the issues stemming from random weight initialization, poor input feature quality, and variable image quality. The contribution involved optimizing weight and bias initialization via GWO-MVO, as well as improving input feature quality using CNN and convolutional autoencoder (CAE), resulting in more informative and effective input features. By synergistically optimizing both the hyperparameters and the input features, our work has significantly improved ELM networks' prediction stability and generalization performance for image classification tasks.

3. Our third contribution showcased the potential of combining quantum computing principles with metaheuristics to develop more effective and efficient hyperparameter optimization methods. Specifically, we developed a quantum-inspired variant of GWO, called qGWO, which is tailored to search efficiently for optimal hyperparameters in CNNs. Our approach, qGWO-CNN, utilizes quantum computing principles to improve GWO's optimization capabilities. Through a series of experiments, the qGWO-CNN approach demonstrated promising results for CNN hyperparameter optimization, significantly outperforming the standard GWO and our previously developed GWO-MVO hybrid algorithm. It is important to note that our optimization efforts were focused solely on the first convolutional layer of the CNN. Despite this targeted optimization on a single layer, the performance

achieved by qGWO-CNN was comparable to state-of-the-art methods that optimize the hyperparameters across all network layers. This highlights the potential for further performance improvement by extending the qGWO-CNN approach to optimize the hyperparameters of additional layers in the CNN architecture.

4. In our fourth contribution, we delved into the efficacy of the logistic regression (LR) model in tackling binary combinatorial optimization problems (COPs). Additionally, we explored how integrating the LR model with the Simulated Annealing (SA) metaheuristic algorithm can enhance its performance. Through our study, we demonstrated how LR can aid in predicting the probability of an element being part of the optimal solution for binary COPs. Our experiments on instances of the Multidimensional Knapsack Problem (MKP) and Subset Sum Problem (SSP) showcased that by combining the LR model with the SA metaheuristic algorithm, we were able to introduce more intelligence to the search process and improve the quality of the solutions generated.

5. Finally, we proposed a novel Discrete Grey Wolf Optimizer (DGWO) algorithm for tackling the Winner Determination Problem (WDP). To further enhance the performance of this metaheuristic algorithm, we introduced a learning mechanism that leverages frequent itemset extraction techniques, notably the FP-max algorithm, during the search process. Unlike our previous contribution, where the ML model was employed in the initialization phase of the metaheuristic, this approach seamlessly integrates the ML technique during the search process itself. By exploring uncommon patterns that may not be present in high-quality solutions, we strived to improve the diversity of the solutions considered by the search algorithm, thereby avoiding premature convergence or stagnation at local optima. Our research has demonstrated that this method produces highly encouraging results, significantly enhancing the quality of solutions obtained for the WDP.

List of publications

Conference papers

- **Kali Ali, S.**, Boughaci, D. (2023). Hybrid Approach Based on Grey Wolf Optimizer for Dropout Regularization in Deep Learning. In: Chikhi, S., Diaz-Descalzo, G., Amine, A., Chaoui, A., Saidouni, D.E., Kholladi, M.K. (eds) Modelling and Implementation of Complex Systems. MISC 2022. Lecture Notes

in Networks and Systems, vol 593. Springer, Cham. *https://doi.org/10.1007/978-3-031-18516-8_9*

- **Kali Ali, S.**, Boughaci, D. (2023). Extreme Learning Machines Based on Convolutional Neural Network and Convolutional Autoencoder for Image Classification: Comparative Study. In: Drias, H., Yalaoui, F., Hadjali, A. (eds) Artificial Intelligence Doctoral Symposium. AID 2022. Communications in Computer and Information Science, vol 1852. Springer, Singapore. *https://doi.org/10.1007/978-981-99-4484-2_18*
- **Kali Ali, S.**, Boughaci, D. (2023). Combining Simulated Annealing with Logistic Regression for Binary Combinatorial Optimization Problems. In IEEE International Conference on Networking, Sensing and Control (ICNSC), Marseille, France, 2023, pp. 1-6. *10.1109/ICNSC58704.2023.10318995*
- **Kali Ali, S.**, Boughaci, D. (2024). Quantum Inspired Grey Wolf Optimizer for Convolutional Neural Network Hyperparameter Optimization. In: Drias, H., Yalaoui, F. (eds) Quantum Computing: Applications and Challenges. QSAC 2023. Information Systems Engineering and Management, vol 2. Springer, Cham. *https://doi.org/10.1007/978-3-031-59318-5_5*

Journal papers

- **Selma Kali Ali** and Dalila Boughaci. 2024. Improving extreme learning machine model using deep learning feature extraction and grey wolf optimizer: Application to image classification. Int. Dec. Tech. 18, 1 (2024), 457–483. *https://doi.org/10.3233/IDT-230382*

Structure of the thesis

The rest of the thesis is structured as follows: **Chapter 1** provides background concepts necessary for understanding the research. It includes an overview of metaheuristics and ML fundamentals, popular algorithms, and techniques. **Chapter 2** explores state-of-the-art methods for combining metaheuristics and ML. It also discusses challenges and considerations for achieving effective synergy between these two paradigms. **Chapter 3** introduces the GWO-MVO algorithm, first applied to estimate the optimal dropout probability rate in CNNs. Additionally, this chapter presents an innovative approach for enhancing the performance of ELM models using deep learning feature extraction techniques and the GWO-MVO algorithm. **Chapter 4** details the qGWO-CNN approach, a method proposed for optimizing CNN hyperparameters. This approach

integrates quantum computing principles with the GWO to enhance the efficiency of the hyperparameter optimization process. **Chapter 5** presents the approaches to solving COPs by synergizing metaheuristics and ML techniques. First, methods incorporating LR with the SA metaheuristic are presented for solving binary COPs. Subsequently, an intelligent DGWO algorithm is introduced and developed specifically to address the WDP in **Chapter 6** . The chapter outlines the basic DGWO algorithm and its intelligent variant that incorporates a learning mechanism based on frequent itemset extraction techniques to support the search process. **Finally**, the thesis concludes by summarizing the essential findings and contributions made throughout the research. Additionally, potential future research directions are discussed, highlighting opportunities for further exploration and advancement in the synergistic integration of ML and metaheuristic optimization techniques.

Part I

State-of-the-art

Chapter 1

Background

1.1 Introduction

Artificial intelligence (AI) was born in the 1950s when pioneering computer scientists began exploring whether machines could "think" - a question we continue to investigate today. AI is a broad area of computer science that aims to create systems capable of performing tasks that typically require human-level intelligence.

Soft Computing (SC) is a subfield of AI comprising computational techniques inspired by human reasoning and intelligence to solve complex real-world problems. CS emerged in the late 20th century as a response to the limitations of traditional "hard" computing approaches. The term "soft computing" was coined by Lotfi A. Zadeh, the pioneer of fuzzy logic, in the early 1990s [23]. SC techniques were developed to handle challenges involving uncertainty, imprecision, and partial truth that conventional computational methods struggled to solve. SC includes three main branches: reasoning techniques, algorithm-based methods, and Machine Learning (ML), as illustrated in Figure 1.1.

Since its inception, SC has found applications in diverse fields such as engineering, finance, healthcare, and robotics. The development of SC techniques has led to tackling increasingly complex and challenging problems in various areas, such as pattern recognition, optimization, decision-making, and data analysis.

Given that this thesis focuses on two essential aspects of SC—metaheuristics and ML methods—this chapter will cover three core concepts: optimization, metaheuristics, and ML.

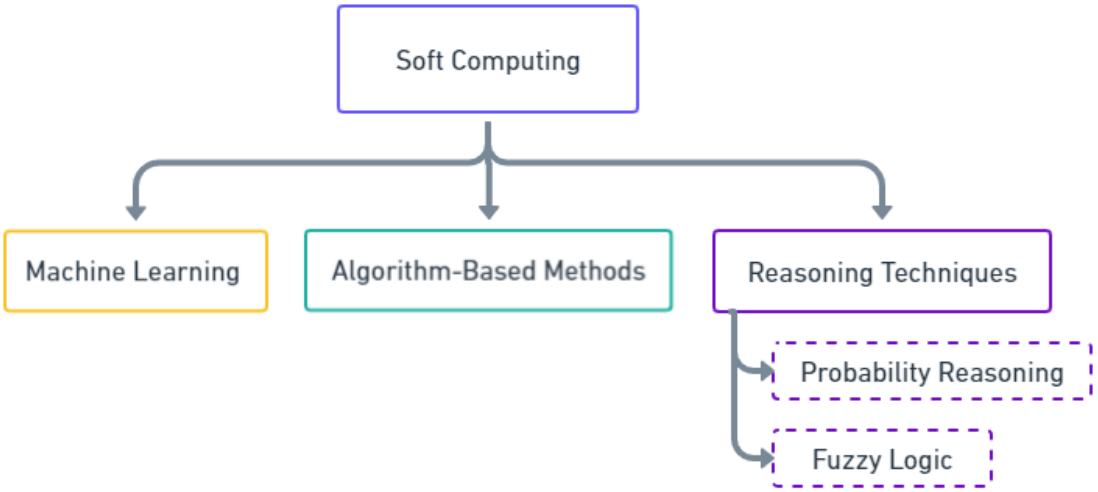


Figure 1.1: Soft Computing techniques (adapted from [64]).

1.2 Optimization

Optimization involves finding the optimum values of the decision variables that lead to the optimal result. This "optimal" outcome is defined as either minimizing the objective (e.g., in the case of cost) or maximizing the goal (e.g., in the case of profit) [32]. The efficient solving of optimization problems has long been the subject of much attention [79].

Optimization problems arise in many fields, including operations research, engineering, bioinformatics, management, etc. With the progress of information technology, new optimization problems are being introduced. However, solving highly complex optimization problems often requires significant computing resources. Computational complexity and the possibility of applying optimization to real-world problems have made optimization a very active area of research in AI.

An optimization problem is denoted by a set of candidate solutions D representing the decision space (search space) and an objective function f, which assigns a value (usually a real value) to each solution s, indicating its quality. Candidate solutions may vary in type and structure depending on the specific problem. In addition, constraints may be imposed to restrict the space of feasible solutions. For a minimization problem, the goal is to find the solution s^* in D that minimizes the objective function formally [30]:

$$s^* = \underset{\forall s \in D}{arg\ min} f(s) \tag{1.1}$$

1.2.1 Optimization types

Optimization problems can be categorized based on various factors, including problem formulation, structure, type, and the nature of the optimal solution(s) sought. The categorization of optimization problems often depends on the specific objective and characteristics of the problem at hand.

1.2.1.1 Global and local optimization

Global and local optimization are two distinct strategies employed to discover optimal solutions for optimization problems. Global optimization consists of identifying the best possible solution in the search space. It aims to find the global optimum, i.e., the best solution among all candidate solutions. On the other hand, local optimization focuses on finding optimal solutions at the local level, i.e., the best solution in a small neighborhood of the search space. It converges on local optima rather than on the global optimum.

1.2.1.2 Constrained and unconstrained optimization

Constrained and unconstrained optimization are two fundamental paradigms in the optimization field, depending on the nature of the problem. In unconstrained optimization, the aim is to find the optimal solution for an objective function without any restrictions on the decision variables, allowing free searching of the entire solution space. Conversely, constrained optimization involves optimizing an objective function while respecting specified constraints that solutions must satisfy. These constraints introduce real-world restrictions on the possible solutions, making the optimization process more complex. Unconstrained optimization often focuses on finding critical points, while constrained optimization requires balancing the optimization goal with satisfying the constraints.

1.2.1.3 Linear and no-linear optimization

Linear and non-linear optimization are two main classes based on the mathematical form of the objective function and constraints. In linear optimization, both the objective and constraints are linear functions. The decision variables appear to be the first power only, and there are no products or powers of decision variables in the objective function or constraints. Meanwhile, non-linear optimization contains non-linearities in the objectives or constraints. This means that decision variables may appear to powers

other than one or be involved in products, ratios, or other non-linear operations. Non-linear programs are more challenging to solve globally due to potential non-convexity and multiple local optima.

1.2.1.4 Discrete and continuous optimization

Based on the type of variables involved, optimization problems can be categorized into discrete and continuous optimization. In discrete optimization, decision variables only take discrete values. These values are often integers or belong to a finite set. This represents discrete configurations or permutations. Therefore, discrete optimization focuses on Combinatorial Optimization Problems (COPs). Continuous optimization deals with decision variables that can take any value in a specified range. Thus, discrete solutions have a finite and countable number of possible solutions, while continuous solutions have an infinite and uncountable set of solutions. In the rest of this subsection, we will provide examples to illustrate continuous and discrete optimization problems.

Multidimensional Knapsack Problem

The Multidimensional Knapsack Problem (MKP) is a COP that extends the standard Knapsack Problem (KP) to multiple dimensions. In the MKP, each item has values across multiple resources rather than just weight in one dimension. The objective is to pick a subset of items that maximizes the total profit while satisfying capacity constraints in each dimension. Mathematically, given n items and m dimension, the MKP is defined in Equation 1.2 [55].

$$MKP \begin{cases} \max \sum_{j=1}^{n} p_j x_j \\ \text{subject to } \sum_{j=1}^{n} w_{i,j} x_j \leq c_i, \text{ for i = 1,...,m} \\ x_j \in \{0,1\}, \text{ for j = 1,...,n} \end{cases} \tag{1.2}$$

where p_j is the profit value of item j, $w_{i,j}$ is the weight of item j in dimension i, c_i is the capacity constraint for dimension i, and x_j is a binary decision variable indicating whether item j is selected or not.

The MKP is NP-hard [55], where the number of possible solutions grows exponentially as the problem size increases, creating a vast search space. This problem has diverse practical applications, including resource allocation, scheduling, and portfolio optimization in finance.

Subset Sum Problem

The Subset Sum Problem (SSP) is a decision problem that inquires whether there exists a subset within a given set of integers whose sum is exactly equal to a specified target value T. Alternatively, the SSP can also be formulated as an optimization, where the goal is to identify a subset of elements from the given set whose sum is as close as possible to T without surpassing it. However, this optimization version of SSP is NP-hard [58]. SSP is widely studied due to its theoretical significance and practical applications across many domains, including finance, logistics, scheduling, and cryptography.

Mathematically, given a set S of n positive integers $S = a_1, a_2, ..., a_n$ and a target T, the optimization aims to find a subset by selecting 0/1 variables x_i to:

$$SSP \begin{cases} \max \sum_{i=1}^{n} a_i \times x_i \\ \text{subject to} \sum_{i=1}^{n} a_i \times x_i \leq T \quad x_i \in \{0,1\} \end{cases} \tag{1.3}$$

Winner Determination Problem

The optimal Winner Determination Problem (WDP) in combinatorial auctions involves finding the set of winning bids S that maximizes the auctioneer's revenue, subject to the constraint that each item can be allocated to at most one bidder. The WDP is equivalent to the NP-hard weighted set packing problem [36, 104].

Formally, let $M = 1, 2, ..., m$ be the set of m items for auction. Let $B = \{B1, B2, ..., Bn\}$ be the set of n bids, where each bid Bj is a tuple (P_j, G_j) comprising a price ${}_Pj$ and a subset ${}_Gj$ of requested items from M. A feasible solution S is a subset of bids from B such that no two bids in S share any commonly requested item ($\forall Bi, Bj \in S(i = j), G_i \cap G_j = \emptyset$). Let $x_j = 1$ if bid Bj is in the winning set S; otherwise, it is 0. Further, consider a 0-1 matrix $A_{m \times n}$ with m rows and n columns. $\forall i \in [1, m], \forall j \in [1, n], A_{i,j} = 1 \; iff \; i \in G_j$, and $A_{i,j} = 0$ otherwise. The WDP can be modeled as the following integer programming formulation [41]:

$$WDP \begin{cases} \max \sum_{j=1}^{n} P_i \times x_j \\ \text{subject to} \sum_{j=1}^{n} a_{i,j} \times x_j \leq 1, \; i \in \{1, ..., m\} \quad x_j \in \{0,1\} \end{cases} \tag{1.4}$$

The WDP finds applications across various domains, including combinatorial auctions, resource allocation, transportation and logistics, and financial optimization.

Function Optimization

Function optimization is a mathematical process that involves finding the values of the decision variables that result in either the maximum or minimum value of this function. The decision variables are continuous, and small changes in their values can lead to changes in the objective function. Function optimization is widely used in various fields like engineering, economics, physics, and ML.

Hyperparameter tuning in ML

Parameter estimation in ML refers to determining the optimal values for a model's parameters to perform well on a given task. In ML, a model is a mathematical representation or algorithm that maps input data to predictions or decisions. Models typically have parameters that influence their behavior, and parameter estimation aims to find the values that result in the best performance. A more detailed explanation will be provided in Section 1.4.2.2.

1.2.1.5 Single and multi-objectives optimization

Single-objective optimization and multi-objective optimization designate two categories of optimization problems, depending on the number of objectives to be optimized. Single Objective Optimization (SOP) involves optimizing just one objective function, i.e., finding the values of the decision variables that optimize a single objective function. Multi-Objective Optimization (MOP), on the other hand, deals simultaneously with two or more objective functions. Here, the aim is to find solutions representing a compromise between the conflicting objectives, known as the Pareto front.

1.2.2 Optimization search algorithms

Finding the optimal solution to a complex optimization problem poses significant challenges. Several optimization algorithms have been proposed to address these challenges. These algorithms can be categorized into two main approaches - exact methods and approximate methods, based on the objectives and characteristics, as shown in Figure 1.2.

Exact optimization algorithms aim to find the global optimal solution to a problem by exhaustively searching the entire solution space. They use mathematical techniques to traverse the space of candidate solutions in a complete, systematic manner. However, most real-world optimization problems are NP-hard, so finding optimum solutions in polynomial time is not feasible unless $NP = P$. Since the exhaustive search requires

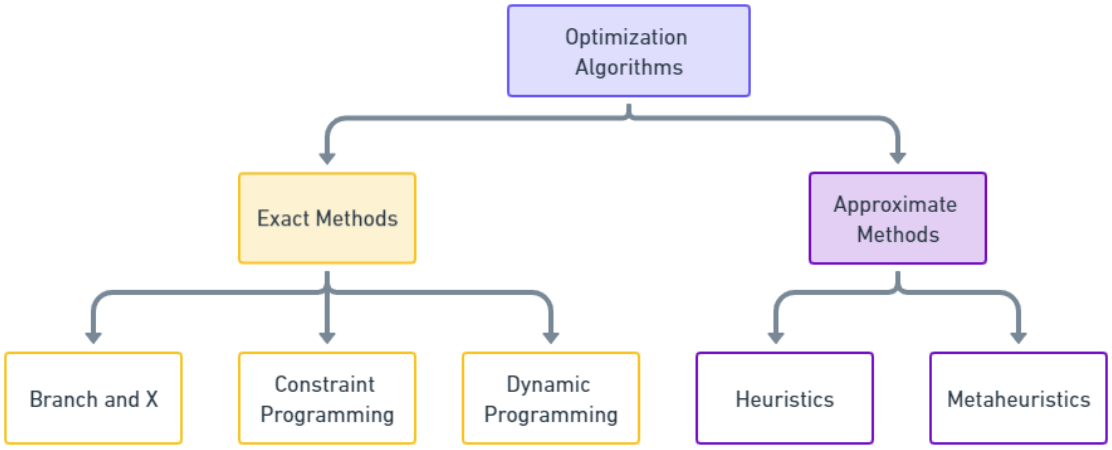


Figure 1.2: Classification of optimization algorithms (adapted from [59]).

computational effort growing exponentially with the problem size, exact algorithms can only be leveraged for small and some medium-sized optimization problems in practice.

Unlike exact algorithms, approximate methods do not guarantee to find a globally optimal solution. Although they do not guarantee optimality, they provide good-quality solutions more efficiently than exhaustive search. For this reason, these techniques are well-suited to large-scale problems for which exact methods become impractical. Approximate approaches are divided into heuristic and metaheuristic algorithms. While heuristics are problem-specific algorithms, metaheuristics are generic methods that can be applied to most optimization problems. As mentioned above, this thesis will focus on metaheuristics.

1.3 Metaheuristics

The term "heuristic" comes from the Greek word "heuriskein" meaning to discover or find. It refers to problem-specific rules and strategies for finding solutions [114]. The prefix "meta" also has Greek origins and means "upper level". Metaheuristics were first described by Fred Glover in 1986 as high-level heuristics that can be applied across optimization problems rather than being adapted to a specific problem [38]. *Metaheuristics* can be defined as search algorithms that provide general guiding strategies to solve complex optimization problems under limited domain knowledge and computational resources. When the search space grows exponentially, metaheuristics

present an efficient approach for finding near-optimal or good-quality solutions in a reasonable time.

Metaheuristics are widely used in diverse fields such as finance, production management, science, and engineering. Most metaheuristics share the following characteristics [21]:

- They are often inspired by principles from nature and sciences like physics, biology, and ethology.
- They incorporate stochastic components involving randomization rather than being deterministic.
- They have multiple parameters that need tuning and adaptation to match the specific problem.

Based on their search philosophy, metaheuristics can be classified into two primary categories: population-based methods and single-solution-based (trajectory-based) methods (see Figure 1.3).

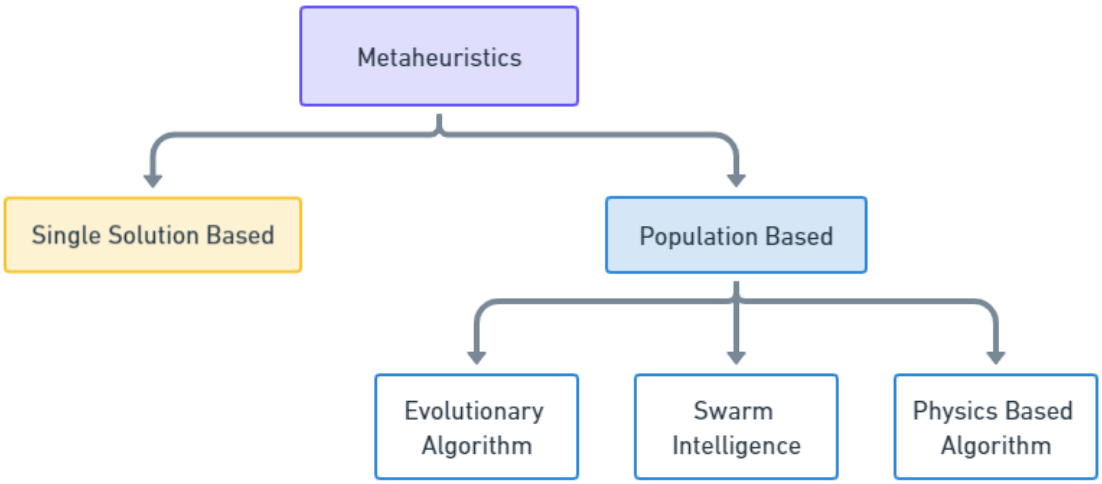


Figure 1.3: Classification of metaheuristics.

According to the no-free lunch theorem [123], no single algorithm can be the best-performing algorithm for all problems. Hence, many metaheuristics have been developed and applied to solve theoretical and practical optimization tasks across diverse domains, including finance, production management, science, and engineering. Additionally, hybrid approaches combining multiple metaheuristic techniques have emerged to harness their complementary strengths and achieve better performance than individual methods alone.

In the following, we will introduce the fundamental concepts and discuss both single-solution and population-based metaheuristics.

1.3.1 Common concepts

This section introduces fundamental concepts shared by all types of metaheuristics. These concepts are essential for addressing any optimization problem. For a more comprehensive overview, refer to [114].

1.3.1.1 Solution representation

The representation of solutions is a crucial design factor in metaheuristic optimization algorithms. It determines the structure and size of the search space. Therefore, the efficiency of exploring the search space depends on the choice of solution representation. An effective encoding should satisfy several properties:

- All feasible solutions to the problem must be representable.
- A traversal path should exist between any two solutions in the search space.
- The representation should enable easy manipulation by the search operators.

Among the solution encodings that have been proposed and utilized in metaheuristic algorithms, we may cite:

- **Binary encoding**: Solutions are represented as binary bit vectors of length n, yielding a search space of size 2^n.
- **Discrete encoding**: Variables can take non-binary discrete values like integers within a range.
- **Real-valued encoding**: Variables are encoded as real numbers within specified bounds.
- **Quantum encoding**: Solutions are represented by quantum bits (qubits), which can represent a superposition of states. This compact encoding can represent an exponential search space.
- **Graph-based encoding**: Solutions are encoded via graphs or trees rather than linear vectors.

1.3.1.2 Objective function

The objective function, also called the fitness function, represents the mathematical representation of the goal or objective to be optimized by the metaheuristic. It allows the assignment of a real-valued quality score to each candidate solution in the

search space to distinguish good solutions from bad ones. Higher objective values indicate better solutions for maximization problems, while lower values are preferred in minimization problems. Hence, the fitness function plays a crucial role in guiding the metaheuristic search process towards high-quality solutions. A well-formulated objective function enables efficient search so that improving the objective value steers the search in the right direction.

1.3.1.3 Intensification and diversification

Exploration and exploitation stand out as essential attributes contributing to the success of metaheuristics when addressing an optimization problem. Exploration, also known as diversification, refers to the ability to discover diverse solutions across the search space. Exploitation, or intensification, means boosting the search for promising areas to improve existing solutions [133]. While exploitation enhances convergence speed, it risks getting stuck in local optima. Conversely, exploration increases the chances of finding global optima but slows convergence. A successful metaheuristic provides sufficient exploration to identify high-quality regions while exploiting those areas without getting trapped locally [17]. The main differences between metaheuristics involve how they achieve this vital balance.

1.3.1.4 Parameter tuning

As mentioned earlier, metaheuristics have various parameters that must be set or adjusted for optimal performance on a problem. The best parameter values are often problem-dependent and found through experimentation and statistical techniques.

Parameter tuning is essential to harness the flexibility of metaheuristics and avoid suboptimal results, but tuning complex algorithms with many parameters poses challenges. Two main approaches are used [95]:

1. **Ad-hoc pilot testing** : This manual tuning through trial-and-error is commonly used in practice but is time-consuming without rigor.
2. **Automatic configuration** : Tools like irace [77] and ParamILS [49] automate parameter tuning through statistical racing, grid search, etc. However, tuning with these tools can be computationally expensive for real-world problems.

1.3.2 Single-solution-based metaheuristics

Single-solution-based metaheuristics, also called trajectory methods, operate on a single-candidate solution. They move this solution throughout the search space, creating a

trajectory as an enhanced version of local search methods. By focusing the search on a single solution, they can thoroughly explore a promising area rather than diffusing across many solutions. However, this search strategy is sensitive to several potential drawbacks, including the risk of premature convergence and being stuck in local optima. Additionally, the initial solution significantly influences the effectiveness and quality of the final solution.

1.3.2.1 Simulated Annealing

Simulated Annealing (SA) is a stochastic optimization metaheuristic for solving optimization problems. It was first proposed in 1983 by Kirkpatrick et al. as an optimization technique [56]. The algorithm is inspired by the metallurgy process of physical annealing, where a material is heated and then slowly cooled to achieve a low-energy crystalline structure. The popularity of SA has grown due to its simplicity and effectiveness in handling discrete and continuous optimization problems.

The SA algorithm starts with a random initial solution and a temperature parameter that controls randomness. It randomly generates a neighbor solution at each step and evaluates its objective value. If the neighbor is better, it becomes the new current solution. However, worse neighbors may also be accepted with a certain probability based on the actual temperature and difference in objective value. The acceptance probability is calculated as follows:

$$p_i = \begin{cases} 1 & \text{if } f(y) \text{ better than } f(x) \\ \exp^{(f(y)-f(x))/t_i} & \text{otherwise} \end{cases} \tag{1.5}$$

where x and y are the current and new solutions, respectively. $(f(x) - f(y))$ is the objective difference and t_i is the temperature at iteration i.
According to Equation 1.6, the temperature parameter gradually decreases over iterations according to an annealing schedule. This cooling schedule controls how the temperature is reduced over time.

$$t_i = \alpha \times t_{i-1} \tag{1.6}$$

where α is the cooling rate between 0 and 1.

This fundamental mechanism allows SA to escape poor local optima early on by exploring widely. Later, as t cools, the focus shifts to intensification and convergence by mostly accepting only better moves. The cooling schedule balances exploration and exploitation to avoid entrapment in local optima. The SA algorithm is described

in Algorithm 1. However, SA can be sensitive to several potential limitations. Poor parameter settings and a low-quality initial solution can lead to slow convergence or getting trapped in local optima.

Algorithm 1 Simulated Annealing

Input: max_iter : max number of iterations, T: initial temperature, α: cooling rate, f : fitness function

Output: S^* : best solution found

Generate a random solution S
$S^* \leftarrow S$
$i \leftarrow 0$
while Not stopping criterion **do**
    Select a neighbor $S' \in N(S)$
    if $f(S')$ better than $f(S)$ **then**
        $S \leftarrow S'$
    else
        $S \leftarrow S'$ with probability $\exp^{(f(S')-f(S))/T}$
    end if
    if $f(S')$ better than $f(S^*)$ **then**
        $S^* \leftarrow S'$
    end if
    Update T using Equation 1.6
    $i \leftarrow i+1$
end while
return S^*

In SA, the time complexity depends on the number of iterations, the complexity of generating new solutions, and the complexity of evaluating the objective function [43]. Combining these factors, the general time complexity of SA can be expressed as:

$$T_{SA} = O(max\_iter \times (f(n) + g(n))) \tag{1.7}$$

where:

- max_iter refers to the maximum number of iterations.
- $O(g(n))$ represents the time complexity of generating a new solution depending on the neighborhood search strategy and problem size.
- $O(f(n))$ represents the time complexity of evaluating the objective function.

1.3.3 Population-based metaheuristics

Population-based metaheuristics operate on a set or population of multiple solutions rather than a single solution. Manipulating a population provides enhanced search space exploration compared to single-solution methods. The most studied population-based approaches are associated with Evolutionary Algorithms (EAs) and Swarm Intelligence (SI). EAs are inspired by biological evolution and genetics. The population represents evolving species. Solutions are combined and mutated over generations to improve fitness. SI is based on decentralized swarm behavior in nature. Interactions between solutions mimic social processes. Moreover, some newer population-based methods draw analogies to physics systems.

Population-based metaheuristics can be thought of as simulating the evolution of a population. The algorithm starts by generating an initial population of random solutions. Then, the population evolves over generations by selecting and variating the fittest solutions until a good solution is found or the stopping criteria are satisfied. However, a lack of transparency/interpretability can make population-based metaheuristics inconvenient or impractical for some problem domains. Next, we will discuss Grey Wolf Optimizer (GWO), Multi-Verse Optimizer (MVO), and Quantum-behaved Particle Swarm Optimization (QPSO) algorithms as illustrative examples.

1.3.3.1 Grey Wolf Optimizer

The Grey Wolf Optimizer (GWO) is a metaheuristic optimization algorithm inspired by the social hierarchy and hunting behavior of grey wolves in nature. Mirjalili et al. [84] proposed it in 2014 as a new swarm intelligence technique. The following will describe the inspiration for GWO and the mathematical model.

Inspiration

Grey wolves have a highly structured social hierarchy that controls pack behavior. The pack is led by dominant alpha wolves (α), who make vital decisions. Beta wolves (β) assist the alphas and communicate orders to the rest of the group. At the lowest rank are omega wolves (ω), who eat last after the others. The remaining wolves are delta wolves (δ) that serve diverse roles like scouting, hunting, and caring for the young. This social hierarchy is illustrated in Figure 1.4. Another notable trait of grey wolves is their cooperative hunting style. According to research, their attack process has three main phases: tracking, circling, and attacking the prey [89].

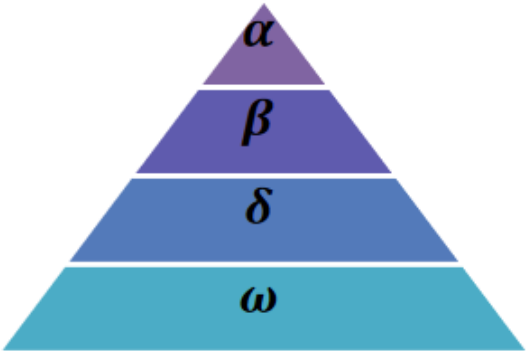


Figure 1.4: Hierarchy of grey wolf dominance decreases from top down [84].

Mathematical Model

This subsection presents mathematical models for the social hierarchy and hunting behavior, as introduced in [84].

- **Social hierarchy**: The population is structured based on a leadership hierarchy of α, β, and δ wolves, representing the top three best solutions. These leaders play a guiding role in searching for prey, while the remaining wolves are considered omega.
- **Encircling prey**: The wolves cooperatively track and encircle potential prey. Mathematically, this is modeled by updating each search agent's position towards the target position.

$$X_{(t+1)} = X_{p,t} - A \cdot D \tag{1.8}$$

$$D = |C \cdot X_{p,t} - X_t| \tag{1.9}$$

$$A = 2 \cdot a \cdot r_1 - a \tag{1.10}$$

$$C = 2 \cdot r_2 \tag{1.11}$$

where X_t denotes the position vector of a gray wolf at t^{th} iteration and $X_{p,t}$ represents the location of the prey at t^{th} iteration.D is a vector that mathematically models the hunting encirclement behavior employed by grey wolves during their hunting activities. A and C are coefficient vectors.r_1 and r_2 are random vectors between $[0,1]$ that belong to a uniform distribution, and a is a vector that decreases linearly from 2 to 0 along the iterations and which is described by Equation 1.12.

$$a = 2 - 2 \times (\frac{t}{T}) \tag{1.12}$$

where t denotes the present iteration, and T represents the maximum number of iterations.

- **Hunting**: Based on their knowledge, the alpha, beta, and delta wolves guide the pack towards promising areas. Each wolf updates its position accordingly using the following equations.

$$X_1 = X_\alpha - A_\alpha \cdot D_\alpha \tag{1.13}$$

$$X_2 = X_\beta - A_\beta \cdot D_\beta \tag{1.14}$$

$$X_3 = X_\delta - A_\delta \cdot D_\delta \tag{1.15}$$

$$X_{(t+1)} = \frac{X_1 + X_2 + X_3}{3} \tag{1.16}$$

- **Exploration and exploitation**: In nature, grey wolves strike a balance between exploring new territories to find potential prey and exploiting known hunting grounds. The GWO algorithm mimics this behavior by using random components A and C. Small absolute A values ($|A| < 1$) encourage exploitation of promising areas found so far. Large absolute A values ($|A|| > 1$) promote the exploration of new regions in the search space. Additionally, the random nature of component C contributes an element of exploration, allowing the algorithm to probe broadly.

The GWO algorithm starts with a randomly initialized population of grey wolf agents. Their fitness is evaluated, and alpha, beta, and delta wolves are identified. The hunting process proceeds for a defined number of iterations or until a termination criterion is met. The GWO method is sketched in Algorithm 2.

The time complexity of the GWO algorithm depends on several factors, including the population size, the problem size, the number of iterations, and the complexity of evaluating the objective function [1]. In general, the time complexity of the GWO algorithm for a one-dimensional problem can be expressed as:

$$T_{GWO} = O(T \times N \times f(n)) \tag{1.17}$$

where:

- T is the maximum number of iterations.
- N refers to the size of the population.
- $f(n)$ represents the time complexity of evaluating the objective function.

Algorithm 2 Grey Wolf Optimizer

Input: T : max number of iterations, n: population size, f : fitness function
Output: X_α

Initialize the grey wolf population $X_i\,(i = 1, 2, .., n)$
Evaluate the population
Select the leaders X_α, X_β and X_δ
$t \leftarrow 0$
while Not stopping criterion **do**
    Update a using Equation 1.12
    for each individual **do**
        Update the position of X_1, X_2 and X_3 using Equations 1.13, 1.14 and 1.15 respectively
        Update the position of the current individual by Equation 1.16
    end for
    Evaluate the new population
    Update the leaders X_α, X_β and X_δ
    $t \leftarrow t + 1$
end while
return X_α

1.3.3.2 Multi-Verse Optimizer

The Multi-Verse Optimizer (MVO) is a novel cosmology-inspired metaheuristic based on the concept of parallel universes and wormhole theory in cosmology. It was proposed by Mirjalili et al. [83] in 2016.

Inspiration

The multi-verse theory shows how big bangs produce multiple universes and how these universes interact with each other. Three concepts have been explored in the MVO algorithm: white holes, black holes, and wormholes. Black holes and white holes interact through a tunnel so that black holes attract everything, and white holes send things out. Meanwhile, a wormhole generates tunnels through time to connect various parts of the universe. Moreover, each of these universes is defined by a rate of inflation that makes it expand.

Mathematical model

In MVO, each candidate solution is modeled as a separate universe. The components of the multi-verse theory are translated into the optimization context as follows:

- Each candidate solution is modeled as a separate universe where all solution attributes represent objects in that universe.

- Wormholes allow the attributes of a solution to move to the best universe randomly.
- The attributes of the best solutions are moved to other solutions through black and white holes.
- Inflation rates define how fast universes expand based on their solution fitness.

The core of MVO is encapsulated in two update Equations 1.18 and 1.19.

$$X_i^j = \begin{cases} X_k^j & \text{if } r_1 < NI(U_i) \\ X_i^j & \text{else} \end{cases} \tag{1.18}$$

where X_i^j is the j_{th} parameter of the universe i and U_i is the universe i. $NI(U_i)$ denotes the rate of normalized inflation of the universe i_{th}, r_1 denotes a random value within $[0,1]$, and X_k^j represents the i_{th} parameter of the k_{th} universe chosen via a roulette selection system.

$$X_i^j = \begin{cases} \begin{cases} X^j + TDR \times ((ub_j - lb_j) \times r_4 + lb_j) & \text{if } r_3 < 0.5 \\ X^j - TDR \times ((ub_j - lb_j) \times r_4 + lb_j) & \text{else} \end{cases} & \text{if } r_2 < WEP \\ X_i^j & \text{else} \end{cases} \tag{1.19}$$

X^j denotes the j_{th} parameter of the best-found universe. lb_j and ub_j are the lower and upper limits of the j_{th} variable. r_2, r_3, r_4 are random numbers in $[0,1]$. TDR and WEP represent coefficients that are computed as follows:

$$WEP = min - t \times \frac{max - min}{T} \tag{1.20}$$

$$TDR = 1 - \frac{t^{\frac{1}{p}}}{T^{\frac{1}{p}}} \tag{1.21}$$

where max and min are constants, p is the exploitation coefficient. t denotes the current iteration, while T is the maximum iteration.

Algorithm 3 outlines the MVO method.

The time complexity of the MVO algorithm is generally related to the number of iterations (T), the number of universes (N), the number of objects (d), and the complexity of evaluating the objective function ($f(n)$). Equation 1.22 represents the time complexity of the algorithm [2].

$$T_{MVO} = O(T \times (N^2 + N \times d + logN + f(n))) \tag{1.22}$$

Algorithm 3 Multi-Verse Optimizer

Input: T : max number of iterations, n: number of universes, max, min, p, ub, lb ,f : fitness function

Output: u^* : best universe

Generate random universes U
Evaluate the fitness of all universes
Select u^*
SU = Sorted universes
NI = Normalize inflation rate (fitness) of the universes
$t \leftarrow 0$
while Not stopping criterion **do**
    Update WEP and TDR using Equations 1.20 and 1.21, respectively
    for each universe i **do**
        Black hole index = i
        for each object j **do**
            $r_1 = random([0,1])$
            if $r_1 < NI(U_i)$ **then**
                $k = rouletteWellSelection(NI)$
                $U(Black\ hole\ index, j) = SU(k, j)$
            end if
            $r_2 = random([0,1])$
            if $r_2 < WEP$ **then**
                $r_3 = random([0,1])$
                $r_4 = random([0,1])$
                if $r_3 < 0.5$ **then**
                $U(i,j) = bestuniverse(j) + TDR \times ((ub_j - lb_j) \times r_4 + lb_j)$
                else
                $U(i,j) = bestuniverse(j) - TDR \times ((ub_j - lb_j) \times r_4 + lb_j)$
                end if
            end if
        end for
    end for
    Evaluate the fitness of all universes
    Update u^*
    $t \leftarrow t + 1$
end while
return u^*

1.3.3.3 Quantum-behaved Particle Swarm Optimization

Quantum-behaved Particle Swarm Optimization (QPSO) is a variant of the Particle Swarm Optimization (PSO) algorithm proposed in 2004 by Sun et al. [112]. As a quantum-inspired metaheuristic, QPSO leverages quantum notions but is designed to run on classical computers, unlike quantum metaheuristics that require quantum hardware. QPSO uses concepts from quantum mechanics like particle position probability and the Schrodinger equation to guide the particles' search process.

General formulation

In the quantum time-space framework, the wave function $\Psi(\mathbf{X},t)$ describes the quantum state of a particle dependent on its position X and time t. In three-dimensional space, the wave function satisfies the following:

$$|\Psi(\mathbf{X},t)|^2 dx\ dy\ dz = Q\ dx\ dy\ dz \tag{1.23}$$

$Q\ dx\ dy\ dz$ is the probability that the particle will appear in the infinitesimal element around the point (x,y,z). In other words, $|\Psi(\mathbf{X},t)|^2 = Q$ represents the probability density function satisfying :

$$\int_{-\infty}^{+\infty} |\Psi(\mathbf{X},t)|^2 dx\ dy\ dz = \int_{-\infty}^{+\infty} Q\ dx\ dy\ dz = 1 \tag{1.24}$$

In the QPSO algorithm, each particle moves in a spin-less way in an N-dimensional Hilbert space with specified potential energy. For simplicity, first, consider $1D$ space where the particle's position is X, and the potential well center is p. The potential energy of the particle in the one-dimensional δ potential well is:

$$V(X) = -\gamma.\delta(X-p) = -\gamma.\delta(Y) \tag{1.25}$$

where Y denotes the intensity of the potential well.

We use the Monte Carlo method to derive the update function for each particle's position. Each dimension is bounded in a δ potential well and updated independently. Equation 1.26 is used to measure the j^{th} $(1 \leq j \leq N)$ component of the position of individual i $(1 \leq i \leq M)$ at the $(t+1)^{th}$ iteration. For a more comprehensive understanding, refer to [112].

$$X_{i,t+1}^{j} = p_{i,n}^{j} \pm \frac{L_{i,t}^{j}}{2} \times ln(\frac{1}{u_{i,t+1}^{j}}) \tag{1.26}$$

where $u_{i,t+1}$ is a sequence of random numbers uniformly distributed on $(0,1)$, varying for each i and j, and $L_{i,t}$ is determined by either of the following two equations:

$$L_{i,t} = 2\alpha|X^j_{i,t} - p^j_{i,t}| \tag{1.27}$$

$$L_{i,t} = 2\alpha|X^j_{i,t} - C^j_t| \tag{1.28}$$

where C^j_t is the mean best position of all particles. Thus, the position of the particle is updated by using either of the following two equations:

$$X^j_{i,t+1} = p^j_{i,n} \pm \alpha|X^j_{i,t} - p^j_{i,t}| \times ln(\frac{1}{u^j_{i,t+1}}) \tag{1.29}$$

$$X^j_{i,t+1} = p^j_{i,n} \pm \alpha|X^j_{i,t} - C^j_t| \times ln(\frac{1}{u^j_{i,t+1}}) \tag{1.30}$$

where α is a positive real number.

Algorithm 4 presents the steps involved in the QPSO algorithm.

The general time complexity of the QPSO algorithm for a one-dimensional problem can be expressed as defined in Equation 1.31 [130].

$$T_{QPSO} = O(T \times N \times f(n)) \tag{1.31}$$

where:

- T denotes the maximum number of iterations.
- N is the number of particles in the swarm.
- $f(n)$ refers to the time complexity of evaluating the objective function.

1.3.4 Parallel metaheuristics

Optimization problems are increasingly complex and resource-intensive. Though metaheuristics can reduce computational complexity, real-world problems often have huge search spaces and expensive objective functions, requiring significant CPU time and memory. Parallel computing offers an effective strategy to design performant parallel metaheuristics.

The proliferation of parallel hardware architectures, including multi-core CPUs, GPUs, and computing clusters, has enabled high-performance metaheuristics to tackle more complex optimization problems. Massively parallel platforms provide the computational power to accelerate metaheuristic search, evaluate solutions faster, and explore larger search spaces.

Algorithm 4 Quantum-behaved Particle Swarm Optimization

Input: T : max number of iterations, n: population size, α ,f : fitness function
Output: G : best position found

Initialize the population X
Initialize best personal best position of all the particles P
Evaluate the population
Select the G
$t \leftarrow 0$
while Not stopping criterion **do**
    Update C_t (for QPSO type 2)
    for each individual i **do**
        for each dimension j **do**
            $\sigma = random([0,1])$
            $P^j_{i,t} = \sigma P^j_{i,t} + (1-\sigma)G^j$
            $u^j_{i,t+1} = random([0,1])$
            update $X^j_{i,t+1}$ by using Equation 1.29 (for QPSO type 1) or by using Equation 1.30 (for QPSO type 2)
        end for
    end for
    Evaluate the the new population
    Update P
    Update G
    $t \leftarrow t+1$
end while
return G

When designing parallel metaheuristics, three main parallelization approaches operate at different levels [114]:

1. **Algorithm level**: Independent or collaborating metaheuristics run concurrently.
2. **Iteration level**: Each iteration of a metaheuristic is parallelized across processors.
3. **Solution level**: Individual solutions are evaluated concurrently.

Algorithm-level parallelism does not alter behavior but provides inter-algorithm parallelism. On the other hand, iterative-level parallelism and solution-level parallelism provide intra-algorithm parallelism focused on accelerating search by parallelizing costly steps.

1.4 Machine Learning

In 1959, IBM researcher Arthur Samuel published a pioneering paper introducing the modern Machine Learning (ML) concept. He defined *machine learning* as the field of study that allows computers to learn without being explicitly programmed [103]. This was a groundbreaking idea at the time. Samuel's paper built upon a longstanding debate in artificial intelligence traced back to Alan Turing in 1950. Turing wondered if computers could be capable of general learning and creativity beyond what programmers directly told them to do.

While Samuel's definition of ML remains influential today, the contemporary understanding of learning warrants clarification. More recently, Tom Mitchell has formulated a concise definition. This definition is as follows [85]:

Definition 1. A computer program is said to learn from experience E with respect to some task T and some performance measure P, if its performance on T, as measured by P, improves with experience E.

ML is an interdisciplinary field that draws on many research areas. It largely overlaps with statistics, as many ML algorithms have direct statistical equivalents. ML aims to automate and scale knowledge engineering, replacing time-consuming human activity with automatic data analysis techniques. It has provided powerful methods for tackling complex real-world challenges in business, education, healthcare, industry, and other sectors.

Broadly, ML can be divided into two main types: deductive and inductive. Deductive learning deduces new knowledge from existing facts and knowledge. On the other hand, inductive learning extracts patterns and rules from large datasets to create programs, generalizing from examples rather than starting with prior knowledge.

In the remainder of this section, we will introduce ML paradigms and fundamentals, as well as ML-based approaches for addressing COPs.

1.4.1 Machine Learning paradigms

Depending on how an algorithm learns, ML paradigms can be classified into four categories. As illustrated in Figure 1.5, there are four fundamental learning methodologies: supervised learning, unsupervised learning, semi-supervised learning, and reinforcement learning. The paradigm choice depends on diverse factors like data availability, objective, and output type. The following subsections provide brief explanations of these paradigms along with examples of algorithms.

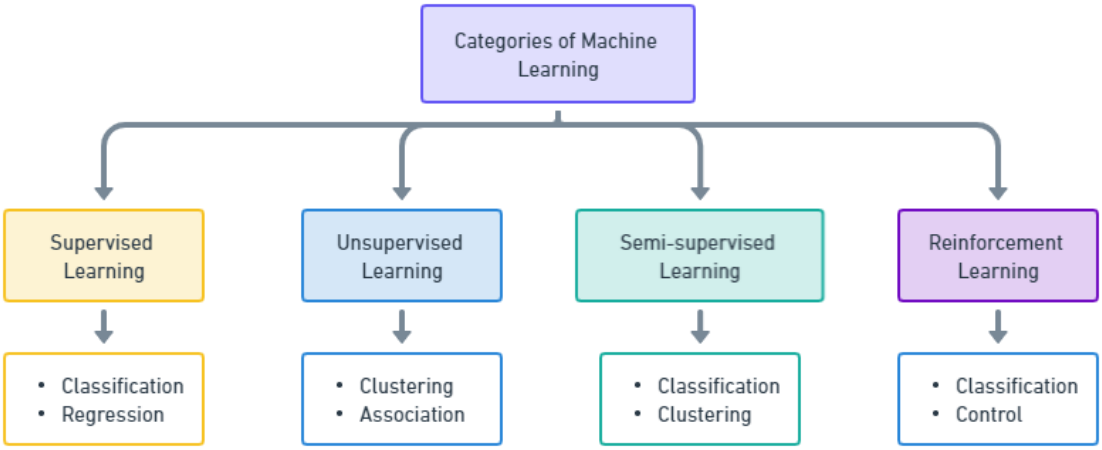


Figure 1.5: Machine Learning categories (adapted from [115]).

1.4.1.1 Supervised learning

Supervised learning is the process of learning a function that maps an input to an output based on sample input-output pairs. Thus, the supervision in the learning comes from the labeled examples in the training dataset [42]. After seeing examples during training, the goal is to accurately predict the output for new unlabeled inputs.

Supervised learning problems are broadly categorized into two main types: classification and regression. Classification involves assigning discrete outputs to inputs by grouping them into distinct classes (binary or multiclass scenarios). Popular classification algorithms include Logistic Regression (LR), Support Vector Machines (SVMs), k-Nearest Neighbors (k-NN), decision trees, Random Forests, and Artificial Neural Networks (ANNs). Regression focuses on predicting continuous numerical outputs by fitting functions to data points. Standard regression algorithms are linear regression, linear multiple regression, and polynomial regression.

Logistic Regression

Logistic Regression (LR) is a foundational classification technique in machine learning. It is commonly used for binary classification problems, where the output variable can only take two values, typically represented as 0 or 1. Given a set of input features, it estimates the probability that an observation belongs to a particular class. LR models this probability using the logistic function, also called the sigmoid function [57]:

$$p(y=1|x) = \frac{1}{1+e^{-z}} \tag{1.32}$$

where $p(y = 1|x)$ is the probability that the output variable y equals one given the input features x, and z is the logit function.

$$z = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + ... + \beta_n x_n \tag{1.33}$$

The β coefficients are estimated from training data using Maximum Likelihood Estimation (MLE).

One of the main limitations of LR is the assumption of linearity between the dependent and independent variables. This linearity assumption may only apply to some datasets and real-life problems. The relationship between variables may be more complex and non-linear in nature. Forced linearity can lead to poor model fit and limited predictive ability when the true relationships are non-linear.

k-Nearest Neighbors

k-Nearest Neighbors (k-NN), known as a lazy learning algorithm, is a simple supervised learning technique for classification and regression [3]. It represents a type of instance-based learning algorithm. It does not construct an internal model but instead stores all training instances in n-dimensional space. k-NN looks at the k closest stored training examples to predict new data points based on a similarity measure like Euclidean distance. The prediction for a new data point is then made based on the majority class or the average of the k nearest data points in the feature space. k-NN is relatively robust to noisy training data, with performance dependent on data quality [105].

Although k-NN is a simple and efficient learning technique, the choice of k value can significantly impact the algorithm's performance. Another drawback of k-NN is its computational inefficiency, especially for large datasets, as it requires storing and searching the entire training dataset for each prediction. In addition, it may not perform well in high-dimensional spaces or when the feature space is not well-defined.

Support Vector Machine

Vladimir Vapnik and his colleagues Bernhard Boser and Isabelle Guyon introduced Support Vector Machine (SVM) in 1992 [18], based on Vapnik and Alexey Chervonenkis' work on statistical learning theory in the 1960s[117]. However, the modern SVM formulation was developed at Bell Labs and published in 1995 by Vladimir Vapnik and Corinna Cortes [26].

SVMs aim to solve classification problems by finding optimal decision boundaries separating data points from different classes. The data is mapped to a high-dimensional space, where these boundaries can be defined by a hyperplane that maximizes the

margin or distance to the nearest points (see Figure 1.6). The kernel trick enables efficient computing distances in this space without explicitly transforming the data, using kernel functions like linear, polynomial, radial basis function, and sigmoid. Different kernels lead to different decision boundary shapes.

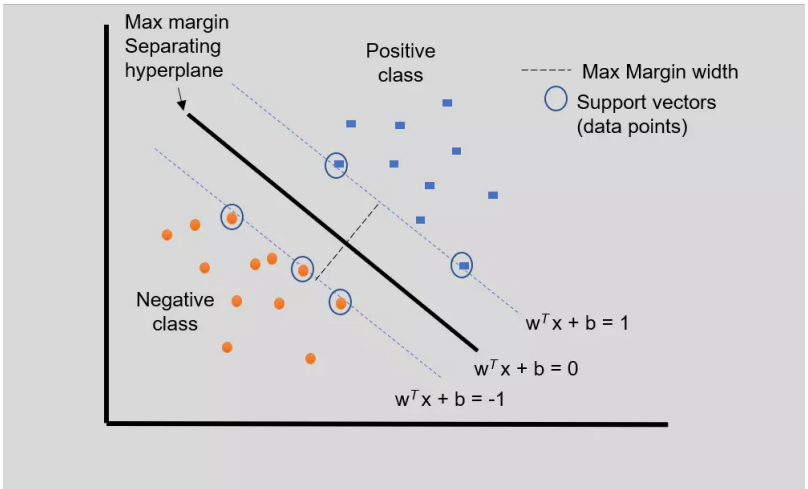


Figure 1.6: Support Vector Machine for binary classification.

When first developed, SVMs achieved state-of-the-art accuracy on simple classification tasks and had solid theoretical foundations. However, they are computationally intensive for large datasets. SVMs also initially struggled with perceptual problems like image classification that require hand-engineered feature extraction.

Artificial Neural Networks and Deep Learning

Artificial Neural Network (ANN) are computational models inspired by biological neural networks in animal brains. They consist of interconnected nodes or neurons organized into layers: an input layer, one or more hidden layers, and an output layer, as shown in Figure 1.7. Each connection between neurons has an associated weight. Neurons apply activation functions to transform their weighted input signals into an output signal that is passed to downstream neurons. Common activation functions include sigmoid, hyperbolic tangent (tanh), and rectified linear unit (ReLU).

ANNs learn by adjusting these weights through a training phase, typically using supervised learning from input-output example pairs. The network makes predictions in the forward pass, compares them to desired targets, and then back-propagates errors through the layers to update weights and minimize prediction errors. This iterative adjustment enables ANNs to learn complex concepts and make accurate predictions.

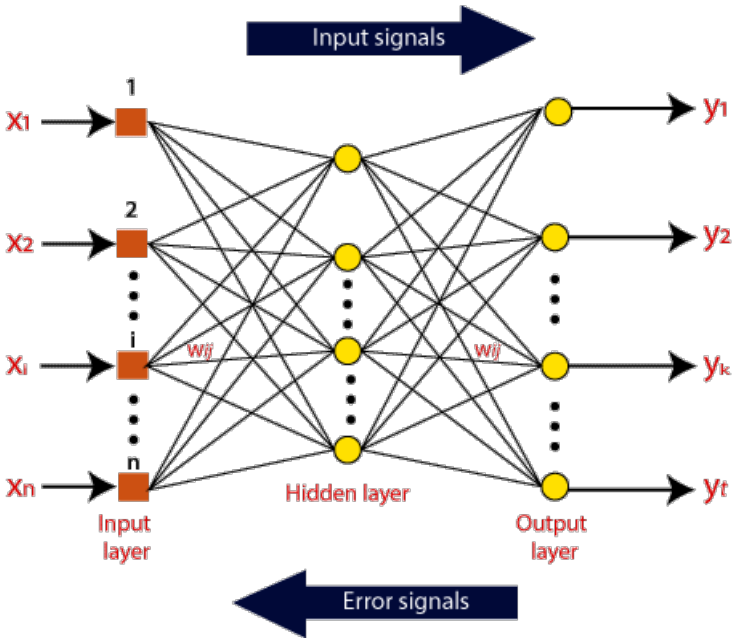


Figure 1.7: Architecture of ANN.

Deep Learning (DL) refers to neural networks with multiple hidden layers that can automatically learn hierarchical data representations. The term "deep" describes the depth of the network architecture, with many layers between the input and output. DL has become prominent because it can handle large amounts of data and extract complex features without manual feature engineering. This makes it well-suited for tasks with large datasets and high complexity.

Though DL has long theoretical roots, it rose to prominence in the early 2010s, driven by advances in hardware, computing capabilities, and algorithms. DL has achieved revolutionary results on skills that mimic human intuition but have long-challenged machines like vision, language, and speech. However, deep learning models require abundant training data to reach high performance.

Typical deep learning neural network architectures designed for supervised learning include Deep Neural Networks (DNNs), which are ANNs with multiple hidden layers; Convolutional Neural networks (CNNs), which specialize in the processing of grid-structured data, such as images; and Recurrent Neural Networks (RNNs), which are designed for sequential data.

a Extreme Learning Machine

The Extreme Learning Machine (ELM) is a machine learning algorithm that falls under supervised learning. Proposed first in 2004 by Huang et al. [46]. It represents a

feedforward neural network with a singular hidden layer. ELM has found applications in both classification and regression.

Unlike traditional neural networks, where the weights are iteratively adjusted during training using the back-propagation algorithm, the ELM's input weights and hidden biases are randomly initialized and remain fixed. Since the hidden layer weights are randomly generated and fixed, the training phase involves only adjusting the weights in the output layer. The output weights are analytically determined using the Moore-Penrose generalized inverse. This simplifies the training procedure and accelerates convergence.

Given a training set (x_i, t_i) with input vectors x_i and output vectors t_i, an ELM with N hidden nodes can be modeled as:

$$\sum_{i=1}^{N} \beta_i f_i(x_j) = \sum_{i=1}^{N} \beta_i f_i(a_i \cdot x_j + b_i) \tag{1.34}$$

where a_i and b_i are random input weights and biases, f is the activation function, "$\cdot$" represents the scalar product, and β_i are the output weights connecting the i^{th} hidden node with the output nodes. We can reduce the equation and write it as follows:

$$H \cdot \beta = T \tag{1.35}$$

H is the hidden layer output matrix, and T is the training data target matrix. The output weights β can be analytically determined by finding a least-square solution as follows:

$$\check{\beta} = H^{+} \cdot T \tag{1.36}$$

where H^{+} is the Moore–Penrose generalized inverse of H. Thus, ELM training involves:

1. Randomly initialize input weights and hidden biases.
2. Calculate hidden layer output matrix H.
3. Analytically determine output weights $\check{\beta}$ using the pseudoinverse of H.
4. $\check{\beta}$ will be used to make predictions on the test set.

The random initialization of weights and biases in ELM contributes to its fast learning speed. However, this can negatively impact performance on complex classification tasks by affecting the learned class boundaries. As a result, ELM needs more hidden nodes than other techniques to achieve good generalization, which increases learning time. Additionally, ELM struggles with complex visual data when raw pixel spectral values are provided as inputs [100].

b Convolutional Neural Network

Convolutional Neural Networks (CNNs) are feedforward networks well-suited for processing grid-structured data like images. Inspired by the work of Hubel and Wiesel in 1962 on simple and complex cells, Kunihiko Fukushima developed the initial model called the "neocognitron" [37]. Since then, several architectures have been proposed, such as LeNet-5 [66], AlexNet [63], VGG [107]. CNNs have two principal components: a convolutional part and a classification part, as shown in Figure 1.8.

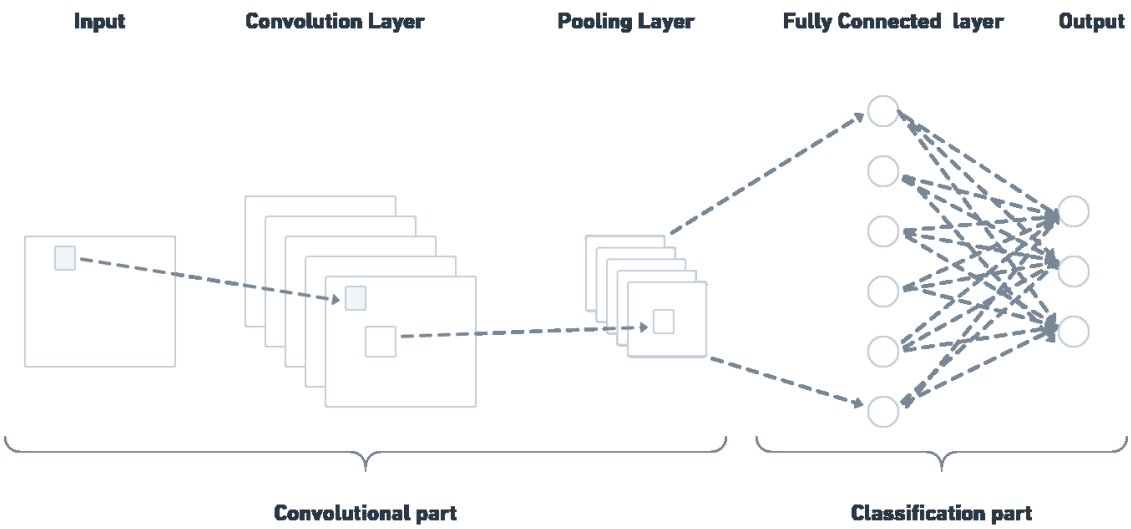


Figure 1.8: Standard CNN architecture (adapted from [116]).

The convolutional part retrieves the image's features by compressing it to reduce its initial size. It mainly consists of two elements: the convolution and pooling layers. The convolution layers apply a set of learnable filters, also called kernels, that extract spatial features through convolution operations. The convolution operation represents the dot product between the input matrix and the kernel. It can be expressed in mathematical terms as follows:

$$z_{i,j,k}^{[l]} = w_k^{[l]} x_{i,j}^{[l]} + b_k^{[l])} \tag{1.37}$$

where $z_{i,j,k}^{[l]}$ is the value of kernel output k_{th} at location i, j of layer l. $x_{i,j}$ is the input x at location i, j. The filter weights are denoted by w, and b represents the bias. Then, we apply the activation function to $(z_{i,j,k}^{[l]}$.

$$g_{i,j,k}^{[l]} = g(z_{i,j,k}^{[l]}) \tag{1.38}$$

Pooling downsamples the convolutional outputs to reduce spatial dimensions and complexity by applying a statistical measure such as maximum, mean, and median.

$$y_{i,j,k}^{[l]} = pooling(g_{i,j,k}^{[l]}) \tag{1.39}$$

The classification part corresponds to a Multi-Layer Perceptron (MLP) model. It comprises one or more fully connected layers and an output layer, usually softmax, to generate class probability predictions.

1.4.1.2 Unsupervised learning

Unsupervised learning involves finding patterns and structures in unlabeled input data without human guidance [42]. In contrast to supervised learning, there is no right or wrong output for each input item. The primary objective of unsupervised learning is to find insights and relationships within the data. Unlabeled data is abundant, making unsupervised learning crucial for mining knowledge from big data across domains like marketing, biomedicine, and multimedia.

Unsupervised learning is vital to exploratory data analysis, helping us understand datasets' properties. By recognizing inherent structures in the data, unsupervised learning can identify meaningful representations, compressions, visualizations, and groupings. Common unsupervised learning tasks include clustering, dimensionality reduction, association rule mining, and density estimation.

Of all the unsupervised learning techniques, this thesis focuses on dimensionality reduction and association rule extraction. Consequently, we will explore a selection of widely recognized algorithms adapted to these particular tasks.

Convolutional Autoencoder

A Convolutional Autoencoder (CAE) is a type of autoencoder network, a class of ANN used for unsupervised learning. Autoencoders aim to learn efficient representations of input data by encoding and decoding it [82]. The convolutional variant, CAE, is specifically designed for processing grid-like data such as images. CAEs compress input images into low-dimensional representations and then reconstruct the outputs to resemble the original inputs. This allows dimensionality reduction and denoising while retaining the most relevant visual features.

Like traditional autoencoders, CAEs consist of an encoder network that extracts features and compresses the input image into a lower-dimensional representation and a decoder network that reconstructs the image from the compressed representation

to generate an output similar to the original input. Figure 1.9 depicts the basic architecture of the CAE model.

Figure 1.9: Basic architecture of CAE model (adapted from [96]).

In addition to standard convolutional and pooling layers, CAEs have deconvolutional and unpooling layers in the decoder. The deconvolutional and unpooling layers perform the opposite operations of the convolutional and pooling layers, respectively, to allow the reconstruction of each subregion's original size. Note that the deconvolution operation is, in fact, the transposition operation.

Frequent Pattern growth

Frequent pattern mining is a crucial task in data mining, especially within the domain of association rule mining. The primary focus of frequent pattern mining is discovering itemsets that co-occur frequently across a collection of transactions. However, association rule mining extends beyond this by uncovering meaningful and statistically significant relationships between these frequently occurring itemsets and expressing them in the form of association rules.

Frequent Pattern growth (FP-growth) is a popular algorithm for mining frequent itemsets and extracting association rules from transactional databases. Pintelas et al. [96] proposed it in 2000 as an efficient and scalable method for mining frequent itemsets without candidate generation.

Prior algorithms like Apriori relied on iteratively generating and testing candidate itemsets, which is expensive for large datasets. FP-growth uses a divide-and-conquer approach to decompose the mining task into smaller ones without candidate generation. This makes FP-growth particularly well-suited for handling large transactional databases.

The FP-tree (Frequent Pattern tree) data structure is the key innovation in FP-growth. After two scans of the dataset, an FP-tree is constructed, which retains all the essential information about frequent patterns. Specialized branching rules are used to build the compressed dataset representation in the FP-tree. Each node stores an item and a count, with a path representing an itemset transaction. Once the

FP-tree is constructed, frequent itemsets can be extracted efficiently by recursively mining conditional FP-trees. After obtaining frequent itemsets, association rules can be generated based on support and confidence thresholds[1].
A refined extension of the FP-growth algorithm, FP-Max was designed to identify only maximal frequent itemsets efficiently.

1.4.1.3 Semi-supervised learning

Semi-supervised learning is a machine learning paradigm that utilizes both labeled and unlabeled data during training [42]. It lies between supervised learning and unsupervised learning. In semi-supervised learning, only a subset of the data is labeled, while the majority remains unlabeled. The labeled data is used to train the model on known patterns, while the unlabeled data assists in capturing the underlying structure or patterns in the entire dataset. Semi-supervised learning applications include classification, regression, prediction, and clustering. It is commonly used in text classification, machine translation, data labeling, anomaly detection, and other domains where labeled data is limited.

1.4.1.4 Reinforcement Learning

Reinforcement Learning (RL) is an advanced algorithm category in machine learning that enables software agents and machines to automatically evaluate the optimal behavior in a particular context or environment to improve efficiency, i.e., an environment-driven approach [52]. In contrast to supervised and unsupervised learning, RL continually refines its model by leveraging feedback from previous iterations. This learning paradigm is based on reward or penalty, aiming to utilize insights from interactions with the environment to make decisions that maximize rewards or minimize risks. RL can explore an environment, understand the dynamics, and learn an optimal policy (strategy) based solely on the reward signal. This makes it applicable to complex problems like robotics and games. Fundamental RL algorithms used include Q-Learning, SARSA, and Deep Q-Networks.

[1] Support threshold - The minimum support value required for an itemset to be considered frequent and included in the mining process.
Confidence threshold - The minimum confidence value required for generating an association rule from a frequent itemset.

1.4.2 Machine Learning basics

1.4.2.1 Data preprocessing

Data preprocessing is a crucial step in the ML workflow before feeding data into a model. It involves various techniques to improve the data's quality, consistency, and compatibility with the chosen ML algorithms. Proper preprocessing enhances model performance. Common preprocessing steps include [42]:

- Data cleaning to handle missing values, noise, outliers, and other errors in the data.
- Data transformation like normalization and scaling to standardize the ranges of features/variables.
- Feature selection to select the most relevant attributes and reduce dimensionality.
- Handling categorical variables through encoding schemes such as one-hot encoding that transform categories into numeric formats.

1.4.2.2 Hyperparameters

Hyperparameters are the configuration settings before training the model. They are called hyperparameters to distinguish them from the learnable parameters (weights and biases) that are optimized during training [40]. Each ML algorithm has a hyperparameter set that determines the model architecture and impacts the training process and performance. Hyperparameters can be grouped into three main categories:

1. **Architecture hyperparameters**: These define the model structure, including the number of layers, neurons per layer, and trees for random forests.
2. **Optimization hyperparameters**: These control the optimization techniques used for training, such as learning rate, batch size, and number of iterations.
3. **Regularization hyperparameters**: These control the regularization techniques applied to prevent overfitting (see Section 1.4.2.3).

However, selecting proper hyperparameters involves exploring a high-dimensional space, making manual tuning impractical due to the vast number of possibilities. Thus, several hyperparameter optimization approaches have been proposed to automate this process, including grid search, random search, Bayesian optimization, and metaheuristic algorithms.

1.4.2.3 Performance evaluation

Estimating ML model performance is critical for assessing generalization on future unseen data. Performance estimation also drives experimentation, such as hyperpa-

rameter tuning and algorithm selection. However, proper evaluation techniques are required based on goals and constraints like dataset size. This sub-section will cover some key concepts involved in the rigorous and appropriate evaluation of ML models.

Evaluation metrics

The evaluation metrics should be chosen and applied based on the ML problem type. Different metrics are appropriate for classification, regression, clustering, and other tasks. Each metric provides a different perspective on prediction accuracy, sensitivity to outliers, and variance explained. Combining these evaluation metrics provides a more comprehensive view of model performance.

a **Classification evaluation metrics**

Different classification metrics may be more suitable depending on factors like class imbalance. No single metric gives a complete picture, so often, a suite of metrics is used for rigorous comparative evaluation. Four terms are essential in computing evaluation measures: the number of true positive predictions (TP), true negative predictions (TN), false positive predictions (FP), and false negative predictions (TP). Standard classification evaluation metrics include:

- **Accuracy**: It is the proportion of correctly classified instances among all instances as defined in Equation. This provides an overall measure of how many instances the model classified correctly. Accuracy is intuitive but can be misleading if classes are imbalanced.

$$Accuracy = \frac{TP + TN}{TP + FP + FN + TN} \tag{1.40}$$

- **Error rate (misclassification rate)**: This provides another overall measure of how many instances the model got wrong, similar to accuracy but focusing on incorrect predictions. Its formula is given by Equation.

$$Error\_rate = 1 - Accuracy \tag{1.41}$$

- **Precision**: It is the proportion of the correct classified instances among all instances predicted as positive. It is useful when false positives are costly. Its mathematical formula is defined as follows:

$$Precision = \frac{TP}{TP + FP} \tag{1.42}$$

- **Recall**: It is the proportion of true positive instances among all actual positive instances. It is useful when false negatives are costly. The following equation determines the recall:

$$Recall = \frac{TP}{TP + FN} \tag{1.43}$$

- **F1-score**: It is the harmonic mean of precision and recall, providing a balance between the two metrics. The formula for calculating the F1-score is provided in Equation 1.44.

$$F1 - score = 2 \times \frac{Precision \times Recall}{Precision + Recall} \tag{1.44}$$

- **ROC AUC**: The Receiver Operating Characteristic (ROC) curve and Area Under the Curve (AUC) are evaluation metrics commonly used to assess the performance of binary classification models, particularly in scenarios where the class distribution is imbalanced. The ROC Curve plots the true positive rate against the false positive rate at different classification thresholds, showing the tradeoff between true positives and false positives. The AUC represents the area under the ROC curve and provides a single scalar value to quantify the model's ability to distinguish between positive and negative instances.

- **Log loss (logarithmic loss /cross-entropy loss)**: It provides a nuanced evaluation of predicted probabilities rather than just hard classifications. It is useful for imbalanced classes and probabilistic classifiers like neural networks. Minimizing log loss encourages high confidence in correct classes. Log loss complements accuracy and provides detailed insights into prediction quality. Its formula is depicted in Equation 1.45.

$$Log\ Loss = -\frac{1}{N}\sum_{i=1}^{N}(y_i log(p_i) + (1 - y_i) log(1 - p_i)) \tag{1.45}$$

  where N is the total number of instances in the dataset, y_i is the actual class label for the i^{th} instance (0 or 1 for binary classification or one-hot encoded vector for multiclass classification), and p_i is the predicted probability of the positive class for the i^{th} instance.

b **Regression evaluation metrics**

Various evaluation metrics can assess the performance of regression models that predict continuous numerical values. Standard regression evaluation metrics include:

- **Mean Squared Error (MSE)**: It measures the average of the squares of the differences between predicted and actual values, illustrated in Equation 1.46. MSE emphasizes large errors, making it more sensitive to outlier predictions that differ significantly from the true values. However, MSE may disproportionately punish models with a few incorrect predictions, even if most predictions are close.

$$MSE = \frac{1}{N}\sum_{i=1}^{N}(y_i - \widehat{y}_i)^2 \tag{1.46}$$

  where N denotes the number of samples. y_i and $\widehat{y}_i$ correspond to the actual and predicted values for the i^{th} sample.

- **Root Mean Squared Error (RMSE)**: It takes the square root of MSE. By taking the square root, RMSE returns the error to the same units as the target variable, making it easier to interpret.

$$RMSE = \sqrt{MSE} \tag{1.47}$$

- **Mean Absolute Error (MAE)**: It calculates the average of the absolute differences between each predicted value and actual value, as shown in Equation 1.48. Taking the absolute value of individual errors before averaging removes the exaggerating effect of squaring in MSE. This makes MAE less sensitive to outliers compared to MSE or RMSE. Also, MAE has the original units of the target variable, making it more interpretable.

$$MAE = \frac{1}{N}\sum_{i=1}^{N}|y_i - \widehat{y}_i| \tag{1.48}$$

- **R-squared (R^2)**: It measures the proportion of variance in actual values explained by the model. It ranges from 0 to 1, with higher values indicating better model fit. The equation for R^2 is as follows:

$$R^2 = 1 - \frac{SS_{res}}{SS_{tot}} \tag{1.49}$$

  where SS_{res} is the sum of squares of residuals (or errors) from the regression line, and SS_{tot} is the total sum of squares, representing the variance of the dependent variable.

Cross-validation

Cross-Validation (CV) is a technique in ML for evaluating model performance. It involves splitting the available data into different subsets called folds. The model is trained on a subset of the data (the training set) and then evaluated on the remaining data (the validation set). This process is repeated multiple times, with each fold used as training and validation data at different times. CV is commonly used for tasks like model selection, hyperparameter tuning, and estimating the overall performance of ML models.

The most frequent type of CV is k-fold cross-validation. Here the data is divided into k equal-sized subsets. The model is trained k times, each time using $k-1$ of the folds as training data and the remaining subset as the validation data. The performance metrics from each iteration are averaged to produce a robust estimate of the model's capabilities.

Overfitting and underfitting

A key challenge in ML is that models must perform well not just on the data they are trained on but also on new, unseen data. This ability to make accurate predictions for unobserved inputs is called generalization. Overfitting and underfitting describe two potential issues that can arise when training and evaluating ML models.

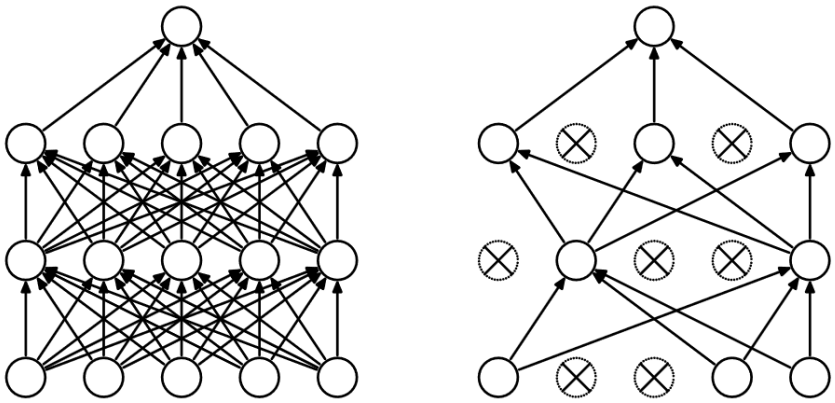
Overfitting, also called High variance, occurs when a model fits the training data too closely, capturing noise and random fluctuations rather than learning generalizable patterns. Overfitted models perform very well on training data but poorly on unseen examples. This happens when models are too complex relative to the amount and diversity of training data.

In contrast, *underfitting* (also referred to as "high bias") occurs when a model cannot fit the training data or generalize to new data. It arises when models are too simple relative to the complexity of the data, or there is insufficient training data.

Addressing these two issues involves balancing model complexity to find the spot between underfitting and overfitting. Several strategies have been proposed to prevent overfitting in ML models. The primary approach consists of using CV for model selection and choosing the optimal model complexity that generalizes best to new data. Other common techniques to reduce overfitting include early stopping, data augmentation [60], L1 regularization, L2 regularization (weight decay) [93], and dropout [111]. For underfitting, solutions involve using more complex models, collecting more training data, and engineering more/better features.

a **Dropout regularization**

Dropout is a widely used regularization technique for neural networks that reduces overfitting [111]. The key idea is to train a different network by randomly dropping neurons in each iteration.



(a) Standard Neural Net (b) After applying dropout.

Figure 1.10: Dropout Regularization [111].

In a standard neural network with L layers, $z(l)$ is the input vector to layer l, and $y(l)$ is the output vector. $W(l)$ and $b(l)$ are the weights and biases. The feedforward operation is described in Equations 1.50 and 1.51.

$$z_i^{(l+1)} = w_i^{(l+1)} y^l + b_i^{(l+1)} \tag{1.50}$$

$$y_i^{(l+1)} = f(z_i^{(l+1)}) \tag{1.51}$$

where f is the activation function.

With dropout, some neurons are randomly deactivated during training based on a Bernoulli distribution with probability p. The feedforward equations become:

$$r_i^{(l)} \sim Bernoulli(p) \tag{1.52}$$

$$\tilde{y}^{(l)} = r^{(l)} \times y^{(l)} \tag{1.53}$$

$$z_i^{(l+1)} = w_i^{(l+1)} \tilde{y}^{(l)} + b_i^{(l+1)} \tag{1.54}$$

$$y_i^{(l+1)} = f(z_i^{(l+1)}) \tag{1.55}$$

The weights are scaled by p at test time to account for the missing neurons during training.

$$W_{test}^{(l)} = pW^{(l)} \tag{1.56}$$

1.4.3 Machine Learning for COPs

Traditional optimization approaches work well for COPs but require revision when the problem formulation changes even slightly. In contrast, ML methods can potentially be applied across many optimization tasks by automatically learning heuristics from training data, reducing the need for manual engineering. Although ML for COPs dates back to the 20^{th} century [110], this direction was primarily abandoned after 2000 due to negative results. Recently, advances in ML have led to renewed interest in these approaches.

Motivated by the sequence-to-sequence model, Vinyals et al. [119] proposed an end-to-end model for combinatorial optimization called Pointer Network (PN). This neural network was trained in a supervised manner to predict optimal solutions for three complex problems: convex hulls, Delaunay triangulations, and the Travelling Salesman Problem (TSP). Despite these architectural improvements, supervised training is limited since optimal labels are unavailable for many COPs. Thus, Bello et al. [15] used RL to train the PN model. Graph neural networks (GNNs) have also been applied to graph-based problems like TSP [97].

Despite their promise, DL and RL have limitations for COPs. DL models require substantial data and may have poor generalization performance. Moreover, RL is often computationally expensive, and numerous simulations are needed to learn good policies.

1.5 Chapter summary

In summary, this chapter provided a comprehensive overview of the most important concepts related to this thesis. We reviewed the different metaheuristic and ML approaches, highlighting their roles in tackling real-world challenges. In addition, we discussed how each of these two techniques can be adapted to solve complex optimization problems, particularly COPs. The next chapter will focus on the synergy between ML and metaheuristics for solving complex problems, drawing on recent work.

Chapter 2

Synergy between ML and Metaheuristics

2.1 Introduction

In recent years, ML and metaheuristic algorithms have gained significant interest as powerful tools for tackling complex problems across diverse domains. While ML techniques excel at extracting patterns and insights from data, metaheuristics offer robust optimization strategies for navigating complex and high-dimensional search spaces. By combining these two paradigms, researchers can leverage the strengths of each to develop more intelligent and efficient hybrid techniques. The synergy between ML and metaheuristics represents a promising avenue for enhancing problem-solving capabilities.

On the one hand, defining a ML model requires further research, as most of the techniques used are based essentially on trial-and-error strategies. By applying metaheuristic algorithms, we can enhance the overall performance of ML models. On the other hand, real-world optimization problems often exhibit high-dimensional, non-linear, and dynamic characteristics, making them challenging for conventional algorithms alone. Integrating data-driven ML models into metaheuristic frameworks allows harnessing learned insights to guide and inform the optimization process for more effective solutions. Broadly, the integration involves two main directions: 1) using metaheuristics to improve ML model performance and 2) incorporating ML to enhance metaheuristic performance, as depicted in Figure 2.1.

This chapter will provide a comprehensive overview of the symbiotic relationship between ML and metaheuristics, highlighting how they can complement each other. We will explore this synergy's various hybrid approaches and methodologies, including

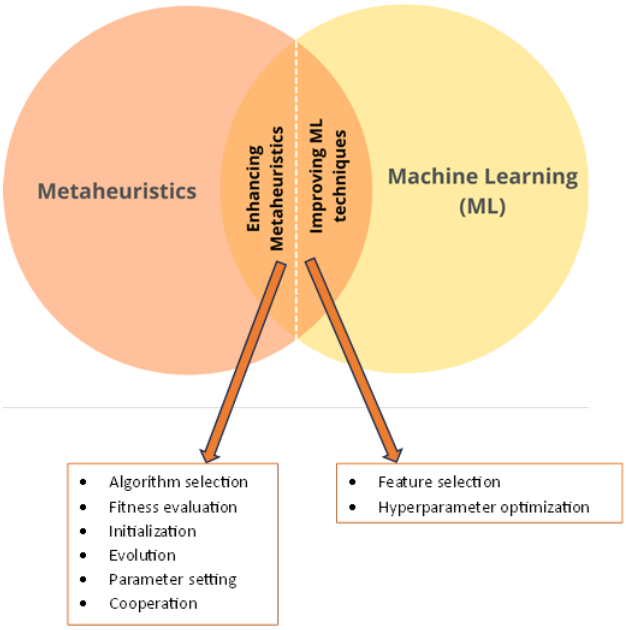


Figure 2.1: Combining ML and Metaheuristics (adapted from [22]).

utilizing metaheuristic techniques to enhance ML models and integrating ML models into metaheuristic algorithms to improve optimization processes, as well as challenges and perspectives.

2.2 Metaheuristics for enhancing ML

Recent advances in ML have enabled models to achieve success across many applications in different fields, but some key challenges can affect their overall performance. Two key issues are input feature selection and hyperparameter tuning. Traditionally, these tasks have demanded extensive manual effort and expertise. Thanks to their efficient search space exploration, metaheuristic algorithms have emerged as a promising approach to automate these tasks. Though promising, realizing the full benefits of automated feature selection and hyperparameter optimization with metaheuristics requires careful design of the search problem representation and operators.

2.2.1 Feature selection

Feature selection is a critical step in the ML pipeline to identify the most relevant subset of features from the original set. This process reduces data dimensionality, removes redundant and irrelevant features, decreases model complexity, and improves

performance. However, determining the optimum feature subset in high-dimensional feature datasets is considered an NP-hard problem. The search space will expand exponentially as the number of features rises. Consequently, metaheuristic algorithms are well-suited to identify feature subsets, as exact algorithms are intractable.

The feature selection is considered as a binary COP. Here, each solution is represented as a binary vector indicating whether each feature is selected (1) or not (0). The length of the vector corresponds to the total number of features in the dataset. Metaheuristics aim to identify the optimal feature subset by minimizing or maximizing an objective function that evaluates subset quality, such as error rate or classification accuracy.

Stochastic Local Search (SLS) was applied in [19] to improve intrusion detection by iterative exploration of neighbor solutions. The objective function was classification accuracy. The proposed method uses SLS for feature selection and then applies Naive Bayes and Bayesian Network classifiers to the selected features. With the Bayesian Network, the proposed SLS achieved 98.35% accuracy using only 22 of 41 features, demonstrating its effectiveness.

The authors of [29] combined statistical filters with multi-objective metaheuristic NSGA-II for cyber-attack detection in Internet of Things (IoT) networks. The filters guide NSGA-II population initialization, enabling faster convergence. Here, two objectives were used - maximizing classification accuracy while minimizing the number of features. The SVM classifier's accuracy on the selected features was used as the fitness function. This hybrid approach leveraged the low complexity of filters and the optimization power of NSGA-II to find an accurate and low-dimensional feature subset for effective attack detection. It provided an optimized, low-dimensional input feature set for the SVM classifier to detect attacks with high accuracy.

A variety of metaheuristic algorithms have been explored for automating feature selection, such as SA [7], Firefly Algorithm (FA) [125], GWO [98], and Bat Algorithm (BA) [35]. However, this remains an active area of research as finding an effective problem representation and search operators tailored to feature selection is critical to the success of metaheuristics.

2.2.2 Hyperparameter optimization

While ML has revolutionized many fields, selecting optimal hyperparameter values remains challenging for most ML techniques. Hyperparameter optimization is a process used to improve the efficiency of tuning ML models. It enhances the model performance and generalization capability, identifies ideal configurations, and enables developers to

concentrate on algorithms rather than tuning. Hyperparameter optimization can be defined as minimizing the error on the test set by finding the optimal hyperparameter values, as shown in Equation 2.1 [70].

$$\lambda_h^* = \underset{\forall \lambda_h \in \wedge}{arg\ min}\, Err(x_{test}, F_{network}(x_{train}, A_{\lambda_h}, \theta)) \tag{2.1}$$

where λ_h is the hyperparameter vector, $\wedge$ is the hyperparameter space, A_{λ_h} is the learning algorithm, θ represents the model parameters, and x_{train} and x_{test} are the training and test datasets.

Manually running and evaluating different hyperparameter configurations to find the best settings is labor-intensive and time-consuming, especially for high-dimensional search spaces. Tuning also requires expert knowledge and is application-dependent. Therefore, automatic hyperparameter optimization is needed to reduce computational cost. In addition, selecting an optimal optimization technique that can efficiently identify optimal hyperparameters is essential.

Regarding hyperparameter tuning, grid, and random search are the most popular but naive methods. Grid search systematically evaluates every hyperparameter combination, which is ineffective when the number of hyperparameters increases due to exponential resource utilization. Random search is more efficient by randomly sampling the space but does not learn from previous tests. Both techniques require substantial time and domain knowledge. Also, they are not adapted for continuous hyperparameters. More advanced techniques, such as Bayesian optimization, have been proposed, but they still struggle with many hyperparameters [90].

Due to the nature of this optimization problem, metaheuristics are strong candidates. Metaheuristic algorithms have a good reputation for efficiently solving complex optimization problems through intelligent, non-exhaustive search. They can find near-optimal solutions at lower computational cost. Therefore, they may allow more efficient hyperparameter optimization for complex and high-dimensional problems than conventional approaches.

Recent works have focused on using metaheuristics for hyperparameter optimization across various ML models, including SVMs [39] and DL models like RNNs [90, 4] and CNNs [34, 108, 11, 87]. PSO is one of the most widely applied metaheuristics across many model types [34, 108]. In addition, FA [4], BA [11], GWO [87], and other metaheuristics have also been successfully used to optimize and enhance the performance of ML models.

Singh et al. [108] proposed a Multi-level Particle Swarm Optimization (MPSO) algorithm for simultaneously optimizing CNN architecture and hyperparameters. Their approach utilizes multiple particle swarms at two levels: Level 1 swarms optimize the architecture, including the number of convolutional, pooling, and fully connected layers. Level 2 swarms then optimize the hyperparameters, such as filters and kernel sizes, for each layer. By cascading the architecture search and hyperparameter tuning, MPSO can learn globally optimal CNN designs. Experiments on MNIST, CIFAR-10, CIFAR-100, Convex Sets, and MDRBI datasets demonstrated that this multi-level metaheuristic approach effectively co-optimizes CNN architectures and hyperparameters in an end-to-end manner. The automated search discovered high-performing architectures surpassing manual design.

Though metaheuristics have shown promise for hyperparameter optimization, significant challenges remain. The massive range of possible hyperparameter values creates a highly vast and complex search space, making finding optimal configurations efficiently difficult. The high dimensionality and noise in the search landscape hinder efficient exploration and exploitation. Conditional inter-dependencies between hyperparameters further complicate the optimization problem. Moreover, evaluating the quality of a hyperparameter configuration requires repetitive training and evaluation of the ML model, which can be computationally expensive. Designing an objective function that accurately evaluates and ranks configurations is critical yet non-trivial. Preventing overfitting and ensuring generalization beyond training data requires robust evaluation. While metaheuristics have demonstrated success for hyperparameter optimization, overcoming these key issues remains an active research direction.

2.3 ML into metaheuristics

Metaheuristics are computational intelligence techniques widely used to solve complex optimization problems. They can find good solutions with reasonable computational effort, making them preferable to exact algorithms. Hence, their popularity has significantly grown over the past two decades.

Metaheuristics generate substantial data during the iterative search, such as high/low fitness solutions, operator sequences, evolution trajectories, local optima, etc. This data contains insights and knowledge about solution properties, operator performance, search precedence, etc. However, classical metaheuristics do not utilize this knowledge. On the other hand, ML can extract such knowledge to guide metaheuristics toward better decisions, improving solution quality, convergence speed, and robustness.

Thus, integrating ML in metaheuristics aims to create more intelligent optimization techniques by extracting insights from data to direct the metaheuristic search process. This new research direction has attracted growing attention recently.

ML can be integrated into metaheuristics at various levels, including algorithm selection, evaluating suitability, initialization, evolution, parameterization, and cooperation between algorithms. For further details, the reader may refer to [53] where a comprehensive survey is provided.

2.3.1 Algorithm selection

Algorithm selection is an important decision when using metaheuristics to solve optimization problems. Exhaustively running all available algorithms to choose the best solution is impractical due to limited computational resources. Therefore, the Algorithm Selection Problem (ASP) aims to automatically select the most appropriate algorithm(s) for solving a problem instance using ML techniques. ASP has four principal components: the problem space, the feature space, the algorithm space, and the performance space. ASP approaches can be offline, constructing the meta-model using training instances to map new instances, or online, building the meta-model dynamically while solving a set of instances. Current works use offline ASP with training data [53]. However, online ASP adapting selections during the search may improve robustness despite higher overhead.

In [33], researchers proposed using an attention-based neural network as a meta-learner for algorithm selection. The meta-learner architecture was similar to the encoder from the Transformer model, using multi-head attention to focus on pertinent features of problem instances selectively. The approach was evaluated on the vehicle routing problem with time windows (VRPTW), using three SA configurations as the algorithm portfolio. The meta-learner was trained on meta-data generated by solving VRPTW instances with the three algorithms. Results showed that the attention-based meta-learner outperformed several baseline methods, including MLP, SVM, and k-NN, in accurately selecting the best algorithm for a given VRPTW instance.

Despite ASP's effectiveness, implementing it presents several challenges [53]:

- **Data generation**: A diverse and representative set of training instances is required to create a reliable meta-model. This data should cover many problem characteristics and consider multiple performance criteria.
- **Affordability**: The cost of ASP must not exceed solving the instance with all algorithms. Otherwise, it is not justified.

- **Data availability**: A sufficient number of training instances are needed to create a reliable meta-model. However, having a pool of instances does not guarantee ASP effectiveness.
- **Instance dissimilarity and algorithmic discrimination**: Instances should elicit different behaviors with different algorithms to learn their strengths/weaknesses.
- **Online vs. offline ASP**: The choice depends on the application. Offline has lower overhead, while online enables more justifiable decisions.
- **Multiple selected algorithms**: When multiple algorithms are chosen, scheduling them requires static or dynamic strategies based on historical performance.
- **Extending ASP to other optimization**: ASP can potentially be applied to other optimization problems.

2.3.2 Fitness evaluation

Fitness evaluation guides metaheuristic search towards promising areas. For some problems, analytical fitness functions are unavailable or expensive to compute. ML can reduce computational effort via fitness approximation or reduction [53].

Fitness approximation aims to replace the original fitness function with a surrogate model that is computationally less expensive to evaluate. Examples of fitness approximation techniques include random forest [131], ANN [45], and SVM [45]. For multi-objective problems, fitness reduction aims to reduce the number of fitness function evaluations by selecting a subset of solutions to be evaluated. Examples of fitness reduction techniques include filtering techniques, such as clustering [113] and feature selection [78]. The choice between fitness approximation and fitness reduction depends on the specific problem and the trade-off between accuracy and computational cost. Fitness approximation is more suitable when the fitness function is computationally expensive, while fitness reduction is more suitable when the number of solutions to be evaluated is large.

hua Hao et al. [45] proposed a new surrogate modelling approach to solve bottleneck stage scheduling problems. The problem was formulated as a parallel machine scheduling problem to minimize total weighted completion time. Then, it was decomposed into an assignment sub-problem and a sequencing sub-problem. A surrogate model based on incremental ELM to quickly estimate the objective value of a given partial solution to the assignment sub-problem. A novel surrogate-model-based differential evolution algorithm (SM-DE) that iterates between the surrogate model and branch-and-bound was developed. The surrogate model provides fast but crude evaluations to guide the search, while the branch-and-bound algorithm provides accurate evaluations to train

the surrogate model. Experiments showed SM-DE significantly outperformed standard DE in solution quality within a limited computation time.

An adaptive multi-objective evolutionary algorithm for MOPs was introduced in [113]. The algorithm adapts to the Pareto optimal set's structure during evolution using dynamic clustering. A sampling technique perturbs current non-dominated solutions through a Gaussian distribution guided by cluster variance-covariance matrices to produce high-quality offspring. The sampling strategy was hybridized with Differential Evolution (DE) to balance exploration and exploitation. Experiments showed that the algorithm outperformed existing methods on test problems with complex Pareto optimal structures and fronts.

While ML-based fitness evaluation enables fast fitness evaluation, it faces challenges in ensuring global accuracy and efficiency. Additionally, selecting the most appropriate ML technique depends on the problem and preferences, which is more complex. Moreover, studies applying approximation to COPs are limited since most of them already have fast analytical fitness functions.

2.3.3 Initialization

The initialization strategy profoundly impacts metaheuristic exploration and exploitation abilities. If the initial solutions are not well diversified, a premature convergence may occur, and the metaheuristic gets stuck in local optima. On the other hand, low-quality initial solutions require more iterations to converge. Metaheuristics typically initialize solutions randomly, greedily, or via a hybrid approach. However, ML can be used to produce good-quality initial solutions and maintain the diversity of solutions.

ML can contribute to the initialization through three primary strategies [53]: complete generation, partial generation, and decomposition. ML techniques replace the solution generation strategies to construct total initial solutions in a complete generation. In partial generation, ML techniques are used to generate partial initial solutions. Decomposition divides the problem space into smaller sub-problems to facilitate the generation of initial solutions. Here, an initial solution is generated for each sub-problem using standard initialization strategies and then combined into an overall solution. The choice of initialization strategy depends on the specific problem and the trade-off between exploration and exploitation.

Learning can be offline by extracting insights from previous instances or online by gathering knowledge dynamically. Offline learning provides rich expertise but may not suit new instances, while online learning adapts to the current instance but can

be time-consuming. ML techniques used in initialization include ANNs [16], RL [6], clustering algorithms [5], and association rule mining [91].

A novel approach that combines regression techniques with GA to tackle the MKP was proposed by Rezoug et al. [102]. Specifically, five distinct regression models—Bayesian Automatic Relevance Determination (BARD), Gaussian Process Regression (GPR), k-Nearest Neighbors Regression (k-NNR), Random Forest Regression (RFR), and Epsilon Support Vector Regression (SVR)—were evaluated as part of the overall solution methodology. Training data was generated by solving small instances with the CPLEX optimizer. Since regression outputs yield continuous values per item, so they can guide the selection order for a feasible solution. The author proposed leveraging predictions made by one GPR to initialize the first population pool. Experiments over known complex MKP datasets showcased how this integrative approach yields competitive results.

The authors of [5] proposed a k-means clustering repair method to enhance initial solutions for a discrete DE algorithm to solve complex TSPs. The main idea was to divide the problem into sub-problems, solve each to find the optimal sub-tour, and recombine sub-tours into a near-optimal complete tour. Results revealed that the proposed approach surpassed most competing methods in proximity to optimal solutions and offered competitive computation times.

ML-based initialization enables faster convergence and better exploration-exploitation trade-offs. However, the complexity of additional parameters is a crucial challenge. Generating labeled training data and selecting informative features are also issues for supervised learning models. Other challenges include defining appropriate states and actions in RL.

2.3.4 Evolution

Integrating ML into metaheuristics during the evolution of the search process can enhance the search process in several key ways. It makes metaheuristics more adaptive, efficient, reusable, automated, and robust. Thus, this integration is an active area of research with great potential where ML can be incorporated in three main ways: operator selection, learnable evolution model, and neighbor generation [53].

2.3.4.1 Operator selection

Selecting the optimal operator in metaheuristics involves choosing the most appropriate operator based on the operators' performance during the search process. However,

the search space is a non-stationary environment, and different operators may be particularly effective at certain stages of the search process but perform poorly at others. Therefore, appropriately employing different operators during the search process can produce better solutions.

ML techniques can be used to select the most suitable operator based on operators' performance during the search process. This approach aims to improve the exploration and exploitation abilities of metaheuristics. Typically, this involves assessing each operator's credit based on the performance achieved by its last W applications, where W is the sliding window size for each operator. RL is commonly employed for this purpose in the literature [53].

In [129], the authors proposed a cooperative water wave optimization algorithm (CWWO) to solve the distributed assembly no-idle flow shop scheduling problem to minimize maximum assembly completion time. CWWO incorporated a reinforcement learning mechanism in the propagation phase to balance exploration and exploitation. The reinforcement learning mechanism is based on the Variable Neighborhood Search (VNS) algorithm. Experiments on benchmarks showed that the proposed CWWO had stability and outperformed other methods.

Regardless, applying RL introduces some algorithmic and implementation challenges related to overhead, tuning, convergence, and credit assignment that must be addressed carefully to realize the full benefits. Hybrid techniques that combine RL with other methods may help overcome some of these challenges.

2.3.4.2 Learnable evolution model

Learnable Evolution Model (LEM) represents a new class of EAs that incorporates ML techniques to generate new populations. Unlike Darwinian-type EAs that use semi-random operators, LEM introduces a learning mode that uses ML techniques to produce new solutions based on the knowledge obtained from past populations. This approach aims to make the search process more intelligent and autonomous, ultimately improving the performance of EAs.

In this mode, a learning algorithm analyzes high-performing and low-performing solutions to extract rules characterizing differences between them. These learned rules are then used to generate new solution populations through a deliberate reasoning process rather than random variation, like in Darwinian EAs. Hence, the learnable evolution model consists of hypothesis generation and hypothesis instantiation. In the hypothesis generation process, ML techniques are used to determine hypotheses that characterize the differences between high-fitness and low-fitness solutions in recent

or previous populations. Subsequently, in the hypothesis instantiation process, new solutions are generated based on the learned hypotheses obtained in the hypothesis generation process.

Adaptive Query (AQ) rule learning algorithms are used for hypothesis generation and rule injection for instantiation. Popular AQ rule learning algorithms include AQ18 & AQ21 rule learning [88, 122] and C4.5 decision tree [51].

Jung et al. [51] proposed a Hybrid Harmony Search (HyHS) algorithm that combines Harmony Search (HS) with the C4.5 decision tree algorithm. HyHS runs HS for a set number of iterations to generate high-quality solutions stored in the Harmony Memory (HM). C4.5 then analyzes the high-quality solutions in HM to induce rules and patterns. These rules guide the generation of new solutions. This hybrid method was evaluated on test functions and water distribution system design problems. The results demonstrated that hybridizing HS with C4.5 improves its efficiency and effectiveness for optimization problems.

While applying ML to generate new populations in EAs holds promise, it poses difficulties arising from the need for fitness evaluations, handling dynamic environments, designing the learning component, reduce the computational overhead. Furthermore, determining the optimal strategy for combining and alternating the learned-based and random variation-based solution generation (Darwinian modes) poses a significant challenge.

2.3.4.3 Neighbor generation

The neighbor generation process plays a pivotal role in metaheuristics, as it determines the quality and diversity of the generated solutions. ML techniques offer a means to guide the neighbor generation process by learning from the properties of good solutions obtained during the search process.

ML techniques can be used to extract knowledge from solutions and utilize them to generate new solutions. This involves the knowledge extraction phases to identify common patterns or characteristics in high-quality solutions and the knowledge injection phases to incorporate these patterns when generating new neighbor solutions. In this way, we can lead the search towards promising areas of the search space and accelerate finding a good solution. Knowledge extraction can be done offline from training instances or online during the search process. While online extraction avoids potential biases from offline patterns, it typically incurs higher overhead. Standard techniques utilized for neighbor generation include association rule mining [132], RL [120], and decision trees [9].

In [132], the authors proposed a method that combines data mining and EAs. The method consists of maintaining a population of high-quality solutions and using a frequent pattern mining algorithm (apriori) to extract common patterns from the population. These extracted frequent patterns are then used to generate new candidate solutions, which are further optimized. Experimental results on benchmark Quadratic Assignment Problem (QAP) instances showed that the new algorithm is highly competitive with state-of-the-art QAP algorithms.

Integrating ML into neighbor generation leverages prior knowledge to focus the search process. Still, it presents several challenges, including determining when to extract knowledge, how frequently to update extracted patterns, handling multiple conflicting patterns, and balancing exploitation and exploration when using extracted knowledge. Additionally, there may be value in learning from bad solutions to complement patterns observed in good solutions and exploring rare patterns to enhance diversity.

2.3.5 Parameter setting

Parameter setting is a crucial aspect of metaheuristics as the performance of the algorithm significantly depends on the values of its parameters [114]. The values of parameters should be properly set to obtain the highest performance. ML techniques can help to establish or control the parameter values before or during the search process.

ML can assist in the parameter setting process in two ways: parameter tuning (offline) and parameter control (online). Parameter tuning identifies appropriate parameters before running the algorithm using training instances. ML models like LR [101, 67] and SVM [67, 127] can be used to predict the performance of different parameter settings. In contrast, parameter control focuses on adjusting the parameter values dynamically during algorithm execution rather than using initially fine-tuned parameters that remain unchanged during the whole execution. RL-based techniques have been predominantly used for controlling parameters during the search process by assigning credit to parameter settings based on their impact on solutions, given their iterative learning capability from interactions with the environment to maximize rewards [24].

The authors in [67] proposed a new approach for tuning PSO parameters using data mining. The method trains regression models to predict practical parameter values based on particle signatures containing information about the problem instance, current solution, and search history. By learning from past runs which settings work

best in different situations, the models can automatically tune the parameters online during execution. Experiments showed that this approach outperforms other PSO tuning methods on benchmark functions.

Chen et al. [24] presented a self-learning GA based on RL for solving the flexible job shop scheduling problem. They proposed an adaptive scheme where RL dynamically tunes GA's crossover and mutation probabilities based on search experiences. The algorithm combines both SARSA and Q-learning RL approaches at distinct phases. The GA environment for RL includes state representation via population fitness, action space as different probability ranges, and reward calculation from improvement in best/average fitness. By balancing exploration and exploitation through intelligent parameter adjustments, this approach achieved better convergence rates and higher-quality solutions than traditional fixed parameter settings.

As the optimal parameter settings may vary not only across different problem instances but also at different stages of the search process for the same problem instance, parameter control, despite its computational cost, is recommended. One of the critical challenges in parameter control is balancing the exploitation of good settings versus the exploration of new ones. Handling continuous parameters is also tricky, requiring separate optimization or self-adaptive mechanisms.

2.3.6 Cooperation

Cooperative metaheuristics leverage the strengths of multiple optimization algorithms by enabling them to work together. This Cooperation can occur between separate metaheuristics or between internal components of a metaheuristic. The cooperation is modeled as a multi-agent system where agents exchange information to guide and focus the collective search. ML helps design intelligent cooperation frameworks by extracting knowledge about the search space from individual agents' experiences. ML techniques used include RL to adapt agent behaviors, association rule mining to identify common good solution characteristics for knowledge sharing, and clustering to group similar solutions together [75, 81, 54].

Information exchange between agents can be direct (many-to-many) or indirect (where agents only use information provided in a common pool). Key requirements are understanding agents' strengths/weaknesses to combine them effectively and generating useful shared knowledge. Information shared includes solutions, partial solutions, characteristics of good/bad regions, etc.

Cooperation can be parallel, with agents running simultaneously to reduce runtime, or sequential. The collaboration between agents can be synchronous or asynchronous.

Asynchronous cooperation, where agents don't wait for each other, is more common as it is operationally easier to implement, but adapting the framework to handle dynamic search landscapes poses challenges.

In [54], a hybrid metaheuristic technique was proposed using the k-means algorithm. The k-means algorithm selects representative solutions from NSGA-II to feed into the Iterated Local Search (ILS) algorithm. This hybrid approach minimizes transportation and hub costs while reducing maximum origin-destination time in a congested hub network. The authors noted that the combination of global and local search algorithms through ML techniques led to high-quality solutions within a short computation time.

Cooperative metaheuristics enhanced by machine learning form an intelligent, adaptable multi-agent optimization process by exploiting complementary algorithm strengths and shared knowledge. However, it can face several challenges. One of the main challenges is the design of a practical cooperation framework that can adapt to the search space. Other challenges include the overhead, tuning, maintaining diversity among agents, and assigning credit for agent performance contributions.

2.4 Chapter summary

Throughout this chapter, we explored the symbiotic relationship between ML and metaheuristics. ML provides an efficient way of extracting patterns from data, while metaheuristics provide robust optimization strategies. Combining both approaches makes it possible to develop hybrid techniques that exploit the strengths of each paradigm.

Metaheuristics can improve ML performance through hyperparameter optimization and feature selection. On the other hand, integrating ML models into metaheuristics enables a more intelligent optimization process that is guided and informed by the knowledge learned.

Reviewing related literature, we highlighted existing works on synergy between ML and metaheuristics. Despite the promising potential of this synergy, several challenges and promising research directions lie ahead. Challenges include the need for further research in defining ML models, addressing high-dimensional and non-linear optimization problems, and managing computational complexity. Promising directions include exploring novel hybrid methodologies, developing adaptive metaheuristic algorithms guided by ML, and applying these hybrid approaches to new complex problem domains.

In the upcoming chapters, we will present our contributions to combining these complementary paradigms for intelligent and efficient problem-solving. First, we will

present our metaheuristic-based approaches for hyperparameter optimization of CNNs and ELMs. Next, we will explain our techniques for solving COPs, including MKP, SSP, and WDP, using ML-enhanced metaheuristics.

Part II

Contributions

Chapter 3

Hybrid GWO for ML Hyperparameter Optimization

3.1 Introduction

Hyperparameter optimization is critical for developing high-performance ML models. However, identifying the optimal combination of hyperparameter values is challenging, often requiring extensive trial-and-error experimentation. Metaheuristic optimization algorithms can automate hyperparameter tuning to find optimal configurations efficiently.

GWO is a nature-inspired metaheuristic based on grey wolf hunting behavior. It is designed for continuous optimization, aligning well with continuous hyperparameter search spaces. While GWO has proven to be a valuable tool in solving numerous optimization problems, the algorithm's inherent limitations, such as the risk of stagnation in local optima, highlight the need for further improvements and modifications.

To address these limitations, we propose in this chapter a novel hybrid optimization algorithm combining the GWO and the MVO, called GWO-MVO, for efficient hyperparameter optimization. The proposed GWO-MVO algorithm is first applied to optimize a single hyperparameter - the dropout probability in CNNs. It is then evaluated on a high-dimensional hyperparameter optimization problem involving optimizing the weights and biases of ELM. Both approaches are assessed on image classification tasks.

This chapter covers the GWO-MVO algorithm methodology, the proposed hyperparameter optimization approaches, along with experimental setups, results, analysis, and limitations. The aim is to demonstrate the capabilities of the hybrid GWO-MVO algorithm in automating the tuning process for boosting ML model performance.

3.2 Proposed Hybrid Grey Wolf Optimizer-Multi-Verse Optimizer

3.2.1 Motivation

The GWO algorithm has received significant attention in recent years due to its inherent advantages:

1. GWO boasts a simple implementation process, making it accessible and practitioner-friendly.
2. The algorithm requires minimal parameter tuning, which reduces the computational load and simplifies the optimization process.
3. GWO has demonstrated robust convergence performance, consistently identifying near-optimal solutions in many optimization problems.

Despite these advantages, when applied to real-world optimization tasks, basic GWO encounters several difficulties that can detract from its effectiveness. One notable limitation is the relatively slow convergence speed in the final stages of the search. In addition, GWO is very sensitive to early convergence, where the algorithm can easily get trapped in local optima.

Various strategies and hybridization approaches have been proposed to address these limitations. These include incorporating local search techniques and hybridizing GWO with other metaheuristics or heuristics [106, 76, 69].

The MVO algorithm represents a newer metaheuristic founded on strong mathematical models designed to facilitate exploration, exploitation, and local search (Section 1.3.3.1). Therefore, a hybrid approach integrating GWO and MVO can be employed. In this scenario, GWO is utilized to explore the search space, while MVO serves as a refinement step. This hybrid strategy may prove beneficial in accelerating convergence, enhancing exploration capabilities, and improving GWO's overall optimization performance.

3.2.2 Proposed hybrid algorithm

Similar to the conventional GWO, our proposed hybrid GWO-MVO algorithm draws inspiration from the leadership hierarchy and hunting behaviors observed in grey wolves. Consequently, each wolf adjusts its position based on the positions of the three leader wolves.

In the standard GWO algorithm, the distance control parameter 'a' plays a pivotal role in influencing the convergence speed and striking a balance between the

exploration and exploitation capabilities of the algorithm. In the original implementation, the control parameter 'a' is linearly decreased from 2 to 0 throughout iterations (Section 1.3.3.1). While this linear decrease strategy exhibits rapid convergence and high accuracy in the early stages for many well-known benchmark problems, the actual search process of GWO is more intricate. This linear reduction of the adaptive parameter 'a' cannot truly capture it.

To overcome this limitation, we propose a nonlinear decay strategy to replace the linear decay of parameter 'a'. The primary objective of this nonlinear decay approach is to decelerate the convergence towards the leader wolves, thereby facilitating better exploration and exploitation of the search space. Our specific nonlinear decay strategy is derived from calculating the TDR parameter in the MVO algorithm (Section 1.3.3.2). Consequently, 'a' is determined by Equation 3.1.

$$a = 2 \times (1 - \frac{t^{\frac{1}{p_1}}}{T^{\frac{1}{p_1}}}) \tag{3.1}$$

In the equation, 't' represents the current iteration, 'T' denotes the maximum number of iterations, and 'p_1' indicates the accuracy rate of exploitation across iterations.

We introduce a new strategy to further bolster the exploitation and local search capabilities around the best solution (represented by the alpha wolf α). The position of the worst wolf is replaced by a new position generated near the alpha wolf α. This replacement strategy is inspired by the MVO algorithm and is described in the following equations:

$$X_{worst} = \begin{cases} X_{new} & \text{if } f^*(X_{new}) < f^*(X_{worst}) \\ X_{worst} & \text{else} \end{cases} \tag{3.2}$$

$$X_{new} = X_\alpha + TDR \times ((ub - lb) \times r_3 + lb) \tag{3.3}$$

$$TDR = 1 - \frac{t^{\frac{1}{p_2}}}{T^{\frac{1}{p_2}}} \tag{3.4}$$

- f^* is the fitness function.
- p_2 defines the accuracy of the exploitation over iterations.
- X_{worst} is the position of the worst wolf.
- lb and ub are, respectively, the lower and upper bounds.
- $r_3 \in [-1, 1]$is a random number selected from a uniform distribution.

It is worth noting that the higher the values of p_1 and p_2, the faster and more precise the local exploitation and search become. To strike an effective balance between exploration and exploitation, p_1 is set to a very small value compared to p_2. Essentially,

the first modification aims to enhance the exploration capabilities, while the second modification reinforces the exploitation around the best solution. The GWO-MVO algorithm is sketched in Algorithm 5.

Algorithm 5 GWO-MVO

Input: T : max number of iterations, n: population size, p_1, p_2, f : fitness function

Output: X_α

Initialize the grey wolf population $X_i\,(i = 1, 2, .., n)$
Evaluate the population
Select the leaders X_α, X_β and X_δ
$t \leftarrow 0$
while $t < T$ **do**
    Update a using Equation 3.1
    for each individual **do**
        Update the position of the current individual by Equation 1.16
    end for
    Evaluate the new population
    Update the leaders X_α, X_β and X_δ
    Update TDR using Equation 3.4
    replace the worst wolf by Equation 3.2
    Update the leaders X_α, X_β and X_δ
    $t \leftarrow t + 1$
end while
return X_α

3.2.3 Time complexity of GWO-MVO

The Time complexity, a crucial aspect in understanding the performance of the hybrid GWO-MVO algorithm, can be calculated as follows:

- Initializing the population takes $O(N \times m)$ time, where N represents the population size, and m represents the problem dimension.
- Initializing the parameters is a constant-time operation and requires $O(1)$ time.
- Evaluating the initial solutions requires $O(N \times f(S))$ time, where $f(x)$ is the time complexity of the fitness function.
- Selecting the positions of the initial leaders using a sorting algorithm like heapsort requires $O(NlogN)$ time.
- Updating the positions of the wolves requires $O(N \times m)$ time.
- Calculating the fitness of each solution requires $O(f(S))$ time.

- Updating the positions of the leaders requires $O(N)$ time.

Based on this analysis, we can conclude that for each generation or iteration, the time complexity is $O(N \times (m + f(S))$. If we assume a maximum number of iterations T, then the total time complexity of the GWO-MVO algorithm is $O(T \times N \times (m + f(S)))$.

3.3 GWO-MVO for dropout regularization

To assess the performance and effectiveness of the proposed hybrid GWO-MVO algorithm, it was initially applied to the task of estimating the optimal dropout probability for CNNs. As discussed earlier (Section 1.4.2.3), the dropout technique is a powerful regularization method that helps mitigate the overfitting problem in DNNs. However, determining the optimal value of the dropout rate through an exhaustive search approach can be computationally expensive and time-consuming. In the rest of this section, we will provide thorough insights into this contribution.

3.3.1 Proposed approach

The main goal is to enhance the accuracy of CNN models by optimizing the dropout probability hyperparameter. Hence, the performance of the CNN model guides the search process for finding the optimal dropout value. Figure 3.1 depicts the flowchart of the proposed approach. In the following, we detail the key components of our approach.

3.3.1.1 Solution representation and initialization

Candidate solutions are represented as the positions of individuals within the GWO-MVO algorithm. Each individual's position corresponds to a potential value for the dropout probability (dp) to be optimized. During initialization, the dp values for each individual are randomly generated, with dp_i assigned a random number between 0 and 1 for individual i. This represents the starting population for the dropout probability values that the optimization algorithm will later evolve.

3.3.1.2 Fitness function

The quality of each potential solution, represented by an individual's position (i.e., a dropout probability value), is evaluated based on the performance of the CNN model when trained using that dropout rate. Specifically, for each dp value encoded, a CNN

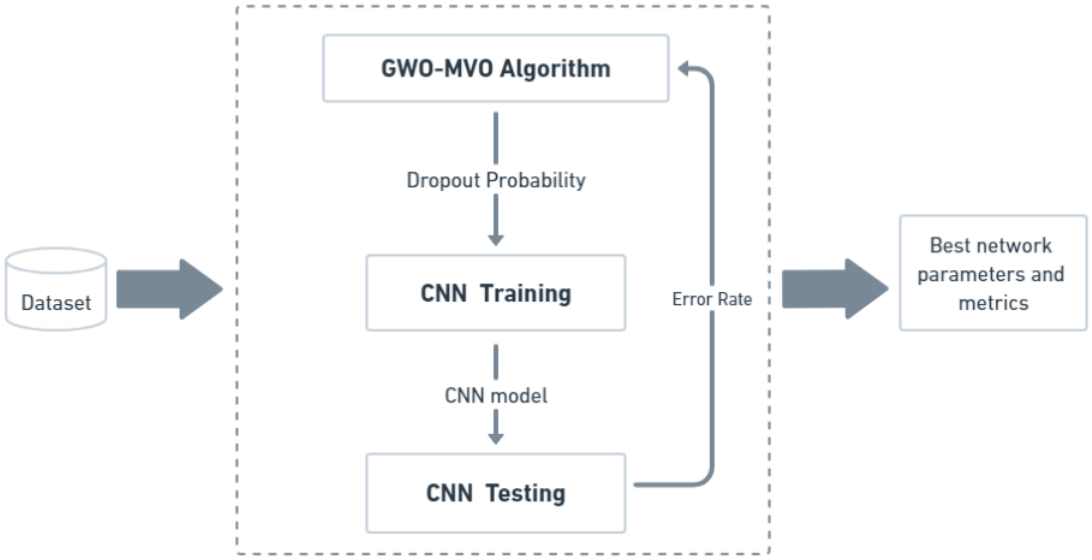


Figure 3.1: Flowchart of the proposed GWO-MVO approach for dropout estimation.

model is defined and trained using that dropout probability. The training process utilizes an early stopping condition adjusted to 5% of the total training epochs. Once training is complete, the model is evaluated on the validation set, and its accuracy is measured. The goal is to minimize the classification error rate, which is the fitness function to guide the search process.

3.3.2 Experiments

In this section, we present the experimental studies conducted to evaluate the performance of the proposed approaches GWO-MVO for optimizing dropout probabilities in CNNs. To enable parallel exploitation and exploration, the position updates of individuals were implemented using Numba, which provides an NVIDIA CUDA back-end for GPU computing in Python [65]. The CNN models were implemented using the Caffe deep learning library, which provides efficient GPU-accelerated implementations of CNN architectures [50]. The experiments were run on a system with a single Nvidia RTX 2060 GPU, AMD Ryzen 7 3700X CPU, 16GB RAM, and Ubuntu 18.04 OS.

3.3.2.1 Datasets

The proposed approach is evaluated on the image classification task using three widely used benchmark datasets:

(a) *MNIST* (b) *USPS* (c) *CIFAR* − 10

Figure 3.2: Examples from the datasets used for dropout regularization.

Table 3.1: Dataset configuration used in dropout regularization.

Dataset	#Training set	#Validation set	#Testing set
MNIST	20,000 (64)	40,000 (100)	10,000 (100)
USPS	2,406 (32)	4,885 (977)	2,007 (2,007)
CIFAR-10	20,000 (100)	30,000 (100)	10,000 (100)

1. **MNIST** [28]: This dataset consists of handwritten digit images ranging from 0 to 9. It is divided into separate training and testing sets. All images are grayscale with a resolution of 28x28 pixels.
2. **USPS** [47]: The USPS dataset is also designed for handwritten digit recognition tasks. Like MNIST, it is split into training and testing sets, with all images being grayscale and having a smaller resolution of 16x16 pixels.
3. **CIFAR-10** [61]: This dataset comprises color images spanning ten different classes. It is divided into separate training and testing sets, with each image having a resolution of 32x32 pixels and represented in color.

These datasets are extensively used for benchmarking ML and computer vision algorithms. Figure 3.2 provides sample images from each dataset's training sets, illustrating the data's diversity. To ensure proper evaluation, each dataset was further split into training and validation subsets, following the methodology employed by De Rosa et al. [27]. Table 3.1 presents the details about the number of images in each subset and the batch sizes used during training.

3.3.2.2 CNN architectures

To ensure a fair and consistent comparison with previous works, we adopted the same CNN architectures proposed in [27]. Two CNN architectures were employed, as depicted in [27]. One architecture was used for the MNIST and USPS datasets, while the other was utilized for the CIFAR-10 dataset.

Table 3.2: CNN parameter configuration for dropout regularization.

Dataset	Learning rate η	Momentum α	Weight decay λ	Dropout ratio p	#Iterations
MNIST	0.01	0.9	0.0005	[0, 1]	10,000
USPS	0.01	0.9	0.0005	[0, 1]	10,000
CIFAR-10	0.001	0.9	0.004	[0, 1]	4000

Table 3.3: Metaheuristic parameter configuration for dropout regularization.

Metaheuristic	Parameters	#Training procedure calls
GWO	#iterations=10, #agents=7	77
GWO-MVO	#iterations=10, #agents=6, $p_1 = 2$, $p_2 = 10$	76

It is important to note that for the USPS dataset, which consists of lower-resolution images (16x16 pixels), the kernel size for the convolutional layers was adjusted to 3x3 instead of the original 5x5 used in the other architectures. This modification was necessary to accommodate the smaller input dimensions of the USPS images.

3.3.2.3 Parameters tuning

We adopted the same parameter configurations for the CNN models as employed in [27] to ensure a fair and comparable evaluation with previous works. Table 3.2 presents the specific parameter settings used for each dataset in our experiments.

Additionally, we conducted a comparative analysis between the GWO-MVO algorithm and the basic GWO algorithm. The control parameters of the metaheuristics were carefully tuned through empirical experimentation. This tuning process aimed to ensure that the total number of learning procedure calls (fitness evaluations) did not exceed the limit of 77, as specified in the previous studies by [27, 11, 10, 12]. Table 3.3 summarizes the optimized values of the metaheuristic parameters utilized in our experiments.

3.3.2.4 Experimental results

To ensure a fair and rigorous comparative study, we adhered to the same testing procedure employed in [27]. Specifically, we executed our experiments 20 separate times, and the average accuracy on the test set was utilized as the primary metric for comparison. Furthermore, to validate the performance of our proposed GWO-MVO algorithm relative to the baseline GWO algorithm, we conducted a comprehensive statistical analysis of the obtained results.

Table 3.4: GWO versus GWO-MVO for dropout regularization

Dataset	Metric	GWO	GWO-MVO
MNIST	Mean	99.22	**99.3**
	Min	99.18	**99.28**
	Max	99.32	**99.33**
	SD	0.03	**0.01**
USPS	Mean	96.37	**96.76**
	Min	96.01	**96.71**
	Max	96.71	**96.91**
	SD	0.19	**0.05**
CIFAR-10	Mean	72.1	**73.19**
	Min	71,38	**72,61**
	Max	72.96	**74.4**
	SD	0.53	**0.48**

Numerical results

Table 3.4 presents the mean (Mean), best (Max), worst (Min) classification accuracy values and standard deviations (SD) obtained by the GWO and the proposed GWO-MVO algorithms. The table shows that the GWO-MVO approach achieves superior results across all datasets. GWO-MVO outperforms the baseline GWO, achieving the highest best accuracy on all three datasets: 99.33% (MNIST), 96.91% (USPS), and 74.4% (CIFAR-10). It also consistently outperforms GWO in terms of average and minimum accuracy values across all datasets. Furthermore, the standard deviations of GWO-MVO are the lowest, indicating highly stable and consistent performance with minimal dispersion around the mean accuracy.

Statistical results

To statistically validate the impact of the proposed improvements to the global optima search in GWO, a non-parametric Wilcoxon Rank-Sum Test was conducted at a 5% significance level ($\alpha = 0.05$) over 20 independent runs on all datasets. The p-values reported in Table 3.5 reveal that the proposed GWO-MVO algorithm significantly outperforms the baseline GWO algorithm across all datasets, with $p-$values less than 0.05. These statistical results support the effectiveness of the hybridization strategy employed in GWO-MVO, leading to significant performance gains over the original GWO in optimizing dropout probabilities for CNNs.

Table 3.5: p-value of GWO-MVO against GWO for dropout regularization.

Dataset	MNIST	USPS	CIFAR-10
p-value	< 0.0001	< 0.0001	< 0.0001

Table 3.6: Comparing average accuracies for dropout regularization.

Method	MNIST		USPS		CIFAR-10	
	Accuracy	p	Accuracy	p	Accuracy	p
Caffe [27]	99.07 (%)	0	95.80 (%)	0	71.47 (%)	0
Dropout Caffe [27]	99.18 (%)	0.5	96.21 (%)	0.5	72.08 (%)	0.5
BA-OM [11]	99.19 (%)	0.5216	-	-	71.76 (%)	0.671
CFAEE [10]	99.28 (%)	0.529	96.88 (%)	0.845	72.32 (%)	0.388
OBSCA-FS [12]	99.26 (%)	0.524	**96.93** (%)	0.838	72.54 (%)	0.394
GWO (our approach)	99.22 (%)	0.384	96.37 (%)	0.503	72.10 (%)	0.814
GWO-MVO (our approach)	**99.3** (%)	0.343	96.76 (%)	0.61	**73.19** (%)	0.809

3.3.2.5 Comparison with previous works

Table 3.6 presents a comparative analysis of the proposed GWO-MVO algorithm's performance against previous works reported in the literature, with a focus on the average classification accuracies obtained across the three benchmark datasets: MNIST, USPS, and CIFAR-10.

On the MNIST and CIFAR-10 datasets, the GWO-MVO algorithm demonstrates superior performance, achieving average accuracy rates of 92.3% and 73.19%, respectively. These results surpass the accuracy rates reported by previous studies, highlighting the effectiveness of the proposed approach in optimizing dropout probabilities for these datasets. However, on the USPS dataset, the GWO-MVO algorithm achieves an average accuracy rate of 96.76%, which is lower than the 96.93% rate obtained by the OBSCA-FS method proposed in the literature.

3.3.3 Discussion and analysis

The experimental results presented in this contribution demonstrate the effectiveness of the proposed GWO-MVO algorithm in optimizing dropout probabilities for CNNs. The algorithm consistently outperforms the baseline GWO and achieves competitive performance compared to other state-of-the-art methods on benchmark datasets.

While the current contribution focuses on optimizing a single hyperparameter (dropout probability), the promising results pave the way for exploring the GWO-MVO algorithm's capabilities in optimizing multiple hyperparameters simultaneously across diverse ML techniques. Addressing the challenges of high-dimensional, mixed-type

hyperparameter spaces and validating the algorithm's performance against state-of-the-art techniques would be crucial steps toward establishing its practical relevance and potential for widespread adoption in the ML community.

With this motivation, the next part of this chapter will apply the GWO-MVO algorithm to a more complex ML hyperparameter optimization problem, extending its applicability beyond the current scope of optimizing a single hyperparameter for CNNs. This progression aims to demonstrate further the algorithm's versatility and robustness in tackling high-dimensional optimization challenges.

3.4 GWO-MVO for optimizing the ELM model

3.4.1 Motivation

The ELM offers advantages over conventional methods by providing fast learning without iterative gradient-based training. ELM requires minimal parameter tuning, making it suitable for real-time network retraining applications. However, ELM's performance can be hindered by the random initialization of weights and biases and poor quality of input features, especially for complex visual data (Section 1.4.1.1).

Numerous contributions have been made to improve the performance of ELMs, focusing on optimizing weight and bias initialization to increase prediction accuracy. Metaheuristic techniques have proven effective in optimizing the input parameters of ELMs. However, most contributions typically focus on addressing straightforward problems with simple datasets and often overlook the issue of overfitting.

Another line of research has explored enhancing the quality of input features for ELMs, such as feeding them with high-level feature representations extracted by DNNs like CNNs. While current CNN-based methods have shown improved robustness over previous approaches, challenges like overfitting and noise persist. CAEs have demonstrated promising performance in handling highly variable conditions but with lower precision than traditional deep networks like VGG16.

Moreover, most image classification systems assume consistent image quality and do not incorporate preprocessing steps, even though image quality can vary in real-world applications. Studies have shown that models trained on noisy data may perform worse than those trained on clean data. However, injecting noise into the training data can be beneficial for applications where image quality is subject to variation after deployment. Nazare et al. Demonstrated that models trained and evaluated on noisy datasets performed less effectively than those trained and evaluated on clean data [92].

Moreover, denoising techniques do not guarantee significant improvement and could potentially yield adverse effects.

In light of these challenges, this research aims to improve the performance of ELM networks for image classification tasks by addressing the issues posed by random weight initialization, poor input feature quality, and variable image quality. The proposed approach combines features extracted by a basic CNN with those extracted by a CAE, leveraging the strengths of both architectures while mitigating their weaknesses.

3.4.2 Proposed approach

In the context of image classification, generating relevant features that enable the classifier effectively distinguish between different classes is crucial. In this work, we propose a method that leverages the strengths of two models: CNN and CAE. Our approach combines the extracted features from both models to feed the ELM classifier.

The first step involves training the CNN and CAE models on the training set for a few epochs. The CNN model performs feature extraction through its designated feature extraction component, while the CAE model employs its encoder to generate feature vectors. These two sets of feature vectors, derived from the CNN and CAE models, are then concatenated to form the input vector for the ELM classifier.

Once the ELM input vector is defined, the GWO-MVO + ELM method determines the optimal initial values for the model weights and biases. This optimization technique improves the performance of the ELM classifier by finding the most suitable parameter values through an efficient search process. The overall proposed approach is outlined in Figure 3.3.

3.4.2.1 Solution representation and initialization

In the proposed approach, each wolf's position represents a vector containing possible values for the weights and biases of the ELM model. The position of an agent has m dimensions, where m is calculated as:

$$m = n \times k + n \tag{3.5}$$

here, n is the size of the hidden layer in the ELM model, and k is the size of the input feature vector.

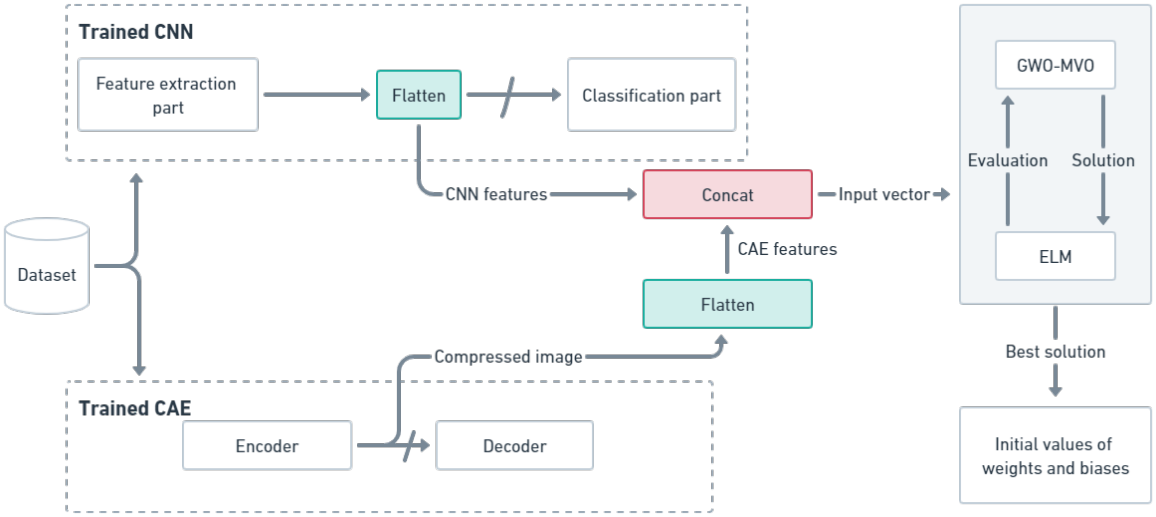


Figure 3.3: Flowchart of the proposed improved ELM model.

The position of an individual i can be represented as follows:

$$X^i = < \underbrace{x_1^i, x_2^i, ..., x_{nk}^i}_{\text{Weights}}, \underbrace{x_{(nk+1)}^i, ..., x_m^i}_{\text{Bias}} > \tag{3.6}$$

In this formulation, the first nk elements of the position vector X^i represent the weights, while the remaining n elements correspond to the biases of the ELM model.

All individuals' initial weights and biases are randomly generated within the range of $[-1, 1]$.

3.4.2.2 Fitness functions

Choosing the fitness function holds significant importance in any metaheuristic approach since it delineates the exploration of the search space. Therefore, we formulated four fitness functions, each utilized separately for evaluating solutions. For every potential solution encompassing the weights and biases, an ELM model is constructed and trained for evaluation.

Bartlett's theory proposes that a smaller norm of the weights is associated with improved generalization performance of the network [13]. Thus, our goal is to minimize the weight matrix norm, which defines the first fitness function.

$$F_1 = \| \check{\beta} \| \tag{3.7}$$

Many prior studies have concluded that assessing the training set is unnecessary, given that the ELM training procedure inherently reduces training error. Nonetheless, the selection of initial weights and biases can significantly influence training error. Hence, our second evaluation function focuses on the classification error rate observed in the training set.

$$F_2 = error\_rate_{trainng\_set} \tag{3.8}$$

To enhance generalization, our third function is established based on the classification error rate observed on the validation set. We employed 5-fold CV to save time. 5-fold CV has proven to be useful in many models [80]. This method involves randomly dividing the training set into five equal partitions. In each iteration, one partition serves as the validation set while the other four are used for training. This process is repeated five times, ensuring that each partition is utilized once as a validation set. The fitness function is determined by the average error rate across the five validation sets, as depicted in Equation 3.9.

$$F_3 = \frac{1}{5}\sum_{i=1}^{5} error\_rate_{validation\_set_i} \tag{3.9}$$

To prevent the issue of overfitting, the fourth fitness function is designed to concurrently optimize the error rates on both the training and validation sets. This approach constitutes a multi-objective optimization problem. Employing the weighted sum method, we merge the two objective functions into a unified function. Additionally, we utilize the 5-fold CV method. The ultimate fitness function is computed using Equation 3.10.

$$F_4 = w_1 \times (\frac{1}{5}\sum_{i=1}^{5} error\_rate_{training\_set_i}) + w_2 \times (\frac{1}{5}\sum_{i=1}^{5} error\_rate_{validation\_set_i}) \tag{3.10}$$

where $w1 = w2 = 0.5$.

3.4.3 Experiments

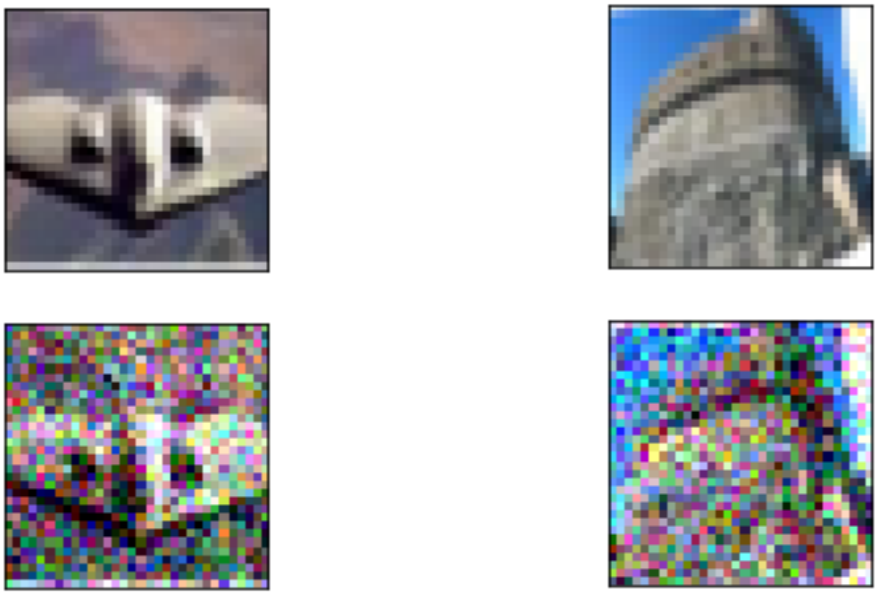
In this section, we elaborate on the experiments conducted and present the results obtained from our proposed approach aimed at enhancing the ELM model. It is noteworthy that all tests were executed on a desktop computer equipped with the following specifications: an AMD RYZEN 7 3700X CPU, 16 GB of RAM, a single GPU (RTX 2060 super), and an Ubuntu 18.04 operating system. Furthermore, all our

programs were implemented in Python, using the Keras API and fundamental data science libraries such as scikit-learn, NumPy, SciPy, and pandas.

3.4.3.1 Datasets

Our study was evaluated on the image classification task, employing two well-known benchmark datasets. In addition to the previously used CIFAR-10 dataset, we utilized the CIFAR-100 dataset, which is similar to CIFAR-10 but comprises 100 classes instead of ten [62].

We created a noisy version for each dataset to investigate the effect of noise on the training and testing sets. Gaussian noise with a mean of 0 and a standard deviation of 50 was applied to introduce noise. This standard deviation value was chosen based on its ability to create a challenging test environment for Gaussian noise, as highlighted in [92]. Figure 3.4 provides an example of both original and noisy images from the training set for each dataset, illustrating the impact of the applied Gaussian noise.



(a) CIFAR-10 **(b)** CIFAR-100

Figure 3.4: Examples of original and noisy images for each training set. The first line shows the original image and the second represents images with Gaussian noise ($\sigma = 50$).

3.4.3.2 CNN architecture

We employed the CIFAR-VGG architecture, a modified version of the popular VGG16 model, specifically adapted to mitigate the overfitting problem by incorporating dropout

and weight decay techniques. Compared to modern complex architectures, CIFAR-VGG offers a simpler and more straightforward network that can be easier to train. To ensure optimal performance, we utilized the same architecture implementation available at (https://github.com/geifmany/cifar-vgg).

3.4.3.3 CAE architecture

To enable dimensionality reduction and feature extraction, we used a simple and conventional CAE architecture. Figure 3.5 illustrates the CAE structures employed in our experiments.

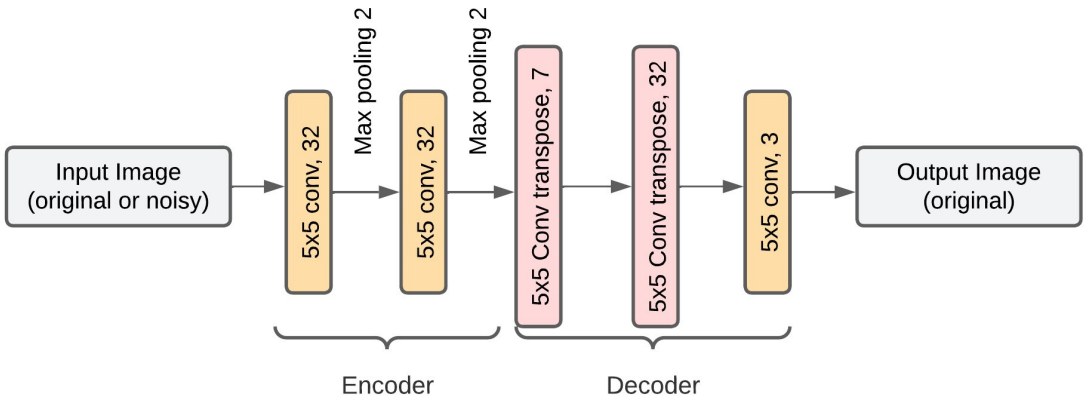


Figure 3.5: CAE architecture for the improved ELM model.

For the training process, we leveraged the Adam optimization algorithm, setting the learning rate to 0.001 and the batch size to 64. Furthermore, the CAE models were trained for a total of 200 epochs to ensure convergence.

3.4.3.4 Parameter settings

In our analysis, we aimed to evaluate the effectiveness of the GWO-MVO algorithm in optimizing the initialization of weights and biases for the ELM classifier compared to the standalone GWO and PSO algorithms. By empirically tuning the control parameters of these metaheuristics, we sought to ensure fair and optimal performance for each algorithm while also considering the computational resources required in terms of learning time and memory usage. Additionally, the performance of the ELM classifier was compared with that of the KNN and SVM classifiers. Table 3.7 presents the values of all parameters used in our experimental study.

Table 3.7: Parameter settings for the ELM model improvement contribution.

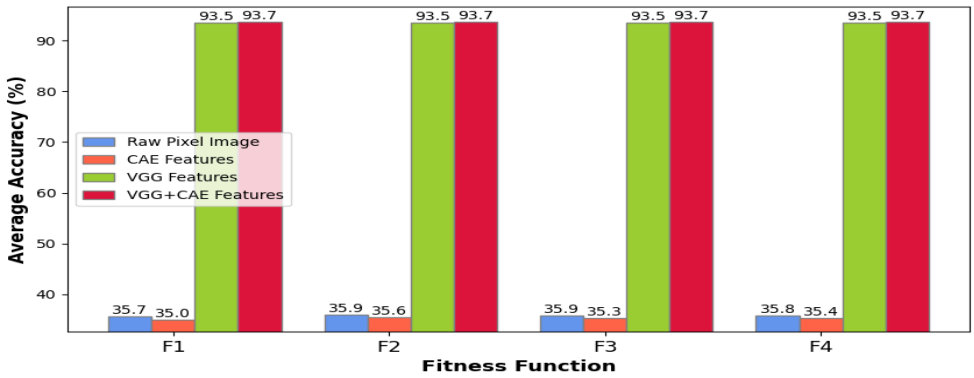
Method	Parameter	value
GWO-MVO	#iterations	20
	#agents	7
	p_1	2
	p_2	10
GWO	#iterations	20
	#agents	7
PSO	#iterations	20
	c_1	2
	c_2	2
	w_{min}	0.2
	w_{max}	0.9
ELM	Activation function	relu
	#hidden nodes	100, 200, 300, 400, 500, and 1000
KNN	#neighbors	5
	Metric	euclidean
SVM	Kernel	rbf
	Gamma	0.001
	C	100

3.4.3.5 Experimental results

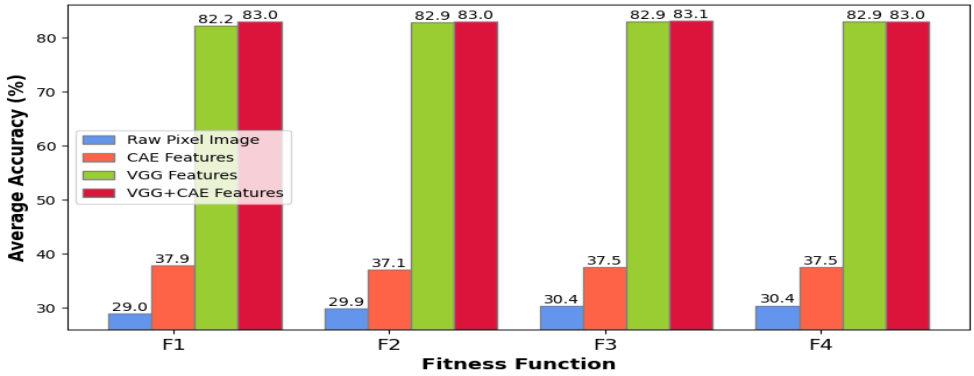
For the remainder of this section, we present the results obtained from our tests. We conducted ten separate runs for each method under evaluation to ensure robustness and reliability.

Input feature impact

To assess the impact of different input feature representations, we contrasted four distinct input types: the raw pixel image, the CAE feature vector, the VGG feature vector, and the combined VGG-CAE feature vector. Each ELM model is initialized using the GWO-MVO technique with a fixed number of 100 hidden nodes. Figures 3.6 and 3.7 illustrate the performance of these four approaches on the CIFAR-10 and CIFAR-100 datasets, respectively, in two scenarios: original data and data with noise.



(a) Original data.

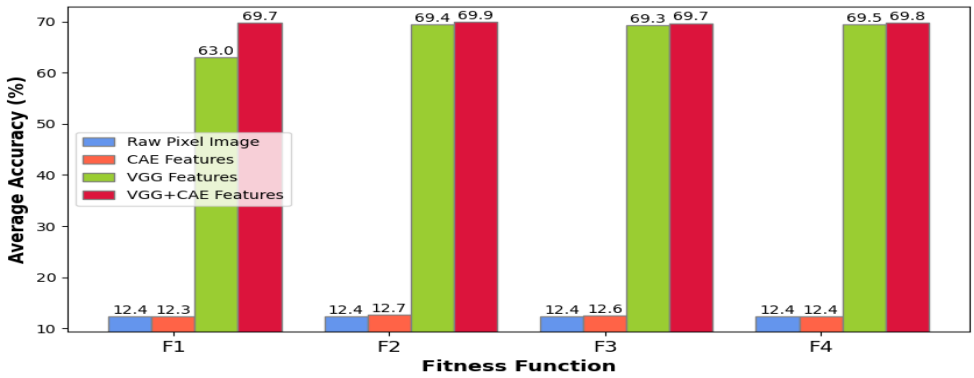


(b) Noisy data.

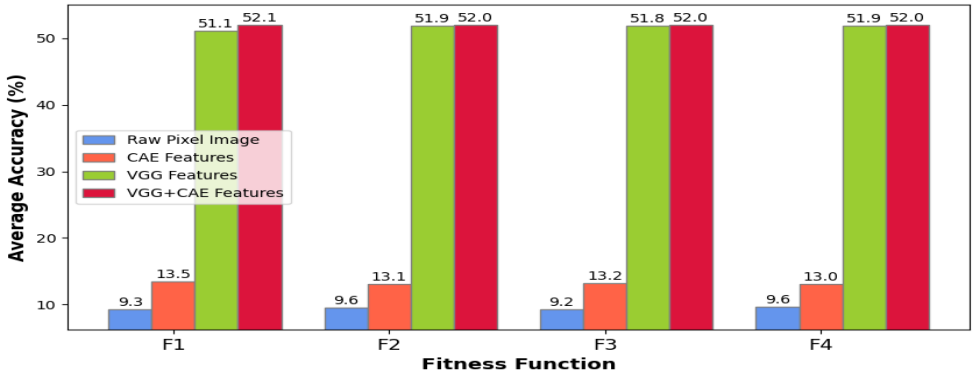
Figure 3.6: Comparative performance of the ELM model initialized by GWO-MVO using 100 hidden nodes based on different input types on the CIFAR-10 dataset (both test versions).

On both CIFAR-10 and CIFAR-100 datasets, across different evaluation scenarios (original and noisy data) and fitness functions, we observe that the VGG features and the combined VGG+CAE features consistently outperform the other input types. However, the combined VGG+CAE features demonstrate a slightly better performance than the VGG features alone. In addition, the CAE features yield the lowest results on the original datasets, while feeding raw pixels directly performs the worst on the noisy datasets.

In ML, the model's capability to accurately fit new data is a crucial aspect. Figure 3.8 depicts the generalization gap between two feature vectors, VGG and VGG+CAE, across the four fitness functions. The outcomes showcased in the figure were derived with 100 hidden nodes. It can be inferred that combining the VGG and CAE features



(a) Original data.



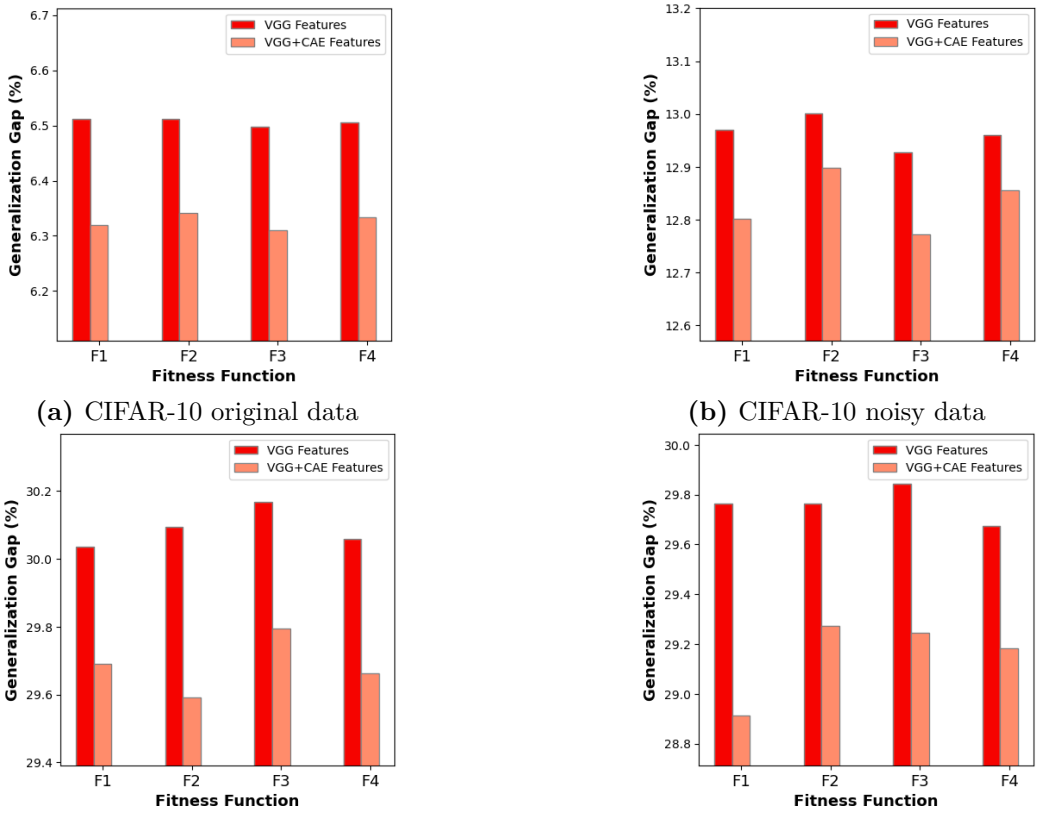
(b) Noisy data.

Figure 3.7: Comparative performance of the ELM model initialized by GWO-MVO using 100 hidden nodes based on different input types on the CIFAR-100 dataset (both test versions).

reduces the generalization gap observed when using only VGG features, indicating better generalization capability.

ELM initialization impact

To analyze the impact of different initialization methods for the input parameters of the ELM model, we employed box plots, which effectively represent data dispersion and identify potential outliers. We compared five initialization methods: random initialization and the proposed GWO-MVO method with each of the four fitness functions (F_1, F_2, F_3, and F_4). We used the combined VGG+CAE feature vector for all methods as input and a fixed number of 100 hidden nodes in the ELM model. The comparison presented in Figure 3.9 demonstrates the superiority of the GWO-MVO



(a) CIFAR-10 original data

(b) CIFAR-10 noisy data

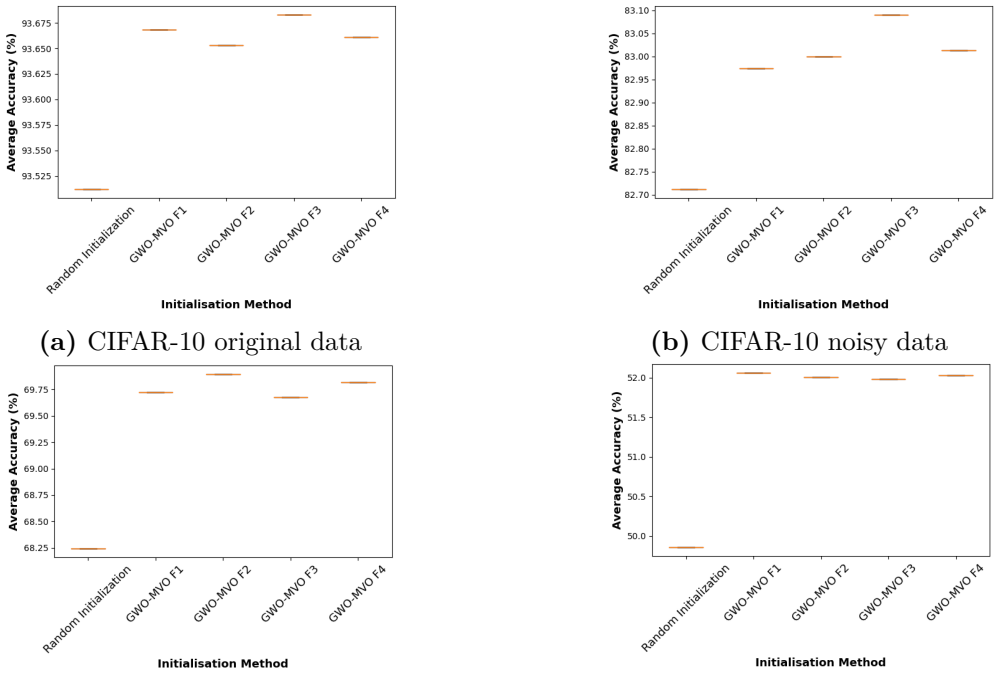
(c) CIFAR-100 original data

(d) CIFAR-100 noisy data

Figure 3.8: Generalization gaps on CIFAR-10 and CIFAR-100 based on the feature input type.

method, regardless of the fitness function employed. In addition to emphasizing the importance of selecting appropriate input features, these results strongly highlight the necessity of initializing the ELM with suitable input parameters through an effective optimization technique.

Figure 3.10 illustrates the generalization gap observed between the two versions (original and noisy) of the CIFAR-10 and CIFAR-100 datasets. The figure compares two initialization methods: random initialization and our proposed GWO-MVO method, using the F_4 fitness function. These results were obtained using the combined VGG+CAE feature vector as input and employing 100 hidden nodes in the ELM model. Thus, in addition to the positive impact of the VGG+CAE input features on the generalization capability (as discussed previously), our GWO-MVO initialization method further reduces the generalization gap. This observation reinforces the crucial significance of adequately initializing the ELM model's parameters through an effective optimization technique.



(a) CIFAR-10 original data

(b) CIFAR-10 noisy data

(c) CIFAR-100 original data

(d) CIFAR-100 noisy data

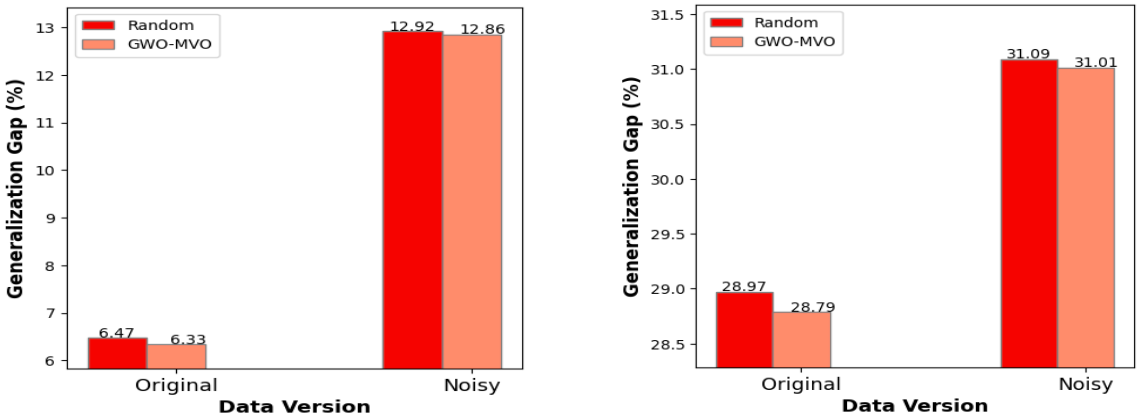
Figure 3.9: Test accuracy values across ten executions for studied ELM initialization techniques with VGG+CAE input feature vector and 100 hidden nodes.

Fitness function impact

Figures 3.11 and 3.12 provide an overview of the performance of the ELM model initialized by the GWO-MVO method across the four fitness functions (F_1, F_2, F_3, and F_4) for the CIFAR-10 and CIFAR-100 datasets, respectively. The evaluation includes the four input feature types presented in Section 7.4.1, employing 100 hidden nodes in the ELM model. Both figures present outcomes for two evaluation scenarios: the original and noisy datasets. The findings reveal a notable similarity in the performance of the four fitness functions, posing challenges in determining a definitive best function. However, the F_1 function demonstrates heightened instability since its results are inconsistent with the other functions in some cases.

Hidden node impact

Figure 3.13 depicts the average accuracy achieved on the test set, based on the number of hidden nodes, for both the original and noisy versions of the CIFAR-10 and CIFAR-100 datasets. The comparison focuses on the VGG and VGG+CAE input features. We examine both random initialization and initialization using the GWO-MVO method with the F_4 fitness function. The results indicate that the network size, represented by the number of hidden nodes, significantly influences performance.



(a) CIFAR-10. **(b)** CIFAR-100.

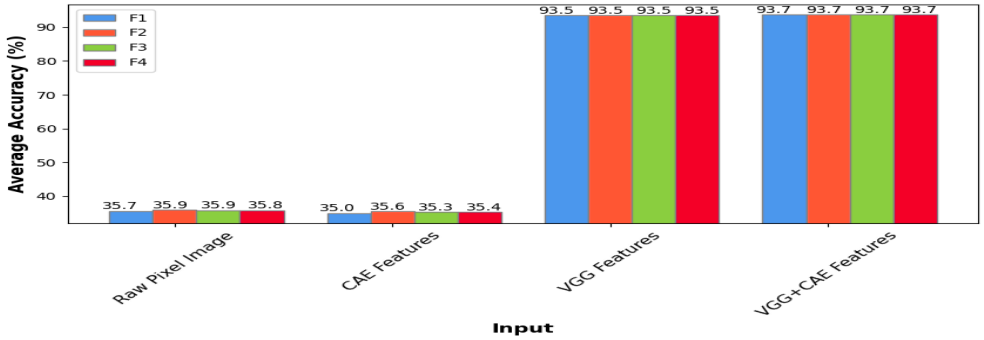
Figure 3.10: Generalization gaps on CIFAR-10 and CIFAR-100 based on the initialization type.

Generally, augmenting the number of hidden nodes tends to improve accuracy. However, this trend exhibits inconsistency, particularly with the original data of the CIFAR-10 dataset. Conversely, while approaches utilizing VGG+CAE features demonstrate superior performance, the intelligent initialization of the ELM model's input parameters via the GWO-MVO method remains pivotal. It enables achieving high accuracies even with relatively fewer hidden nodes.

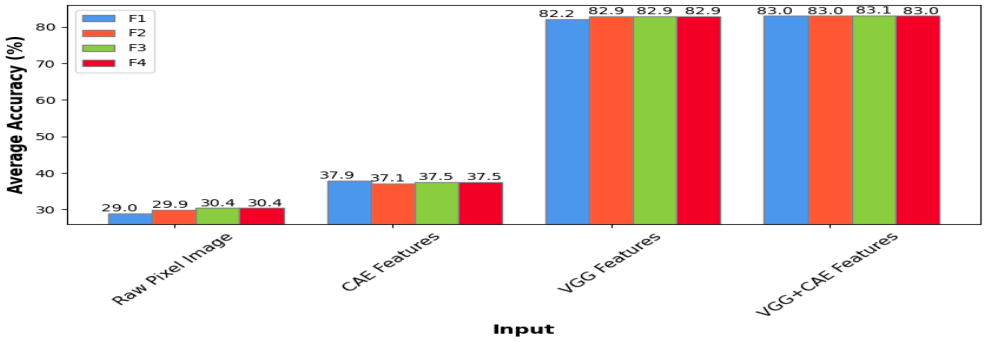
Table 3.8 presents the best test set performance achieved by each method, with the best and worst results highlighted. Our proposed approach demonstrated superior performance across original and noisy versions of the CIFAR-10 and CIFAR-100 datasets.

For CIFAR-10, our method with the F_4 function achieved the best test accuracies of 93.71% $\pm$ 0.06 (original) and 83.35% $\pm$ 0.07 (noisy). It is worth highlighting that the optimal result for the original version was obtained using only 200 hidden nodes, showcasing the efficacy of our approach in minimizing resource requirements and training time. On the original CIFAR-100, the best average accuracy of 71.2% $\pm$ 0.18 was achieved with F_3, while F_2 performed best (54.23% $\pm$ 0.11) on noisy data.

Notably, models fed directly with raw input images without feature extraction performed the worst, especially when using GWO-MVO with F_1. This highlights the importance of optimizing ELM initialization with optimal input features for best performance.

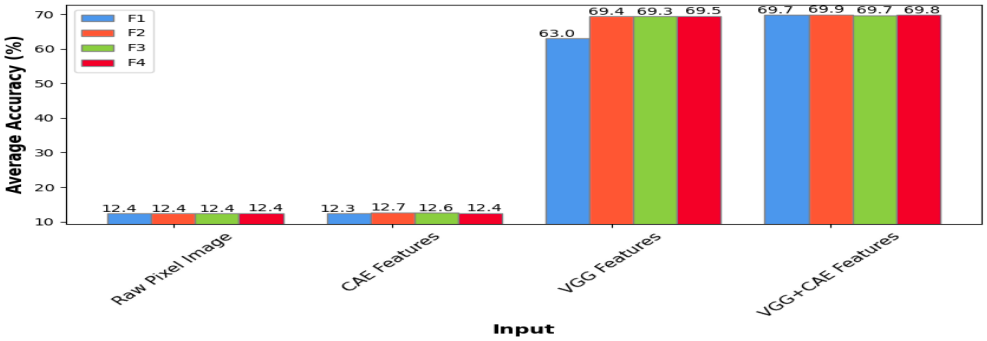


(a) Original data.

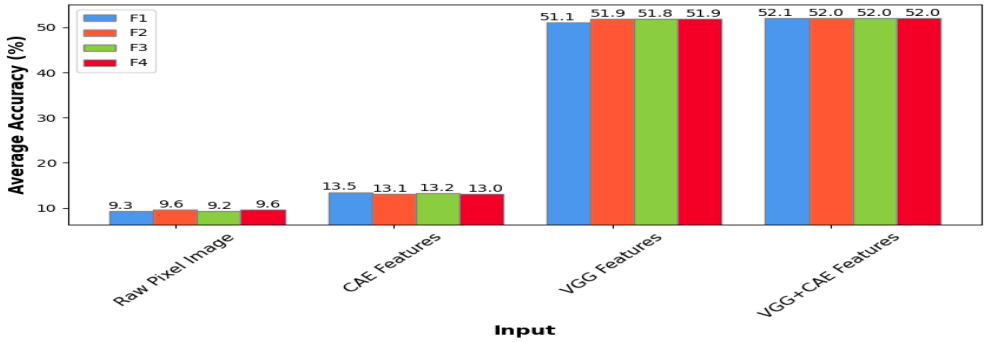


(b) Noisy data.

Figure 3.11: ELM model performance using GWO-MVO initialization and 100 hidden nodes on the four fitness functions for the CIFAR-10 dataset (both test versions).

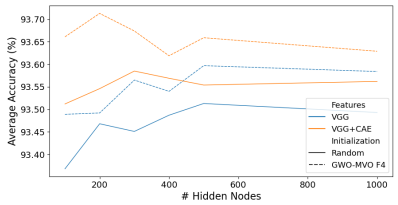


(a) Original data.

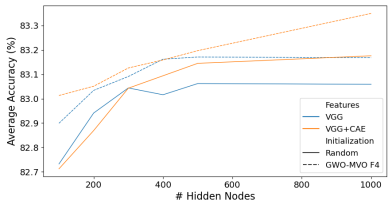


(b) Noisy data.

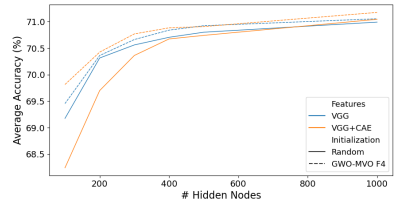
Figure 3.12: ELM model performance using GWO-MVO initialization and 100 hidden nodes on the four fitness functions for the CIFAR-100 dataset (both test versions).



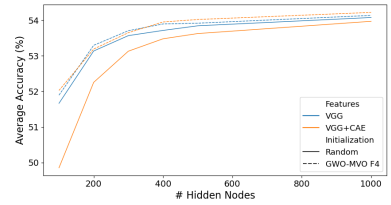
(a) CIFAR-10 original data



(b) CIFAR-10 noisy data



(c) CIFAR-100 original data



(d) CIFAR-100 noisy data

Figure 3.13: Average accuracy over the two test sets based on the number of ELM hidden nodes.

Table 3.8: Best average accuracy and standard deviation (in percentages) for ELM model optimization.

Original Version

Input Feature	# input Features	Initialisation	Fitness Function	CIFAR-10 # Hidden Nodes	CIFAR-10 Best Result	CIFAR-100 # Hidden Nodes	CIFAR-100 Best Result
Raw Pixel Image	3072	Random	-	1000	44.33 ± 0.23	1000	18.13 ± 0.33
		GWO-MVO	F_1	500	39.41 ± 0.27	500	14.3 ± 0.21
		GWO-MVO	F_2	1000	44.41 ± 0.31	500	14.34 ± 0.15
		GWO-MVO	F_3	1000	44.27 ± 0.4	1000	18.25 ± 0.23
		GWO-MVO	F_4	1000	44.47 ± 0.24	1000	18.25 ± 0.16
CAE	1793	Random	-	1000	46.2 ± 0.39	1000	18.92 ± 0.25
		GWO-MVO	F_1	1000	49.23 ± 0.23	1000	20.27± 0.18
		GWO-MVO	F_2	1000	49.39 ± 0.25	1000	20.24 ± 0.15
		GWO-MVO	F_3	1000	49.37 ± 0.26	1000	20.14 ± 0.26
		GWO-MVO	F_4	1000	49.3 ± 0.2	1000	20.21 ± 0.22
VGG	512	Random	-	500	93.51 ± 0.03	1000	70.99 ± 0.04
		GWO-MVO	F_1	1000	93.6 ± 0.03	1000	70.58 ± 1.55
		GWO-MVO	F_2	300	93.58 ± 0.06	1000	71.07 ± 0.1
		GWO-MVO	F_3	500	93.60 ± 0.08	1000	71.07 ± 0.05
		GWO-MVO	F_4	500	93.6 ± 0.06	1000	71.06 ± 0.11
VGG+CAE	2305	Random	-	300	93.59 ± 0.06	1000	71.05 ± 0.11
		GWO-MVO	F_1	300	93.69 ± 0.04	200	70.44 ± 0.11
		GWO-MVO	F_2	1000	93.68 ± 0.05	1000	71.16 ± 0.1
		GWO-MVO	F_3	500	93.68 ± 0.04	1000	**71.2 + 0.18**
		GWO-MVO	F_4	200	**93.71 ± 0.06**	1000	71.18 ± 0.11

Noisy Version

Input Feature	# input Features	Initialisation	Fitness Function	CIFAR-10 # Hidden Nodes	CIFAR-10 Best Result	CIFAR-100 # Hidden Nodes	CIFAR-100 Best Result
Raw Pixel Image	3072	Random	-	1000	36.99 ± 0.26	1000	12.86 ± 0.25
		GWO-MVO	F_1	1000	34.79 ± 0.32	300	11.18 ± 0.14
		GWO-MVO	F_2	1000	37.75 ± 0.29	500	10.74 ± 0.2
		GWO-MVO	F_3	1000	36.82 ± 0.31	1000	13.48 ± 0.28
		GWO-MVO	F_4	1000	38.02 ± 0.37	300	11.28 ± 0.22
CAE	1793	Random	-	1000	45.99 ± 0.35	1000	18.74 ± 0.27
		GWO-MVO	F_1	1000	47.07 ± 0.18	1000	19.01 ± 0.1
		GWO-MVO	F_2	1000	47.46 ± 0.31	1000	19.03 ± 0.1
		GWO-MVO	F_3	1000	46.57 ± 0.26	1000	18.82 ± 0.1
		GWO-MVO	F_4	1000	47.43 ± 0.41	1000	19.06 ± 0.21
VGG	512	Random	-	500	83.06 ± 0.07	1000	54.08 ± 0.14
		GWO-MVO	F_1	1000	83.2 ± 0.05	1000	54.11 ± 0.1
		GWO-MVO	F_2	500	83.16 ± 0.12	1000	54.08 ± 0.1
		GWO-MVO	F_3	1000	83.22 ± 0.07	1000	54.03 ± 0.1
		GWO-MVO	F_4	500	83.17 ± 0.07	1000	54.14 ± 0.1
VGG+CAE	2305	Random	-	1000	83.18 ± 0.08	1000	53.97 ± 0.14
		GWO-MVO	F_1	100	82.97 ± 0.08	500	54.06 ± 0.1
		GWO-MVO	F_2	1000	83.28 ± 0.09	1000	**54.23 ± 0.11**
		GWO-MVO	F_3	1000	83.3 ± 0.11	1000	54.21 ± 0.17
		GWO-MVO	F_4	1000	**83.35 ± 0.07**	1000	54.22 ± 0.07

Ablation study

To assess the effectiveness of the key components of our approach, we performed ablation studies. Tables 3.9 and 3.10 present comprehensive details, including standard deviation, mean accuracy, best accuracy, worst accuracy, and the rate of generalization gap for both CIFAR10 and CIFAR100 datasets. Notably, the results from both datasets across both versions exhibit striking similarities. The use of VGG features alone substantially improves the test accuracies but suffers from overfitting issues, as indicated by the increased generalization gap. Incorporating CAE features mitigates this issue by enhancing the generalization capability, although it may slightly reduce the accuracy in some cases. Ultimately, optimizing the ELM model's parameters through the GWO-MVO method leads to further improvements in prediction stability, as reflected by the positive changes in all evaluated metrics.

Table 3.9: Ablation study of key components in VGG+CAE+GWO-MVO-ELM using 100 hidden nodes on CIFAR-10.

Original					
Settings	SD	Mean Acc (%)	Best Acc (%)	Worst Acc (%)	Mean GG (%)
ELM	0.54	34.46	35.27	33.5	**0.11**
+ VGG	0.1	93.37	93.57	93.24	6.62
+ CAE	0.08	93.51	93.62	93.39	6.47
+ GWO-MVO (F_4)	**0.07**	**93.66**	**93.8**	**93.59**	6.33
Noisy					
ELM	0.54	28.61	29.61	27.63	**0,39**
+ VGG	0.12	82.73	82.93	82.55	13.02
+ CAE	0.17	82.71	82.92	82.37	12.92
+ GWO-MVO (F_4)	**0.08**	**83.01**	**83.18**	**82.9**	12.86

SD: Standard Deviation, Acc: Accuracy, GG: Generalization Gap.

Table 3.10: Ablation study of key components in VGG+CAE+GWO-MVO-ELM using 100 hidden nodes on CIFAR-100.

Original					
Settings	SD	Mean Acc (%)	Best Acc (%)	Worst Acc (%)	Mean GG (%)
ELM	0.22	11.2	11.63	10.98	**1.18**
+ VGG	**0.09**	69.18	69.35	69.05	30.28
+ CAE	0.27	68.24	68.59	67.68	30.86
+ GWO-MVO (F_4)	0.11	**69.82**	**69.99**	**69.66**	29.66
Noisy					
ELM	0.29	8.34	8.91	7.89	**1.44**
+ VGG	0.33	51.67	52.1	50.96	29.61
+ CAE	0.37	49.86	50.37	48.97	28.29
+ GWO-MVO (F_4)	**0.16**	**52.03**	**52.26**	**51.79**	29.18

SD: Standard Deviation, Acc: Accuracy, GG: Generalization Gap.

To validate our approach, GWO-MVO-ELM is compared against two other metaheuristic-based ELM models: PSO-ELM and GWO-ELM. Table 3.11 displays the average accuracy results, using the VGG+CAE feature vector to feed these classifiers. Across all datasets, the GWO-MVO-ELM achieved the highest average accuracy regardless of the fitness function employed, demonstrating its superiority.

Table 3.11: Comparing average accuracy between different metaheuristic-based ELM approaches using 100 Hidden Nodes.

Dataset	Data version	Fitness function	PSO	GWO	GWO-MVO
CIFAR-10	Original	F_1	93.46	93.44	**93.67**
		F_2	93.57	93.45	**93.65**
		F_3	93.42	93.47	**93.68**
		F_4	93.43	93.49	**93.66**
	Noisy	F_1	82.78	82.83	**82.97**
		F_2	82.98	82.83	**83**
		F_3	82.72	82.85	**83.09**
		F_4	82.91	82.75	**83.01**
CIFAR-100	Original	F_1	69.45	69.59	**69.73**
		F_2	69.59	69.57	**69.9**
		F_3	69.28	69.52	**69.68**
		F_4	69.28	69.63	**69.82**
	Noisy	F_1	52.03	51.86	**52.06**
		F_2	51.82	51.88	**52.01**
		F_3	51.88	51.91	**51.99**
		F_4	51.56	51.86	**52.03**

Additional experiments are conducted with other classifiers - KNN, SVM, standard ELM, and our optimized ELM - to further verify the proposed method's efficacy. To obtain an accurate estimate of the ELM training time, 1000 hidden nodes were used. The results, including average accuracy and training time for both dataset versions, are reported in Table 3.12. We note that training time was ignored for KNN due to its simplicity. Using VGG+CAE features for each classifier, our method with F_2, F_3, and F_4 outperforms other classifiers in accuracy, especially on noisy data (2% over KNN on CIFAR-100). However, F_1 shows the worst accuracy here, indicating suboptimality with many hidden nodes.

Table 3.12: Comparison between different classifiers performance.

Method	CIFAR-10				CIFAR-100			
	Original		Noisy		Original		Noisy	
	Acc (%)	TT (s)	Acc (%)	TT (s)	Acc (%)	TT (s)	Acc (%)	TT (s)
VGG+CAE+KNN	93.56	N/A	82.61	N/A	70.78	N/A	52.26	N/A
VGG+CAE+SVM	93.54	10.88	82.41	134.3	70.92	135.18	52.91	322.85
VGG+CAE+ELM	93.56	**8.77**	83.18	**7.66**	71.05	**8.47**	53.97	**7.96**
Proposed (F_1)	85.32	1131.54	61.28	909.71	36.25	701.7	22.44	614.32
Proposed (F_2)	**93.68**	1234.53	83.28	1245.72	71.16	1293.38	**54.23**	1267.83
Proposed (F_3)	93.65	5100.58	83.3	5095.76	**71.2**	5302.27	54.21	5301.16
Proposed (F_4)	93.63	5106.73	**83.35**	5089.35	71,18	5238.393	54.22	5300.96

1000 hidden nodes were used for the ELM classifier.
Acc: Accuracy, TT: Training Time.

On the other hand, our approach exhibits the longest training time, especially for F_3 and F_4, which use CV. Although the standard ELM is the fastest, optimizing its weights/biases adds substantial computational costs. This aspect requires further consideration.

3.4.3.6 Comparison with previous works

To conduct a comprehensive evaluation, we compare our method to similar works of equivalent complexity on the CIFAR-10 and CIFAR-100 datasets. The proposed approach is benchmarked against several state-of-the-art techniques, including VGG16 [107], DCNN [92], VGG-16-pruned-A [68], VGG16-AFP-E [31], DCT-Net[1] [44], SNN-VGG-15 [94], and CIFAR-VGG [74]. The results are presented in Table 3.13, with the best accuracies in bold.

Our approach achieves the highest classification accuracy on original and noisy data, outperforming all techniques. This improvement is particularly notable for the noisy data, where our method significantly surpasses others. We obtained 93.8% and 83.47% accuracy on standard and noisy CIFAR-10, respectively. For CIFAR-100, our approach achieved 71.4% and 54.57% accuracy on original and noisy data. These results demonstrate our proposed technique's strong performance and noise robustness, validating its efficacy and highlighting superior stability on noisy data compared to existing methods.

[1]DCT: Discrete Cosine Transform

Table 3.13: Comparison with similar image classification methods on CIFAR-10 and CIFAR-100.

Work	Feature extraction network	Approach	CIFAR-10				CIFAR-100			
			Original		Noisy		Original		Noisy	
			Acc (%)	Err (%)	Acc (%)	Err (%)	Acc (%)	Err (%)	Acc (%)	Err (%)
Simonyan and Zisserman [107]	VGG16	-	88.1	11.9	-	-	63.5	36.5	-	-
Nazaré et al. [92]	DCNN	Without using denoising methods	81.92	18.08	66.08	33.92	-	-	-	-
		With denoising methods	-	-	62.77	37.23	-	-	-	-
Li et al. [68]	VGG16	Pruning Filters	93.4	6.6	-	-	-	-	-	-
Ding et al. [31]	VGG16	Pruning Filters	92.94	7.06	-	-	-	-	-	-
Hossain et al. [44]	VGG16	DCT	88.44	11.56	62.92	37.08	67.5	32.5	49.81	50.19
Niu et al. [94]	VGG15	Hard reset	-	-	-	-	62.85	37.15	-	-
		Soft reset	-	-	-	-	63.09	36.91	-	-
		TF-reset reset	-	-	-	-	64.27	35.73	-	-
Liu and Deng [74]	CIFAR-VGG	-	93.56	6.44	81.57	18.43	70.48	29.52	52.37	47.63
Our work	CIFAR-VGG + CAE	GWO-MVO F_1	93.76	6.24	83.29	16.71	71.36	28.64	54.37	45.63
		GWO-MVO F_2	93.77	6.23	83.4	16.6	71.36	28.64	54.43	45.57
		GWO-MVO F_3	93.78	6.22	83.44	16.56	**71.4**	**28.6**	**54.57**	**45.43**
		GWO-MVO F_4	**93.8**	**6.2**	**83.47**	**16.53**	71.3	28.7	54.37	45.63

Acc: Best accuracy, Err: Best error rate.

3.4.4 Discussion and analysis

The synergistic integration of CNN, CAE, and optimized ELM enables each model to compensate for the weaknesses of the others while benefiting from their complementary advantages. This framework maximizes ELM's classification capabilities by enhancing its input features and hyperparameters. The key to success is consistent prediction stability, maintenance of generalization performance, and reduction in overfitting achieved through the proposed techniques.

While deep feature extraction helps improve training time, optimizing ELM's weights and biases remains computationally expensive. Further enhancements in training efficiency, such as integrating parallelism into ELM tuning, could be pursued. Nonetheless, this study puts forth a practical optimization approach that unlocks the capabilities of ELMs via a multi-pronged methodology, broadening their applicability to challenging computer vision tasks. Even with room for improvement in training time, the techniques presented provide a promising template for optimizing ELM-based solutions.

3.5 Chapter summary

In this chapter, we introduced a novel hybrid optimization algorithm, GWO-MVO, which combines GWO with the MVO method. This approach aims to improve GWO's optimization ability, particularly in hyperparameter optimization tasks.

The effectiveness of the proposed hybrid GWO-MVO algorithm was initially evaluated to estimate the optimal dropout probability for CNNs. Experimental results demonstrate the algorithm's proficiency in optimizing dropout probabilities for CNNs, consistently surpassing the baseline GWO and achieving competitive performance compared to other state-of-the-art methods on benchmark datasets.

Additionally, we proposed a novel approach to enhance ELM performance for image classification tasks. The contribution involved optimizing weight and bias initialization through GWO-MVO and improving input feature quality using CNN and CAE. This approach enhances prediction stability, maintains generalization performance, and reduces overfitting by maximizing classification capabilities through improved input features and hyperparameters.

Through this chapter, we demonstrated the competitiveness of GWO-MVO on benchmark hyperparameter optimization problems. The results provide valuable insights into designing practical metaheuristic algorithms for automating the ML model development process.

Chapter 4

Quantum Inspired GWO for CNN Hyperparameter Optimization

4.1 Introduction

Quantum Computing (QC) is an emerging interdisciplinary field combining mathematics, physics, and computer science. Integrating quantum principles into metaheuristic algorithms has significantly improved the performance of traditional metaheuristics. Quantum-inspired metaheuristics demonstrate faster convergence and better balance between exploration and exploitation during optimization. This enables quantum-inspired variants to outperform conventional metaheuristic optimizers on many problems.

This chapter introduces a novel approach called qGWO-CNN that uses a novel quantum-inspired variant of GWO to search for optimal CNN hyperparameters efficiently. The proposed qGWO method also incorporates elements from a hybrid GWO-MVO algorithm previously shown to be effective.

The outline of this chapter is as follows: Section 4.2 explains the fundamental concepts of the proposed approach. Section 4.3 presents experiments and results evaluating qGWO-CNN. Section 4.4 provides a detailed discussion and analysis of the results, including limitations and future work.

4.2 Proposed approach

4.2.1 qGWO algorithm

The core of our algorithm is based on principles of quantum dynamics used in QPSO [112]. Our algorithm distinguishes itself from prior versions of quantum Grey

Wolf Optimizer (qGWO) by harnessing the effectiveness of GWO-MVO. Like QPSO, we employ the Monte Carlo method to derive the update function for each wolf's position (Section 1.3.3.3). The j^{th} dimension ($1 \leq j \leq N$) of the position of the wolf i ($1 \leq i \leq M$) is updated as follows:

$$X^j_{i,t+1} = g^j_{i,t} \pm \frac{L^j_{i,t}}{2} ln(\frac{1}{u^j_{i,t+1}}) \tag{4.1}$$

where $u^j_{i,t+1}$ is a sequence of random numbers uniformly distributed in $[0,1]$, varying for each position and dimension across iterations. The local attractor $g^j_{i,t}$ for a wolf is determined by the average of the best three wolves, denoted as α, β, and δ, without additional exploitation. Hence, the expression for $g^j_{i,t}$ is:

$$g^j_{i,t} = x^j_{m,t} - \frac{1}{3}(A_{\alpha,t} \cdot D^j_{\alpha,t} + A_{\beta,t} \cdot D^j_{\beta,t} + A_{\delta,t} \cdot D^j_{\delta,t}) \tag{4.2}$$

Here, $x^j_{m,t}$ represents the j^{th} mean dimension of the position of the top three wolves (α, β, δ) at iteration t. D_k retains its definition as in traditional GWO. Drawing from our prior GWO-MVO algorithm, we derive the parameter A_k using a similar approach, where 'a' undergoes a non-linear decrease to maintain a favorable equilibrium between exploration and exploitation (Chapter 3).

$$a = 2 \times (1 - \frac{t^{\frac{1}{p_1}}}{T^{\frac{1}{p_1}}}) \tag{4.3}$$

Here, 't' denotes the current iteration, while 'T' represents the maximum number of iterations. The constant p_1 regulates the degree of exploitation across the iterations.

To strengthen the focus on the exploitation towards the alpha wolf rather than the average position of the gray wolves as in [118], the value of $L^j_{i,t}$ can be established through:

$$L^j_{i,t} = 2 \cdot \mu \cdot |X^j_{i,t} - X^j_{\alpha,t}| \tag{4.4}$$

μ is not a fixed constant as in standard QPSO. Instead, it decreases non-linearly over iterations, diverging from previous quantum-inspired GWO algorithms. Drawing inspiration from the computation of the MVO parameter (TDR) employed in GWO-MVO (Section 1.3.3.2), μ is defined as shown in Equation 4.5.

$$\mu = 1 - \frac{t^{\frac{1}{p_2}}}{T^{\frac{1}{p_2}}} \tag{4.5}$$

p_2 serves as a control parameter governing the level of the exploitation around wolf α. Ultimately, the update of a wolf's position is done as follows:

$$X_{i,t+1} = \begin{cases} g^j_{i,t} + \mu \cdot |X^j_{i,t} - X^j_{\alpha,t}| \times ln(\frac{1}{u^j_{i,t+1}}) & \text{if } R > 0.5 \\ g^j_{i,t} - \mu \cdot |X^j_{i,t} - X^j_{\alpha,t}| \times ln(\frac{1}{u^j_{i,t+1}}) & \text{else} \end{cases} \tag{4.6}$$

where R is a random number selected from a uniform distribution within the range $[0,1]$.

Algorithm 6 outlines the proposed qGWO.

Algorithm 6 Quantum Inspired Grey Wolf Optimizer

Input: T : max number of iterations, n: population size, p_1, p_2, f : fitness function

Output: X_α

Initialize the grey wolf population $X_i\,(i = 1,2,..,n)$
Evaluate the population
Select the leaders X_α, X_β and X_δ
$t \leftarrow 0$
while $t < T$ **do**
    Update a using Equation 1.12
    for each individual **do**
        Determine the position of the current wolf's local attractor by Equation 4.2
        Update the position of the current wolf by Equation 4.6
    end for
    Evaluate the new population
    Update the leaders X_α, X_β and X_δ
    $t \leftarrow t + 1$
end while
return X_α

Our proposed qGWO algorithm's time complexity is determined similarly to the GWO-MVO algorithm (refer to Section 3.2.3). Consequently, for a given maximum iteration count, T, qGWO's complexity is expressed as $O(T \times N \times (m + f(S))$, where N denotes the population size, m signifies the dimensionality of a solution candidate, and $f(S)$ represents the time complexity of the fitness function.

4.2.2 qGWO-CNN

The proposed qGWO-CNN method uses qGWO to optimize specific hyperparameters like kernel size and padding in a predefined CNN architecture. Specifically, the

qGWO algorithm is employed to determine these hyperparameter values. The qGWO algorithm updates the positions of the wolves based on the positions of the top three best-performing wolves (leaders) in the population. The update rules ensure that the hyperparameter values stay within specified ranges. Evaluating each wolf entails generating a corresponding CNN model, training it on a training sample, and assessing its performance on a validation sample. The optimization process flow is illustrated in Figure 4.1.

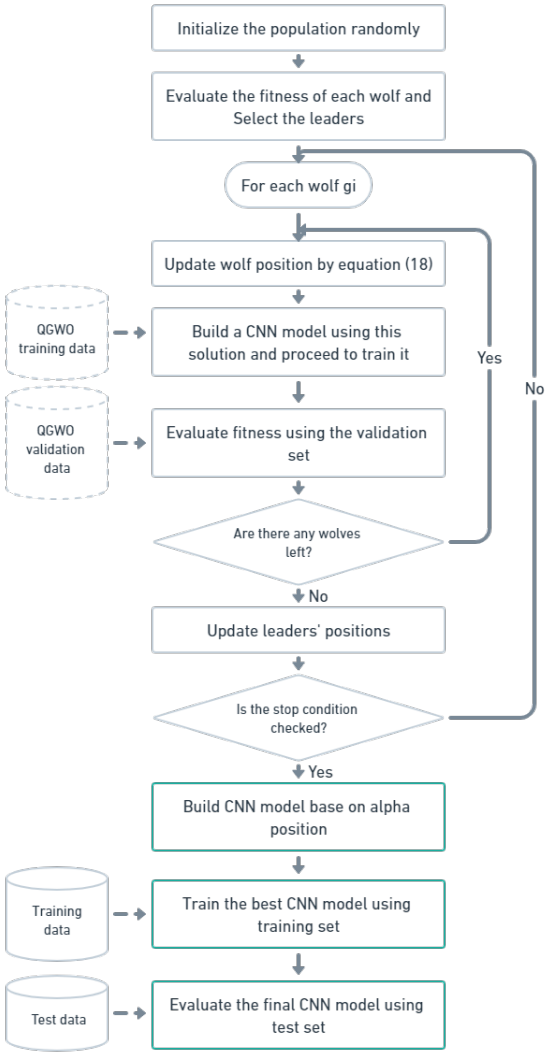


Figure 4.1: Flowchart of the proposed qGWO-CNN approach.

The following provides detailed explanations of the approach's key aspects, including solution representation, population initialization, and the fitness function.

4.2.2.1 Solution encoding

In qGWO-CNN, each wolf represents a potential solution defined by a set of hyperparameter values. This solution is encoded as a numerical vector, where each element corresponds to one hyperparameter being optimized. The length of the vector equals the number of hyperparameters. For example, consider a CNN with three hyperparameters: convolution kernel size, pooling kernel size, and dropout rate. A possible solution denoting a wolf's position could be (5, 3, 0.5). The first element, 5, refers to a convolution kernel size of 5, the second element, 3, refers to a pooling kernel size of 3, and the third element, 0.5, refers to a dropout rate of 0.5. The data type (integer or real number) of each vector element matches the type of the associated hyperparameter. In this way, the wolf position vectors fully specify the hyperparameter configurations encoded by each wolf for evaluation.

4.2.2.2 Population initialization

When initializing the wolf population, the hyperparameter values are randomly generated within predefined lower and upper bounds defining each hyperparameter's search space. The limits help constrain the range of possible values for a hyperparameter based on its characteristics. For example, kernel sizes may have minimum and maximum size constraints, while dropout rates would be bounded between 0 and 1.

4.2.2.3 Fitness function

A CNN model is constructed to evaluate a wolf's solution using the hyperparameter values encoded in the wolf's position vector. This CNN model is then trained for a fixed number of epochs on the training data. The trained model is tested on the validation set to compute the classification error rate (CER) as the fitness score. A lower CER indicates a better CNN model. Therefore, training and evaluating a corresponding CNN model assesses each potential hyperparameter solution represented by a wolf. The CER performance on the validation set provides a quantitative fitness estimate for that solution.

4.3 Experiments

Experiments were implemented in Python using Keras, scikit-learn, NumPy, SciPy, and pandas. The programs were run on a desktop with an AMD Ryzen 7 3700X CPU, 16GB RAM, Nvidia RTX 2060 GPU, and Ubuntu 18.04 OS.

To evaluate qGWO-CNN in a typical CNN setting, the AlexNet architecture was chosen as a moderately complex CNN model. The CIFAR-10 dataset was used as a standard benchmark for evaluating CNNs. Since training CNNs is computationally expensive, the number of training epochs was limited to five per evaluation. Models were trained using only 1/5 of the training set (i.e., 10,000 samples), with 128 instances for validation due to its typical complexity level.

4.3.1 Selected hyperparameters

Given the computational constraints of our test environment, we focused on optimizing three key hyperparameters - the kernel size of the first convolutional layer, the stride size of the first convolutional layer, and the dropout rates. The first convolution layer is critical for feature extraction from the raw input data. Dropout is an effective technique for reducing overfitting in CNNs. The AlexNet architecture has two dropout layers, so four hyperparameters were optimized. Table 4.1 summarizes the optimized hyperparameters along with the minimum and maximum values defining each search space.

Table 4.1: CNN hyperparameters to optimize.

Hyperparameter	Type	Range
Kernel size	Integer	[3,11]
Stride size	Integer	[1,5]
Dropout rate	Real	[0,1]

4.3.2 Parameters tuning

Experiments were conducted to select appropriate input parameters for the proposed qGWO-CNN algorithm. For comparison, the GWO and GWO-MVO algorithms were also tested. The parameters for GWO-CNN, GWO-MVO-CNN, and qGWO-CNN are detailed in Table 4.2. These parameter settings enabled a fair comparison between qGWO-CNN, GWO, and GWO-MVO.

Table 4.2: Metaheuristic parameters for CNN hyperparameter optimization.

Method	#iterations	#wolves	p_1	p_2
GWO-CNN	10	20	-	-
GWO-MVO-CNN	10	20	2	10
qGWO-CNN	10	20	2	9

4.3.3 Experimental results

Ten independent optimization runs were performed to compare qGWO-CNN against GWO-CNN and GWO-MVO-CNN. The results in Table 4.3 show that qGWO-CNN significantly outperformed the other two methods. Using AlexNet on CIFAR-10, qGWO-CNN achieved the best test accuracy of 84.71%, 4.89%, and 4.6% higher than GWO-CNN and GWO-MVO-CNN, respectively.

Table 4.3: Experimental results of CNN hyperparameter optimization.

Approach	Average accuracy	Best accuracy	Worst accuracy	SD
GWO-CNN	79.58%	79.82%	79.11%	0.21%
GWO-MVO-CNN	80.11%	80.3%	79.74%	0.19%
qGWO-CNN	**84.18%**	**84.71%**	**84.01%**	**0.18%**

The box plot in Figure 4.2 further illustrates the distribution of test accuracies over the ten runs. qGWO-CNN consistently achieved the highest accuracies, while GWO-CNN performed the worst. This demonstrates that qGWO-CNN was able to find better CNN hyperparameters than the other optimization algorithms. The quantum-behaved mechanisms in qGWO likely contributed to its superior performance.

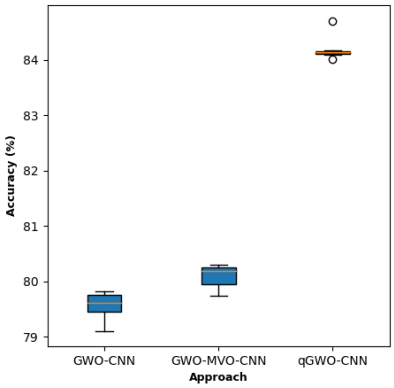


Figure 4.2: Boxplots of the test accuracies achieved on CIFAR-10 when optimizing the CNN hyperparameter.

4.3.4 Comparison with previous works

Table 4.4 compares the average test accuracy on CIFAR-10 achieved by qGWO-CNN against other hyperparameter optimization methods, including standard AlexNet [126],

PSO-CNN [126], PSO-b [109], MPSO-CNN [108], and cPSO-CNN [121]. The results show that qGWO-CNN improves substantially over standard AlexNet by 6.43%. It also outperforms PSO-CNN and PSO-b, which apply PSO for CNN hyperparameter and architecture optimization. Although MPSO-CNN and cPSO-CNN achieve higher accuracy, qGWO-CNN remains competitive since it optimized only the first convolutional layer here.

Table 4.4: Performance comparison in terms of average accuracy between qGWO-CNN and different approaches on CIFAR-10.

Approach	Network	Optimization	Accuracy
Standar AlexNet [126]	AlexNet	-	77.75%
PSO-CNN [126]	AlexNet	Hyperparameter optimization	80.15%
PSO-b [109]	13-CNN	Architecture optimization	81.47%
MPSO-CNN [108]	DCNN	Architecture optimization	87.34%
cPSO-CNN [121]	AlexNet	Hyperparameter optimization	**91.33%**
qGWO-CNN	AlexNet	Hyperparameter optimization	84.18%

4.4 Discussion and analysis

The proposed qGWO-CNN approach shows promising results for hyperparameter optimization of CNNs, significantly outperforming standard GWO and GWO-MVO in terms of test accuracy on CIFAR-10 using AlexNet. The quantum-behaved mechanisms in qGWO likely contribute to its superior optimization performance compared to standard GWO. qGWO-CNN also achieves better results than other optimization techniques like standard PSO, demonstrating its competitiveness. With optimization focused only on the first conv layer, qGWO-CNN attains test accuracy comparable to state-of-the-art methods that optimize all layers, indicating room for further improvement.

Overall, the results highlight the potential of qGWO for automated hyperparameter tuning of CNNs to boost their performance on image classification tasks. However, further analysis could discuss computational efficiency, convergence behavior, and scalability to larger CNNs and datasets. Thus, future work may extend qGWO-CNN to optimize the full CNN architecture and assess it on larger benchmark datasets.

4.5 Chapter summary

This chapter proposed a novel approach called qGWO-CNN that uses a quantum-inspired variant of GWO to efficiently search the high-dimensional space of possible

hyperparameters and find optimal configurations for CNNs. Experiments on the CIFAR-10 image classification benchmark demonstrate that qGWO-CNN achieves superior accuracy versus other GWO-based methods when optimizing hyperparameters of the AlexNet CNN architecture. Comparisons show the promise of the quantum-inspired GWO algorithm for automated hyperparameter optimization to boost CNN performance. Further analysis is still needed to gauge the capabilities and limitations of the qGWO-CNN approach fully.

Chapter 5

Combining SA with LR for Binary COPs

5.1 Introduction

In the previous two chapters, we focused on optimizing ML models using metaheuristics, specifically on optimizing hyperparameters. In this chapter, we move on to the second part of this thesis, which consists of improving metaheuristics with ML models. We aim to leverage ML models to enhance the optimization process of metaheuristic algorithms.

As discussed in Section 2.3, combining the strengths of ML models and metaheuristic algorithms can lead to the development of more efficient and effective optimization techniques. In this chapter, we explore the integration of a logistic regression model in initializing the simulated annealing metaheuristic algorithm, providing an informed initialization approach.

5.2 Motivation

Binary combinatorial optimization problems represent a class of challenges where the decision variables are restricted to binary values (0 or 1). These problems are formulated as combinatorial optimization tasks, which involve finding the best solution from a set of feasible options. Solving binary combinatorial optimization problems can be computationally demanding because the search space of possible solutions is often vast, and the problem may be NP-hard, meaning no known efficient algorithm can solve it in polynomial time (Section 1.2.2).

ML techniques have recently gained increasing attention for their potential in designing efficient algorithms to tackle combinatorial optimization problems. Integrating ML models has introduced greater intelligence into the search process, improving solution quality. By leveraging the capabilities of ML, these algorithms can effectively navigate complex solution spaces and identify high-quality solutions to combinatorial optimization challenges (Section 2.3).

Inspired by previous research [102] where regression models were used to initialize GA for solving the MKP, this study aims to explore the effectiveness of LR in tackling binary combinatorial optimization problems and integrate it with the SA metaheuristic algorithm to enhance its performance. Although SA is widely used, its random initialization may require numerous iterations to obtain an optimal solution. Since the algorithm relies on random perturbations to explore the solution space, there is no guarantee that it will converge to a good solution within a reasonable time. To address this issue, we propose initializing SA using the LR model.

5.3 Proposed methods

Our approach involves training a LR model using small instances and then utilizing it to build solutions for larger samples. Subsequently, we employ the SA algorithm to refine the solutions obtained from the LR model further. This combined approach leverages the strengths of both techniques, aiming to provide high-quality solutions efficiently. We apply these methods to two binary COPs: the MKP and the SSP.

5.3.1 LR for solving binary COPs

LR is employed to solve binary combinatorial optimization problems by using it to estimate the probability of each element being part of the optimal solution. To apply LR in this context, we require a dataset consisting of labeled input features and corresponding binary output labels. This dataset is then used to train the LR model, enabling it to learn the relationship between the input features and the binary output labels, representing the elements' inclusion or exclusion in the optimal solution.

5.3.1.1 Input features

To derive the input features for the logistic regression model, we draw inspiration from the nature of the combinatorial optimization problem at hand. In the case of

the MKP and the SSP, we take cues from greedy heuristic algorithms to select these features. Drawing inspiration from greedy heuristic algorithms and incorporating problem-specific knowledge, we can derive meaningful input features that capture the essential characteristics of the combinatorial optimization problems. This enables the logistic regression model to effectively learn and estimate the probability of each element's inclusion in the optimal solution.

- **MKP input features**: We use a single input feature that considers the weight of each item relative to the dimensions of the knapsack and the corresponding constraints, as well as the value or profit associated with each item. This feature is derived from three efficiency measures used in MKP [99]:

$$e_j^{scaled} = \frac{p_j}{\sum_{i=1}^{m} \frac{w_{i,j}}{c_i}} \tag{5.1}$$

$$e_j^{st} = \frac{p_j}{\sum_{i=1}^{m} w_{i,j}(\sum_{l=1}^{n} w_{i,l} - c_i)} \tag{5.2}$$

$$e_j^{fp} = \frac{p_j}{\sum_{i=1}^{m} r_i w_{i,j}} \tag{5.3}$$

$$r_i = \frac{\sum_{j=1}^{n} w_{i,j} - c_i}{\sum_{j=1}^{n} w_{i,j}} \tag{5.4}$$

  where p_j is the profit value of item j, $w_{i,j}$ is the weight of item j in dimension i, c_i is the capacity constraint for dimension i.
  We found through testing that it is not necessary to normalize these formulas. For a more detailed explanation of the efficiency measures used for the MKP, the reader is referred to [55].

- **SSP input features**: We utilize two input features for each element. The first feature is the gap between the element value and the target T. The second feature is the element density, calculated as the ratio of the element's value to the target sum T.

5.3.1.2 Labels

For both studied problems, we select small instances for which the optimal solution

is known or can be easily obtained using exact algorithms. Each optimal solution is represented as a binary vector, where a value of 1 indicates that the corresponding element is selected, and a value of 0 means that the element is not selected. This binary vector serves as the label for training the LR model.

By using small instances with known optimal solutions, we can generate labeled data for training the model. The input features derived from the problem-specific characteristics are paired with the binary vector representing the optimal solution.

5.3.1.3 Predict solution

After training the LR model, a feasible solution for an instance can be generated using the following naive method:

- Feed the model with the corresponding input feature vector of the instance elements.
- Obtain the output probability vector and sort the elements in descending order according to their probabilities.
- Initialize an empty solution set S.
- Add the elements to the solution set S one by one while ensuring that the solution remains feasible.

5.3.2 Combining SA with LR

While the LR model provides probabilities that estimate the likelihood of each element being part of the optimal solution, it lacks a comprehensive instance description, making it challenging to find the optimal solution directly. However, the item probabilities obtained from the LR model can be leveraged to support optimization algorithms.

On the other hand, SA is a widely used metaheuristic algorithm for solving COPs. However, its random initialization can slow convergence to the optimal solution. To address this limitation, we propose using the probabilities provided by the LR model to construct an initial solution that serves as a starting point for the SA algorithm.

5.3.2.1 Solution representation and initialization

The set of all elements in the problem instance is represented as a binary vector, where each bit of the vector corresponds to an element and can take on the value of either 0

or 1. A value of 1 indicates that the corresponding element is selected, while 0 denotes that the element is not selected.

The initial solution for the SA algorithm on a given instance is the solution generated by the LR model for that instance. If an element is present in the solution obtained from the LR model, its corresponding bit in the binary vector is assigned the value 1; otherwise, it is given the value 0.

5.3.2.2 Fitness Functions

The fitness function is formulated based on the specific characteristics of the COP being addressed. In the case of the MKP, the fitness function is calculated as the sum of the profits or benefits associated with the selected elements. Conversely, for the SSP, the fitness function is determined by summing up the values of the selected elements. In both problems, the objective is to maximize the fitness function, representing the solution's overall value.

5.3.2.3 Repair procedure

SA explores the solution space by randomly perturbing the current solution, as outlined in Section 1.3.2.1. However, such perturbations can occasionally yield infeasible solutions. To avoid constraint violations that could result from adding a random element to the solution, a repair procedure becomes necessary:

- **For the MKP**: The repair procedure involves removing the item with the smallest benefit value that restores feasibility.
- **For the SSP**: The repair procedure entails removing the item with the highest value.

5.3.2.4 The overall algorithm

The SA algorithm leverages the feasible solution obtained from the LR model as a starting point, guiding its search toward promising areas of the solution space. The algorithm begins with this initial solution and iteratively explores the solution space by randomly perturbing the solution by adding or removing an element. To maintain solution feasibility, a repair procedure is performed after each perturbation. The new solution is then evaluated and accepted under certain conditions. This process

is repeated over a specified number of iterations, which is determined empirically. As the algorithm progresses, the probability of accepting a new solution gradually decreases. This controlled exploration allows the algorithm to escape local optima while converging towards high-quality solutions. Algorithm 7 summarizes the steps of the LR+SA method.

Algorithm 7 The SA+LR method for Binary Combinatorial Optimization Problems.

Input: max_iter : max iterations, T: initial temperature, α: the cooling rate, I : problem instance, LR_model: LR model, f: fitness function

Output: the best feasible solution found S^*

Generate the solution S of the instance I using the LR_model
$S^* \leftarrow S$
$i \leftarrow 0$
while $i < max\_iter$ **do**
    Select a neighbor $S' \in N(S)$
    if S' not feasible solution **then**
        Repair S'
    end if
    if $f(S') > f(S)$ **then**
        $S \leftarrow S'$
    else
        $S \leftarrow S'$ with probability $\exp^{(f(S')-f(S))/T}$
    end if
    if $f(S') > f(S^*)$ **then**
        $S^* \leftarrow S'$
    end if
    Update T using Equation 1.6
    $i \leftarrow i+1$
end while
return S^*

5.3.2.5 Time complexity

The time complexity of our LR+SA approach is summarized as follows:

- In the initialization phase, LR+SA requires $O(N)$ time, where N represents the dimension of a candidate solution.
- The initialization of variables and parameters is of complexity $O(1)$.
- Generating a new solution is $O(1)$ in time complexity since the neighborhood search strategy reverses a randomly chosen bit.
- Evaluating each agent's fitness value requires $O(f(S))$ time, which is the time complexity of evaluating the objective function.

- Updating the different variables and parameters is a constant-time operation that requires $O(1)$ time.

For the two COPs studied, the fitness function's time complexity is O(N). Therefore, the total time complexity of the LR+SA approach is $O(max\_iter \times N)$, where max_iter denotes the maximum number of iterations.

5.4 Experiments

All our methods are implemented in Python, utilizing the following libraries: scikit-learn, NumPy, ortools, and pandas. We note that all the tests were executed on a desktop computer equipped with an AMD RYZEN 7 3700X CPU and 16 GB of RAM.

5.4.1 Benchmarks

The LR model is trained in a supervised manner, which requires instances with known optimal solutions. To facilitate this, small-sized benchmarks are chosen for training purposes. However, larger-sized benchmarks are utilized to evaluate and compare the performance of the methods studied. Table 5.1 lists the datasets employed for each optimization problem studied in this work.

Table 5.1: MKP and SSP benchmarks.

Problem	MKP	SSP
Training instances	SAC-94 dataset, based on a variety of real-world problems[1] (54 instances)	OR-library Subset Sum instances[2] (7 instances)
Evaluation instances	OR-Library [25] dataset, presented by Chu and Beasley (270 instances)	Random generation [71] a_i uniformly random in $[1, 10^3]$ and $T = n\frac{10^3}{4}$ where n in $4, 8, 12, \ldots 100$ (25 instances)

[1] https://www.cs.cmu.edu/Groups/AI/areas/genetic/ga/test/sac/0.html
[2] https://people.sc.fsu.edu/~jburkardt/datasets/subset_sum/subset_sum.html

5.4.2 Parameter settings

An experimental study was conducted to determine the optimal parameter setting for the SA algorithm. The initial temperature and the cooling rate (α) parameters were tuned through this study. Values of 250, 500, and 1000 were tested for the initial temperature. For α, values of 0.75, 0.85, and 0.95 were evaluated. The optimal values of these parameters for the MKP and the SSP are presented in Table 5.2.

Table 5.2: SA parameter configuration.

Parameter	T	α	#iterations
MKP	500	0.75	1000
SSP	1000	0.95	1000

5.4.3 Experimental results

The experiments were conducted ten times for every problem instance and technique to accommodate the stochastic characteristics inherent in SA-based methods.

5.4.3.1 Numerical results

To conduct a comprehensive evaluation and comparative study, the GAP metric is employed to assess the quality of the obtained solutions. The GAP is a commonly used metric to measure the performance of algorithms solving the MKP instances. It refers to the relative difference between the value of the best-known solution and the value of the solution found by the optimization algorithm, expressed as a percentage, as shown in Equation 5.5.

$$GAP = \frac{best\_known\_solution - solution\_found}{best\_known\_solution} \times 100\% \tag{5.5}$$

The basic SA algorithm is used as a baseline for comparison. The average GAP values are reported in the results, with the best results highlighted in bold.

a **MKP results**

The performance evaluation of SA, LR, and LR+SA methods is conducted on 270 test instances from the OR library for the MKP. These instances vary in constraints (5, 10, and 30) and item numbers (100, 250, and 500). Table 5.3 presents the average GAP

Table 5.3: Average GAP for MKP.

m	n	SA	LR			LR+SA			GPR [102]
			scaled	st	fp	scaled	st	fp	
5	100	4.741	3.557	3.806	3.586	1.59	1.562	**1.518**	4.34
	250	3.89	2.462	2.598	2.776	1.159	**1.11**	1.162	3.14
	500	4.95	1.995	1.883	1.879	**0.769**	0.785	0.821	2.54
10	100	4.342	4.464	4.613	4.518	2.547	2.4	**2.377**	5.01
	250	3.575	3.408	3.473	3.506	1.688	**1.558**	1.632	3.98
	500	4.4	2.276	2.352	2.413	**1.032**	1.122	1.096	2.82
30	100	3.458	5.376	5.015	5.397	3.189	2.967	**2.928**	5.37
	250	2.958	3.954	3.844	3.987	2.306	2.396	**2.286**	4.45
	500	3.953	3.03	3.126	3.063	1.74	1.824	**1.631**	3.71
average		4.03	3.391	3.412	3.458	1.78	1.747	**1.717**	3.93

for each of the nine data groups. The outcomes for LR-based techniques are delineated for each of the three specified input features highlighted in Section 5.3.1.

Remarkably, the LR+SA approach demonstrates superior performance across all input features. Notably, the fp feature showcases slightly enhanced performance compared to others. Furthermore, the results indicate that employing the LR model's naive solution construction method can outmatch the basic SA algorithm, particularly in scenarios with limited constraints or many items (i.e., 500). This underscores the LR model's capability to predict high-quality solutions even with a single feature.

When comparing the LR model's performance to the GPR model, which was found to be the most effective regression model in a previous study [102], the classification model (LR) demonstrates superior performance over regression models.

b **SSP results**

The results of the SA, LR, and LR+SA methods on the SSP test instances are presented in Table 5.4. Similar to the findings observed in the MKP analysis, the LR+SA method exhibits superior performance compared to other approaches. Furthermore, the results indicate that the LR model can predict solutions as well as those obtained by the LR+SA model, a feat not replicated by the basic SA algorithm. These observations imply that the LR model can predict high-quality solutions for the SSP, and when combined with the SA metaheuristic in the LR+SA approach, it can further enhance the solution quality, outperforming both the standard SA and LR methods.

Instance	SA	LR	LR+SA
p4	20.05	**18.3**	**18.3**
p8	1.55	4.8	**1.31**
p12	1.156	2.333	**1.11**
p16	0.67	1	**0.09**
p20	0.252	0.16	**0.088**
p24	0.215	0.167	**0.113**
p28	0.58	1.143	**0.526**
p32	0.27	0.238	**0.191**
p36	0.192	0.122	**0.078**
p40	0.133	0.03	**0.014**
p44	0.115	0.155	**0.1**
p48	0.131	0.1	**0.049**
p52	0.065	**0**	**0**
p56	0.107	0.114	**0.037**
p60	0.093	0.04	**0.027**
p64	0.091	0.025	**0.023**
p68	0.072	0.053	**0.032**
p72	0.077	0.022	**0.019**
p76	0.08	0.042	**0.025**
p80	0.067	0.025	**0.02**
p84	0.053	0.029	**0.023**
p88	0.04	0.036	**0.014**
p92	0.072	0.009	**0.007**
p96	0.042	0.029	**0.015**
p100	0.056	0.008	**0.006**

Table 5.4: Average GAP for SSP.

5.4.3.2 Statistical results

To evaluate the impact of our enhanced approach on the SA algorithm, we conducted a non-parametric Wilcoxon Rank-Sum test on ten runs with a significance level of $\alpha = 0.05$. The p-values resulting from comparing the SA+LR method (using the fp feature) with the baseline SA algorithm are displayed in Table 5.5. Following the criterion where an algorithm is deemed significantly superior if the p-value is below 0.05, it can be confidently affirmed that our SA+LR approach notably surpasses the original SA algorithm on both binary COPs.

By incorporating the LR model's outputs as an informed initialization for the SA algorithm, our proposed SA+LR method demonstrates a statistically significant improvement over the traditional SA algorithm. The p-values obtained from the

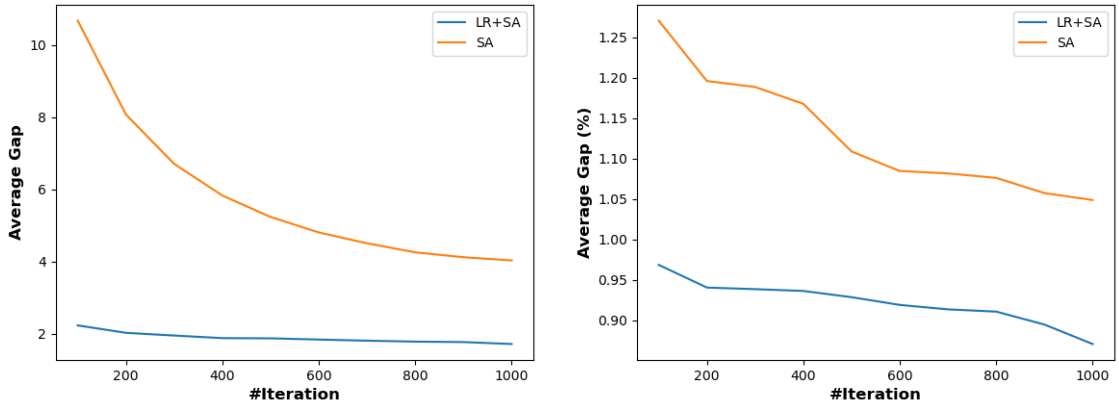
Table 5.5: p-value of Wilcoxon's rank-sum test.

Dataset	MKP	SSP
p-value	1.11e−9	0.04

Wilcoxon Rank-Sum test provide strong evidence that integrating of the LR model's predictions with the SA metaheuristic leads to superior performance in solving these COPs.

5.4.3.3 Impact of SA initialization on convergence speed

Figure 5.1 illustrates the impact of the LR initialization on the convergence of the SA algorithm for the MKP and the SSP. The average GAP results are reported after every 100 iterations for two initialization strategies: random initialization and initialization using the LR model predictions (using the fp feature).



(a) MKP **(b)** SSP

Figure 5.1: Comparing the GAP convergence of SA and LR+SA.

For both the MKP and SSP, the convergence plots demonstrate that the SA algorithm initialized with the LR model (SA+LR) outperforms the randomly initialized SA algorithm. Notably, the average GAP obtained by SA+LR after only 100 iterations is significantly lower than the average GAP achieved by the randomly initialized SA after 1000 iterations. This highlights the significant time-saving benefits of initializing SA with the LR model.

Integrating the LR model's predictions in the initialization step enhances the convergence speed of the basic SA method, allowing for faster and more efficient search

processes. By starting from a high-quality initial solution provided by the LR model, the SA+LR approach can explore the solution space more effectively and converge towards optimal or near-optimal solutions in fewer iterations than the randomly initialized SA algorithm.

5.5 Discussion and analysis

This study demonstrates the effectiveness of integrating ML techniques, specifically LR, with the SA metaheuristic algorithm for solving binary COPs. The proposed LR+SA approach outperforms the conventional SA and LR methods across various MKP and SSP instances.

The LR model, trained on small instances with known optimal solutions, effectively learns to predict the probability of each element being part of the optimal solution. By using these probabilities to construct an initial solution, the SA algorithm can be initialized with a high-quality starting point, leading to faster convergence and improved solution quality compared to random initialization. The proposed LR+SA approach combines the strengths of both the LR model and the SA metaheuristic, benefiting from the LR model's ability to provide an informed initialization and the SA algorithm's capacity for controlled exploration and refinement of the solution while avoiding premature convergence to local optima.

While the results are promising, they are not at the state-of-the-art level. Besides, it is vital to acknowledge the potential limitation of initializing metaheuristic algorithms with a single solution, even if it is a good-quality solution. Initialization based on just one solution can lead to convergence towards a local optimum, hindering the algorithm's ability to explore the solution space thoroughly and potentially missing the global optimum.

To address this limitation, further investigations and tests are recommended. One potential approach could be to apply the LR model's predictions as an initialization strategy for population-based metaheuristics. By initializing a diverse population of solutions, where some solutions are based on the LR model's outputs and others are randomly initialized, these algorithms could maintain a diversified exploration of the solution space while benefiting from the informed initialization provided by the LR model.

5.6 Chapter summary

This chapter demonstrated the promising potential of integrating ML techniques into metaheuristic algorithms during the initialization phase to solve binary COPs more effectively.

We proposed combining the LR model with the SA metaheuristic to tackle binary COPs. By using the LR output probabilities to construct an initial high-quality solution, the SA algorithm could initialize its search from a better starting point, leading to faster convergence and improved solution quality than random initialization. The proposed LR+SA approach outperformed both conventional SA and standalone LR across various WDP and SSP instances.

However, one possible drawback involves starting with a singular solution, which could lead to premature convergence. A recommended approach is to initialize population-based metaheuristics with a diverse set of solutions, some informed by the machine learning model and others randomly initialized, to maintain exploration while benefiting from informed initialization.

Chapter 6

Intelligent Discrete GWO for WDP

6.1 Introduction

In the previous chapter, we explored the integration of ML into metaheuristic algorithms during the initialization phase. In this chapter, we shift our focus to applying ML methods, specifically frequent itemset extraction techniques, during the evolution of the search process to enhance the algorithm's performance.

As demonstrated in [132], incorporating frequent itemset extraction techniques during the search process can augment the exploration and exploitation capabilities of metaheuristic algorithms, ultimately leading to improved solution quality and computational efficiency in solving complex optimization problems.

Our contribution presents a novel Discrete Grey Wolf Optimizer (DGWO) algorithm for tackling the Winner Determination Problem (WDP). The search and attack operators of the algorithm are redefined to enhance coding efficiency. Two strategies are proposed to solve WDP: 1) The newly developed DGWO algorithm and 2) An enhanced version of DGWO incorporating a learning mechanism during the search process by leveraging frequent itemset extraction techniques from ML.

6.2 Motivation

Data mining is the process of extracting novel and potentially valuable knowledge from datasets through patterns and rules. This new-found knowledge, obtained through ML techniques, has proven to be highly effective and is increasingly being leveraged to guide the search for better solutions in metaheuristic optimization procedures (Section 2.3).

Integrating ML models into the neighbor generation process is a promising approach that utilizes prior knowledge to steer the search process more effectively. Most existing literature focuses on identifying and exploiting common features present in good solutions found during the search process. However, while these patterns derived from a set of optimal solutions can indeed accelerate convergence towards high-quality solutions, they may also inadvertently limit the diversity of solutions explored by the metaheuristic, leading to premature stagnation at local optima (Section 2.3.4.3).

The GWO algorithm, a popular metaheuristic inspired by the hunting behavior of grey wolves, guides its search process primarily based on the three best solutions found so far (referred to as the leaders). This inherent design emphasizes exploitation based on patterns present in high-quality solutions rather than exploration of the search space. Consequently, we believe that exploring rare patterns, which may not necessarily be present in the current set of high-quality solutions, could be a more interesting and potentially fruitful approach to improving solution diversity and avoiding stagnation at local optima.

Our research takes the form of a case study investigating the WDP, a well-studied and challenging COP that has wide applications in various domains, including operations research and computer science. By applying our proposed approach to this complex problem, we aim to demonstrate the potential benefits of integrating rare pattern exploration into metaheuristic search processes.

6.3 Proposed approach

6.3.1 DGWO algorithm for the WDP

The GWO algorithm was originally designed to solve continuous function optimization problems. However, the WDP falls under the category of discrete COPs. As a result, several issues need to be addressed to adapt and implement the GWO algorithm effectively to tackle the WDP. This transition requires careful consideration and adaptation of various components of the GWO algorithm. These modifications are necessary to ensure that the algorithm can effectively explore the discrete solution space, handle constraints, and accurately optimize the objective function specific to the WDP.

6.3.1.1 Conflict graph

To ensure the feasibility of allocations during the search process, we have employed a conflict graph, which is a helpful tool. In the conflict graph, the vertices represent individual bids, and edges connect pairs of bids that cannot be accepted together in the same allocation due to conflicting item bundles. Essentially, this graph provides a visual representation of bid conflicts, enabling us to quickly detect and prevent the inclusion of mutually exclusive bids in the same allocation, thereby ensuring the generation of feasible solutions throughout the search process.

6.3.1.2 Solution representation

For the WDP, a solution represents a set of winning bids. The feasible solution is represented as a combination of bids that satisfies the non-conflict constraint. We utilize a vector (V) of integers with a variable length limited by the number of bids, where each component $V[i]$ corresponds to the number of a winning bid.

6.3.1.3 Population initialization

To generate a feasible solution for the WDP, we employ the random key encoding mechanism [14], which operates as follows: we generate n real numbers sequenced by a random order r, where n is the number of bids, and r is a permutation of key values. The values in r essentially act as priorities for the bids. This mechanism ensures that all bids have a chance of being accepted based on their randomly assigned priorities (order values).

To construct an allocation (solution), we first select the bid with the highest order value in r and include it in the allocation. Next, we consider the bid with the second-highest order value and accept it into the allocation if it does not conflict with any previously accepted bids. If it conflicts, we discard it. This process is repeated until all n bids have been examined. The subset of accepted, non-conflicting bids forms a feasible solution to the WDP.

Example 1. Consider an auction of four items 1, 2, 3, 4 with five bids $\{B1, B2, B3, B4, B5\}$. Each bid $Bi = (P_i, G_i)$ specifies the bundle of items G_i for which the bidder is prepared to pay the price P_i. The five bids are:

- B1=(120.25, 1, 2, 4)

- B2=(300, 1, 2, 3)
- B3=(200.10, 3, 4)
- B4=(200, 2, 3)
- B5=(100.10, 3)

To generate a feasible solution, we follow these steps:

- Since there are five bids, we generate five random numbers sequenced by an order r. Let us say $r = \{0.85, 0.65, 0.60, 0.38, 0.70\}$.
- The bid with the highest order value is $B1(0.85)$, so we include its number in the allocation: $V = \{1\}$.
- The bid with the second-highest order value is $B5(0.70)$. Since $B5$ does not conflict with $B1$, we add it to the allocation: $V = \{1, 5\}$.
- The bid with the third-highest order value is $B2(0.65)$. However, $B2$ conflicts with $B1$ (they share items 1 and 2), so we discard $B2$.
- The bids with the lowest order values are $B3(0.60)$ and $B4(0.38)$. Both are discarded as they conflict with bids already in the allocation.

We obtain the allocation $V = \{1, 5\}$ as one feasible solution for the WDP.

6.3.1.4 Solution evaluation

The objective function F measures the quality of solutions in the WDP; It represents the overall revenue obtained from the winning bids in allocation V. Therefore, the goal is to maximize this objective function.

$$F(V) = \sum_{i=1}^{L} Price(Bi) = \sum_{i=1}^{L} Pi \tag{6.1}$$

where L is the number of winning bids in the allocation V. $Price(Bi)$, or Pi is the price of the i^{th} winning bid Bi in the allocation.

For Example 1, the objective function value or overall revenue is the sum of the prices of the winning bids $B1$ and $B5$: F(V) = Price(B1) + Price(B2) = 120.25 + 100.10 = 220.35

6.3.1.5 DGWO algorithm

In the proposed DGWO algorithm for the WDP, the search process is guided by three

leaders, similar to the original GWO algorithm. These leaders are referred to as α, β, and δ. The DGWO method begins with an initial population of solutions generated using the random key encoding technique. At each iteration, the algorithm performs either an intensification or a diversification procedure with a certain probability (p).

The intensification procedure focuses on exploiting the promising regions around the current leaders by generating new solutions in their vicinity. This phase aims to intensify the search and potentially refine the best solutions found so far. On the other hand, the diversification procedure promotes exploration by generating new diversified solutions from the current leaders. This phase helps the algorithm escape local optima and explore different search space regions, maintaining population diversity.

The choice between intensification and diversification is made probabilistically based on the value of p. A higher value of p favors intensification, while a lower value promotes diversification. By balancing these two phases, the DGWO algorithm aims to achieve an effective trade-off between exploitation and exploration, allowing it to converge toward high-quality solutions while maintaining diversity in the search process.

a **Intensification**

The intensification step is a crucial phase that involves exploring the neighborhoods around the three leading solutions. During this step, to update the position X of an individual, the algorithm first calculates the Hamming distances between the current individual's solution and the solutions of the three leaders. The Hamming distance represents the number of bids that differ between two solutions, providing insights into their similarity. For example, the distance between the current solution $V1 = \{5,1,6,11,9\}$ and $V2 = \{3,2,5,10,6\}$ is three.

For each leader, a random number m_i is generated within the range $[0, d_i[$, where d_i is the Hamming distance from that leader ($i = 1, 2$, or 3). This random number determines the number of new bids that will be introduced into the current solution from the respective leader's solution. The algorithm then randomly selects m_i bids that are present in the leader's solution but not in the current solution X. These new bids are added to X. In contrast, any conflicting bids (that share items with the new bids) are removed from X, as sketched in Algorithm 8.

This process is repeated for each leader, resulting in three new candidate solutions: $X1$, $X2$, and $X3$. After generating these candidates, the algorithm selects the best solution (i.e., the one with the highest fitness value) among $X1$, $X2$, and $X3$ as the

Algorithm 8 Move to leader

Input: *c_graph*: conflict graph, X: current position, L: leader position
Output: X

1: $d =$ the distance between the current position and the leader position
2: $r_d =$ find a random number between $[0, d[$
3: select r_d random bids (B) form L that do not exist in X
4: add B to X
5: remove all conflicting bids from X except B
6: **return** X

updated position for the current individual. The complete intensification process is outlined in Algorithm 9.

Algorithm 9 Intensification

Input: *c_graph* : conflict graph, X: current position, X_α, X_β, X_δ, F: fitness function
Output: X

1: $X_1 \leftarrow$ move to X_α (Algorithm 8)
2: $X_2 \leftarrow$ move to X_β (Algorithm 8)
3: $X_3 \leftarrow$ move to X_δ (Algorithm 8)
4: Evaluate X_1, X_2, X_3
5: Update X by the best of X_1, X_2 and X_3
6: **return** X

The intensification step aims to exploit the promising regions around the three leaders by introducing new bids from their solutions into the current solution while maintaining feasibility by resolving conflicts. This approach helps to refine and improve the current solution by incorporating elements from the leading solutions, ultimately guiding the search towards high-quality regions of the search space.

b **Diversification**

In the diversification step, a new solution is randomly generated using the same random key encoding mechanism employed to initialize the population. A crossover operator with this newly generated solution is then applied. The diversification procedure is detailed in Algorithm 10. The new random solution serves as a source of diversity, introducing potentially new and unexplored combinations of bids into the population.

The overall DGWO algorithm for the WDP, integrating both the intensification and diversification steps, is summarized in Algorithm 11. The core search process is repeated iteratively until a specific termination criterion is reached. We choose the

Algorithm 10 Diversification

Input: *c_graph* : conflict graph, X: current position
Output: P

```
Generate a random solution P
for each bid in X do
    if there is no conflict then
        add bid to P
    end if
end for
return P
```

criterion for termination depending on the number of iterations in which the top three solutions (α, β, and δ) remain unchanged. If the leading solutions remain unchanged for a predetermined number of iterations (*max_no_change*), the algorithm terminates, indicating convergence or stagnation in the search process.

The Time complexity of the DGWO approach for solving the WDP can be computed as follows:

- Creating the conflict graph is a costly process since it requires checking the items in each bid against the items in all the other bids. Therefore, its time complexity is $O(b^2 \times g^2)$, where b is the number of bids and g is the maximum number of items in a bit.
- The following operations are performed to create a single-candidate solution:
  - Creating the vector V requires $O(b)$.
  - Creating a feasible solution using V requires verifying if there is no conflict with any previously accepted bids; in the worst case, it takes $O(b^2)$.

  Overall, creating the initial population requires $O(n \times b^2)$ operations, where n is the population size.
- Evaluating the population requires $O(n \times z)$, where z is the maximum number of bids in a feasible solution.
- The time complexity of the intensification phase is $O(n \times z^2)$.
- The time complexity of the diversification phase is $O(n \times z^2)$.

Thus, for each generation or iteration, the time complexity is $O(b^2 \times g^2 + n \times b^2 + n \times z + n \times z^2)$. If we assume a maximum number of iterations T, then the total time complexity is $O(b^2 \times g^2 + n \times b^2 + T \times (n \times z + n \times z^2))$.

Algorithm 11 DGWO for WDP

Input: I: instance, n: population size, F : fitness function, p, max_no_change
Output:X_α

Create the conflict graph
Initialize the population randomly
Evaluate the population
Select the three leaders (X_α , X_β, X_δ)
$no\_change \leftarrow 0$
while $no\_change < max\_no\_change$ **do**
    for each individual **do**
        $r =$ find a random number between $[0,1]$
        if $r < p$ **then**
            Intensification (Algorithm 9)
        else
            Diversification (Algorithm 10)
        end if
    end for
    Update the leaders (X_α , X_β, X_δ)
    if no change in leaders **then**
        $no\_change \leftarrow no\_change + 1$
    else
        $no\_change \leftarrow 0$
    end if
end while
return X_α

6.3.2 No-frequent pattern based DGWO

In contrast to previous work, we aim to incorporate patterns not commonly found in optimal solutions during the search phase. The proposed No-Frequent Pattern-based Discrete Grey Wolf Optimizer (NFP-DGWO) algorithm seeks to extract meaningful information and learn from the DGWO iterations. However, this requires striking a proper equilibrium between the DGWO search and the learning mechanism.

The NFP-DGWO algorithm follows the principle of applying the learning process to an elite solution set after each time interval. This elite set represents the m best solutions found so far from the beginning of the search process. Using the FP-max algorithm, the most frequent bids in the elite solutions are extracted to form a list. A new solution is then created based primarily on two components: the alpha position (X_α), which represents the current best solution, and non-frequent bids that are not commonly found in elite solutions. Including non-frequent bids aims to introduce

diversity and explore unexplored regions of the search space. To generate this new solution, a local search procedure is implemented using the Variable Neighborhood Search (VNS) algorithm [86], as outlined in Algorithm 12. To maintain a fixed population size, the worst individual is removed each time this new solution is added.

Algorithm 12 NFP-DGWO Learning Process

Input: c_graph : conflict graph, F: fitness function, $no\_frequent$: non-frequent bids, X_α, k_{max}, t_{max}

Output: X_{new}

$t \leftarrow 0$
$X_{new} \leftarrow X_\alpha$
while $t < t_{max}$ **do**
    $k \leftarrow 0$
    while $k < k_{max}$ **do**
        $X_1 \leftarrow Shake(c\_graph, X_{new}, no\_frequent, k)$ (Algorithm 13)
        $X_2 \leftarrow First\ Improvement(c\_graph, X_1, no\_frequent, F)$ (Algorithm 14)
        if $F(X_2) > F(X_1)$ **then**
            $X_{new} \leftarrow X_2$
            $k \leftarrow 0$
        else
            $k \leftarrow k+1$
        end if
    end while
    $t \leftarrow t+1$
end while
return X_{new}

Algorithm 13 Shake

Input: c_graph : conflict graph, X: current solution, $no\_frequent$: non-frequent bids, k

Output: Xw

$i \leftarrow 0$
while $i < k$ **do**
    select a random bids (B) form $no\_frequent$ that do not exist in X
    add B to X
    remove all conflicting bids from X except B
    $i \leftarrow i+1$
end while
return X

Algorithm 14 First Improvement

Input: c_graph : conflict graph, X: current solution, $no\_frequent$: non-frequent bids, F

Output: X_{new}

1: **for** each bid B in $no\_frequent$ and not in X **do**
2: $X_{new} \leftarrow X$
3: add B to X_{new}
4: remove all conflicting bids from X_{new} except B
5: **if** $F(X_{new}) > F(X)$ **then**
6: **return** X_{new}
7: **end if**
8: **end for**
9: $X_{new} \leftarrow X$
10: **return** X_{new}

The proposed approach combines the strengths of the DGWO search process with a learning mechanism that exploits infrequent patterns. By incorporating infrequent bid patterns rarely present among the elite solutions, this hybrid algorithm promotes diversity within the search process. This can potentially lead to the discovery of better solutions. The overall NFP-DGWO search process for solving the WDP is sketched in Algorithm 15.

Concerning the time complexity, the overall time complexity of the NFP-DGWO algorithm is a combination of the time complexity of the DGWO operations and the following factors:

- The FP_itemsets discovery process complexity is $O(m \times z \times \log m + s \times 2^m)$, where m is the elite list size and s is the average size of frequent itemsets.
- The time complexity of the learning process is $O(K \times t_{max} \times (m^2 + b \times m))$, where K is the maximum number of neighborhoods considered, which can be exponential, and t_{max} is the maximum number of iterations in the learning process.

Combining all the factors, the overall time complexity of the NFP-DGWO algorithm for solving the WDP becomes $O(b^2 \times g^2 + n \times b^2 + T \times (n \times z + n \times z^2 + \frac{1}{LT}(m \times z \times \log m + s \times 2^m + K \times t_{max} \times (m^2 + b \times m))))$. Where LT represents the learning interval, which is the number of iterations DGWO runs before applying the learning procedure. We note that for DGWO and NFP-DGWO algorithms, the bids within each solution

Algorithm 15 NFP-DGWO for WDP

Input: I: instance, n: population size, m: elite size, F : fitness function, p, max_no_change, LT: learning interval, min_sup: minimum support, k_{max}, t_{max}

Output: X_α

Create the conflict graph c_graph
Initialize the population randomly
Evaluate the population
Select the three leaders (X_α , X_β, X_δ)
$no\_change \leftarrow 0$
$i \leftarrow 0$
select $elite\_list$
while $no\_change < max\_no\_change$ **do**
    for each individual **do**
        $r =$ find a random number between $[0,1]$
        if $r < p$ **then**
            Intensification (Algorithm 9)
        else
            Diversification (Algorithm 10)
        end if
    end for
    if $(i+1)$ **mod** $LT = 0$ **then**
        discover $FP\_itemsets$ with min_sup in $elite\_list$
        $no\_frequent \leftarrow$ bids that do not exist in $FP\_itemsets$
        $X_{worst} \leftarrow Learning\ Process(c\_graph, F, no\_frequent, X_\alpha, k_{max}, t_{max})$ (Algorithm 12)
    end if
    Update the leaders (X_α , X_β, X_δ)
    Update $elite\_list$
    if no change in leaders **then**
        $no\_change \leftarrow no\_change + 1$
    else
        $no\_change \leftarrow 0$
    end if
    $i \leftarrow i+1$
end while
return X_α

Table 6.1: WDP benchmarks.

Class	Instances	Selected instances
REL_1000_500	in101 to in200	in101 to in110
REL_1000_1000	in201 to in300	in201 to in210
REL_500_1000	in401 to in500	in401 to in410
REL_1500_1000	in501 to in600	in501 to in510
REL_1500_1500	in601 to in700	in601 to in610

are sorted to facilitate the use of the dichotomic search algorithm, which determines the presence or absence of a specific bid within a solution.

6.4 Experiments

The DGWO and NFP-DGWO algorithms were implemented in Python using the mlxtend, NumPy, and pandas libraries. Experiments were conducted on a machine with an Intel Core i5-12500 @3.0 GHz processor, 16 GB RAM, under Linux 22.04.

Extensive empirical studies were performed on well-known benchmarks to evaluate the algorithms' performance. This section describes the benchmarks, parameter tuning process, and experimental results. The proposed approaches are compared against state-of-the-art competing methods to demonstrate their effectiveness.

6.4.1 Benchmarks

The benchmarks used in this study are carefully selected from the LG benchmarks. These benchmarks are complex enough that even CPLEX, a state-of-the-art optimization solver, cannot find optimal solutions within reasonable time constraints. This benchmark set consists of 500 instances, each categorized into one of five classes based on the number of bids and items involved. These classes are named REL_X_Y, where X represents the number of bids, and Y denotes the number of items. Each class contains 100 examples. To ensure a comprehensive analysis and facilitate comparisons with earlier works, 50 samples were selected from these classes, as detailed in Table 6.1.

6.4.2 Parameter settings

To determine the optimal parameter settings for the proposed algorithms, we employed an experimental study utilizing the automatic tuning tool irace [77]. This tool facilitates

Table 6.2: Parameter setting of DGWO and NFP-DGWO.

Algorithm	n	m	p	max_no_change	LT	$minsup$	k_{max}	t_{max}
DGWO	150	-	0.1	800	-	-	-	-
NFP-DGWO	150	70	0.1	800	200	0.4	4	75

the systematic adjustment and tuning of algorithm parameters through an iterative racing procedure. The final parameter values obtained through this tuning process are presented in Table 6.2.

6.4.3 Experimental results

In order to conduct a thorough performance evaluation, we executed each algorithm ten times on every benchmark instance. The experimental results obtained for the DGWO and NFP-DGWO algorithms are presented in Tables 6.3–6.7. These tables report several key metrics, including the maximum fitness value (max), the minimum fitness value (min), the average fitness value across the runs (avg), and the average convergence time in seconds (time) for each algorithm on the respective set of benchmark instances.

Based on the experiment's results, the NFP-DGWO algorithm outperforms the DGWO algorithm across all benchmark instances in terms of objective function performance. The NFP-DGWO algorithm's superior performance can be attributed to its ability to effectively balance exploration and exploitation, thereby avoiding local optima and premature convergence. In essence, NFP-DGWO displayed a superior capacity to solve WDP instances compared to the DGWO algorithm, thanks to its innovative approach of exploring infrequent bid patterns to foster diversity and its ability to guide the search process adaptively. By strategically incorporating these rarely occurring patterns, NFP-DGWO achieved a higher degree of solution diversity, mitigating the risk of premature convergence and enabling a more thorough search space exploration.

Although the NFP-DGWO algorithm achieves superior solution quality compared to the DGWO algorithm, it is associated with longer computation times, mainly when dealing with larger instance sizes. The extended runtime is due to the computational nature of the learning process employed by NFP-DGWO.

Figure 6.1 compares the convergence behavior and performance of DGWO and NFP-DGWO when applied to the same problem instance across five independent runs. It illustrates the trade-off between convergence speed and final solution quality.

Although DGWO converges relatively slowly, it does not allow deeper exploration of the solution space, as the final fitness function values are of lower quality compared

Table 6.3: Results on subset of REL_1000_500.

Instance	DGWO				NFP-DGWO			
	max	min	avg	time	max	min	avg	time
101	64595.18	59982.4	62124.22	**26.26**	**69840.08**	**65725.66**	**66784.09**	95.12
102	65814.66	61996.59	63340.81	**29.22**	**69362.8**	**65703.02**	**67213.43**	89.07
103	67443.73	60045.54	61736.83	**20.57**	**69791.25**	**65722.87**	**67222.15**	73.55
104	64829.97	59625.29	61762,87	**21**	**69580.34**	**65169.28**	**67080.4**	80.54
105	65152.27	60388.22	62855.5	**25.07**	**70001.25**	**67026.14**	**68802.74**	92.2
106	62107	57409.93	59372.81	**27.5**1	**68204.65**	**63246.41**	**64814.26**	77.07
107	64839.45	59092.68	61336.61	**29.93**	**68587.17**	**65392.96**	**66808.21**	82.86
108	68557.91	64725.53	67015.32	**30.72**	**75147.18**	**70793.06**	**72825.77**	110
109	62790.61	57843.89	60395.89	**28.06**	**65822.84**	**61488.64**	**63233.29**	82.27
110	64497.5	58332.74	59960.34	**26.04**	**67395.07**	**62711.35**	**64105.6**	92.9

Table 6.4: Comparison between DGWO and NFP-DGWO on subset of REL_1000_1000.

Instance	DGWO				NFP-DGWO			
	max	min	avg	time	max	min	avg	time
201	79331.63	72515.75	74924.94	**20.81**	**81557.74**	**79853.62**	**80194.45**	61.98
202	83810.89	79175.17	81648.79	**21.86**	**90708.13**	**87627.42**	**88901.78**	76.62
203	80110.73	76365.99	78367.57	**18.75**	**86239.21**	**86239.21**	**86239.21**	68.42
204	84879.4	77274.07	79956.05	**20.31**	**87075.43**	**87071.48**	**87075.03**	73.62
205	84016.44	77192.9	79668.2	**21.68**	**86515.95**	**84016.44**	**84016.44**	70.49
206	84981.3	81304.24	84115.65	**21.41**	**89425.94**	**84981.3**	**85474.94**	50.1
207	85482.58	80792.92	83552.88	**16.46**	**93129.25**	**93115.57**	**93121.04**	82.77
208	83663.53	80053.95	82023	**20.61**	**91782.04**	**91782.04**	**91782.04**	69.82
209	86455.43	78807.33	82295.15	**23.78**	**86755.8**	**86455.43**	**86547.65**	67.92
210	86124.80	78689	82543.71	**23.71**	**88500.61**	**86124.8**	**86420.15**	68.6

Table 6.5: Comparison between DGWO and NFP-DGWO on subset of REL_500_1000.

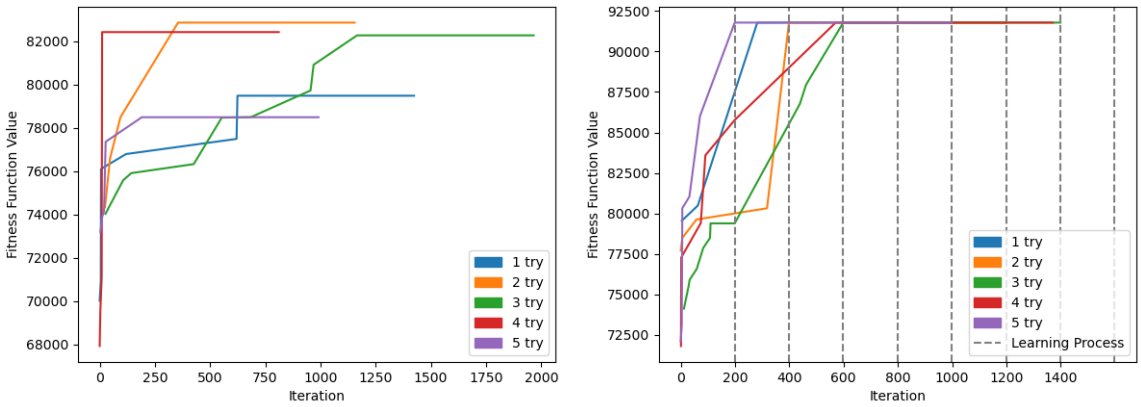
Instance	DGWO				NFP-DGWO			
	max	min	avg	time	max	min	avg	time
401	**77417.48**	69810.44	75762.89	**7.5**	**77417.48**	**77417.48**	**77417,482**	21.6
402	72134.25	70022.98	71647.24	**8.09**	**76273.34**	**74469.07**	**75912.48**	29.42
403	**74843.96**	66792.17	70693.42	**9.45**	**74843.96**	**74843.96**	**74843.96**	28
404	75528.10	71625.93	72546.94	**7.59**	**78761.69**	**78761.69**	**78761.69**	35
405	74984.49	72686.65	73084.22	**9.08**	**75915.9**	**74899.13**	**75291.7**	27.79
406	71898.26	69348.57	70526.14	**8.78**	**72863.32**	**72439.95**	**72651.64**	34.23
407	**76365.72**	70757.64	73121.07	**8.71**	**76365.72**	**73935.28**	**75844.38**	28.47
408	**77018.83**	70039.24	72725.48	**8.6**	**77018.83**	**77018.83**	**77018.83**	38.81
409	70021.81	65594.14	67391.43	**8.49**	**73188.62**	**73188.62**	**73188.62**	34.33
410	71791.58	68709.25	70368.13	**9.01**	**73791.66**	**72395.99**	**73260.19**	32.48

Table 6.6: Comparison between DGWO and NFP-DGWO on subset of REL_1500_1000.

Instance	DGWO				NFP-DGWO			
	max	min	avg	time	max	min	avg	time
501	85367.51	78001.81	81254.39	**31.14**	**87830.42**	**84524.04**	**85819.89**	101.88
502	79495.68	74144.57	76434.13	**32.21**	**86236.91**	**81294.88**	**82038.48**	103.69
503	83089.17	78636.83	80352.03	**30.03**	**86191.79**	**83277.71**	**84048.52**	94.92
504	83947.14	76900.05	79087.47	**28.16**	**85600**	**83947.14**	**84139.73**	97.47
505	80441.12	77836.4	78185.09	**20.83**	**84860.17**	**81148.83**	**83014.03**	83.96
506	82038.17	74419.45	77979.79	**34.66**	**83744.5**	**82038.17**	**82608.36**	107.09
507	85509.43	79333.8	82000.11	**34.16**	**90288.47**	**85103.89**	**87568.27**	106.8
508	80630.53	78642.53	79390.01	**31.14**	**86853.5**	**84033.39**	**85364.9**	112.32
509	82292.47	78497.01	80202.76	**36.31**	**86672.04**	**82672.26**	**84670.71**	115.59
510	83513.13	77448,82	79608.08	**26.31**	**88163.82**	**85599.33**	**86216.98**	104.36

Table 6.7: Comparison between DGWO and NFP-DGWO on subset of REL_1500_1500.

Instance	DGWO				NFP-DGWO			
	max	min	avg	time	max	min	avg	time
601	103948.76	94509.9	98936.07	**29.13**	**108800.45**	**104114.61**	**105438.26**	102.43
602	98695.83	92615.86	95589.07	**30.8**1	**105611.48**	**99401.46**	**101249.06**	86.78
603	94805.29	90459.1	92415.31	**25.3**	**102568.78**	**97043.33**	**98038.52**	99.5
604	103737.1	96149.23	101143.02	**29.51**	**106614.79**	**103737.1**	**105427.51**	92.08
605	103026.8	98667.58	100860.41	**30.67**	**107180.51**	**103026.8**	**103806.34**	113.45
606	**105218.21**	93615.93	99857.15	**35.49**	**105218.21**	**105218.21**	**105218.21**	101.75
607	98076.11	94936.09	96244.04	**29.22**	**105869,45**	**100990.55**	**102903.8**	115.86
608	96567.45	93547.25	95049.72	**32.61**	**105266.11**	**100012.06**	**101687.6**	132.79
609	101342.99	94653.71	98476.87	**26.3**	**109472.33**	**101342.99**	**104733.41**	84.41
610	104810.11	99700.22	101049.7	**25.99**	**111835.92**	**106453.56**	**108229,42**	110.51



(a) DGWO **(b)** NFP-DGWO

Figure 6.1: Comparing the convergence of DGWO and NFP-DGWO on the 208 instance.

to NFP-DGWO. On the other hand, the NFP-DGWO approach achieves higher final fitness function values, suggesting that adaptive learning mechanisms can potentially lead to better solutions with fewer iterations.

Moreover, the graphs highlight the differences in smoothness and run-to-run variability between the two algorithms. The convergence trajectories of NFP-DGWO are smoother and more gradual, likely due to the periodic learning phases that adjust the algorithm's behavior. Additionally, NFP-DGWO demonstrates less variability across the five runs, with the lines following a more consistent trajectory, potentially due to the learning mechanism's ability to exploit information from previous iterations and adapt accordingly.

6.4.4 Comparison with state-of-the-art algorithms

To assess the effectiveness of the proposed NFP-DGWO algorithm, we conducted a comprehensive analysis, comparing it to several state-of-the-art approaches. These included the Memetic Algorithm (MA) [20], the discrete Dynamic Convexized Method (DCM) [73], the abcWDP local search algorithm [128], the Hybrid Binary Harmony Search Algorithm (HBHSA) [72], and the DHS-ACO hybrid ant colony algorithm [124]. Tables 6.8-6.12 present a comprehensive comparative analysis across the studied benchmark instances. For the previously published algorithms, we reported the mean value of the evaluation function, which indicates the highest value obtained by those methods. However, for the proposed NFP-DGWO algorithm, we report both the best value (max) and the average value (avg) achieved. This gives us a better understanding

Table 6.8: State-of-the-art comparison on subset of REL_1000_500.

Instance	MA	DCM	abcWDP	HBHSA	DHS-ACO	NFP-DGWO	
						avg	max
101	67101.93	67330.25	**72724.62**	67973.71	**72724.62**	66784.09	69840.08
102	67797.61	70186.90	**72518.22**	70706.86	**72518.22**	67213.43	69362.80
103	66350.99	67496.73	**72129.50**	69151.79	**72129.50**	67222.15	69791.25
104	64618.41	69791.24	**72709.65**	71184.80	**72709.65**	67080.40	69580.34
105	66376.83	69274.29	**75646.13**	72725.20	**75646.13**	68802.74	70001.25
106	65481.64	65110.89	**71258.61**	66461.43	**71258.61**	64814.26	68204.65
107	66245.70	67026.55	**69713.40**	68476.54	**69713.40**	66808.21	68587.17
108	74588.51	73357.63	**75813.21**	72729.34	**75813.21**	72825.77	75147.18
109	62492.66	64548.53	**69475.90**	66023.87	**69475.90**	63233.29	65822.84
110	65171.19	65547.86	**68295.29**	66405.36	**68295.29**	64105.60	67395.07

Table 6.9: State-of-the-art comparison on subset of REL_1000_1000.

Instance	MA	DCM	abcWDP	HBHSA	DHS-ACO	NFP-DGWO	
						avg	max
201	77499.82	78856.3	**81557.74**	80079.58	**81557.74**	80194.45	**81557.74**
202	90464.2	88850.76	**90708.13**	90490.79	**90708.13**	88901.78	**90708.13**
203	86239.21	82551.15	**86239.21**	84491.86	**86239.21**	**86239.21**	**86239.21**
204	81969.05	83666.49	**87075.43**	85057.33	**87075.43**	87075.03	**87075.43**
205	82469.19	84130.23	**86515.95**	85422.67	**86515.95**	84016.44	**86515.95**
206	86881.42	86333.52	**91518.96**	89211.1	**91518.96**	85474.94	89425.94
207	91033.51	89753.32	**93129.25**	92042.88	**93129.25**	93121.04	**93129.25**
208	83667.76	85927.42	**94904.68**	87803.87	**94904.68**	91782.04	91782.04
209	81966.65	84752.54	**87268.97**	85265.19	**87268.97**	86547.65	86755.8
210	85079.98	86229.86	**89962.4**	87917.57	**89962.4**	86420.15	88500.61

of the consistency and overall performance of NFP-DGWO, as well as its ability to achieve high-quality solutions consistently. The best results among all the compared algorithms are highlighted in bold for easy identification.

Based on the results presented in Tables 6.8-6.12, it is evident that the abcWDP and HBHSA algorithms consistently outperform other approaches for the studied benchmark instances. Nevertheless, the proposed NFP-DGWO algorithm proves to be a competitive and effective solution, achieving results that are highly comparable to those obtained by the top-performing algorithms. The analysis further shows that NFP-DGWO outperforms the MA, DCM, and DHS-ACO algorithms across the five benchmark classes in most instances. Additionally, the NFP-DGWO algorithm displays strong performance, obtaining the best results in most instances within the REL_500_1000 and REL_1000_1000 subsets. However, there are some challenges in certain instances, particularly those within the REL_1000_500 subset.

Table 6.10: State-of-the-art comparison on subset of REL_500_1000.

Instance	MA	DCM	abcWDP	HBHSA	DHS-ACO	NFP-DGWO	
						avg	max
401	72948,07	75438,49	**77417,48**	76035,94	**77417,48**	**77417,48**	**77417,48**
402	71454,78	75146,65	**76273,34**	**76273,34**	**76273,34**	75912,48	**76273,34**
403	**74843,96**	71309,10	**74843,96**	72465,39	**74843,96**	**74843,96**	**74843,96**
404	78761,68	76877,34	**78761,69**	77091,37	**78761,69**	**78761,69**	**78761,69**
405	72674,25	75104,28	**75915,90**	75684,28	**75915,90**	75291,70	**75915,90**
406	71791,03	72055,10	**72863,32**	72203,10	**72863,32**	72651,64	**72863,32**
407	73935,28	74443,52	**76365,72**	73650,63	**76365,72**	75844,38	**76365,72**
408	72580,04	74766,64	**77018,83**	74747,72	**77018,83**	**77018,83**	**77018,83**
409	68724,53	71965,11	**73188,62**	71924,64	**73188,62**	**73188,62**	**73188,62**
410	71791,57	73092,48	**73791,66**	73726,39	**73791,66**	73260,19	**73791,66**

Table 6.11: State-of-the-art comparison on subset of REL_1500_1000.

Instance	MA	DCM	abcWDP	HBHSA	DHS-ACO	NFP-DGWO	
						avg	max
501	79132.03	86353.64	**88656.96**	87341.22	**88656.96**	85819.89	87830.42
502	80340.76	84207.64	**86236.91**	84896.64	**86236.91**	82038.48	**86236.91**
503	83277.71	84547.97	**87812.38**	86313.22	**87812.38**	84048.52	86191.79
504	81903.02	82742.72	**85600**	84604.71	**85600**	84139.73	**85600**

Table 6.12: State-of-the-art comparison on subset of REL_1500_1500.

Instance	MA	DCM	abcWDP	HBHSA	DHS-ACO	NFP-DGWO	
						avg	max
601	99044.32	103273.33	**108800.45**	104402.60	**108800.45**	105438.26	**108800.45**
602	98164.23	102390.49	**105611.48**	103152.40	**105611.48**	101249.06	**105611.48**
603	94126.96	98794.90	**105121.02**	104928.00	**105121.02**	98038.52	102568.78
604	103568.86	103522.86	**107733.81**	106694.10	**107733.81**	105427.51	106614.79
605	102404.76	103600.76	**109840.98**	106322.10	**109840.98**	103806.34	107180.51
606	104346.07	102906.98	**107113.07**	104499.70	**107113.07**	105218.21	105218.21
607	105869.44	103297.49	**113180.28**	108241.00	**113180.28**	102903.79	105869.45
608	95671.77	100547.10	**105266.11**	104428.30	**105266.11**	101687.60	**105266.11**
609	98566.94	102506.90	**109472.33**	106122.20	**109472.33**	104733.41	**109472.33**
610	102468.60	109516.88	**113716.97**	**113716.97**	**113716.97**	108229.42	111835.92

6.5 Discussion and analysis

The synergistic combination of exploring infrequent patterns and leveraging the strengths of the population-based framework underpinning the GWO algorithm enabled the NFP-DGWO algorithm to consistently outperform the DGWO algorithm. This superior performance is due to NFP-DGWO's ability to balance exploration and exploitation effectively.

Capitalizing on the inherent strengths of the population-based GWO algorithm, NFP-DGWO maintained a diverse population of solutions, intelligently guiding the search process through the collective information gathered from the entire population. This approach promoted diversification while simultaneously enabling efficient exploitation of promising regions of the search space. Moreover, NFP-DGWO facilitated the exploration of an expanded feasible search space by strategically incorporating infrequent patterns that were rarely present in the elite solutions discovered during the search process. This strategic technique allowed NFP-DGWO to venture into previously unexplored regions of the search space, potentially uncovering high-quality solutions that may have been overlooked.

Furthermore, NFP-DGWO allows the exploration of an expanded feasible search space by incorporating infrequent patterns rarely present in the elite solutions. This strategic approach facilitated the exploration of previously unexplored regions of the search space, potentially leading to the discovery of better solutions.

Despite the increased runtime due to the computationally intensive learning process, the additional computational effort invested by NFP-DGWO paid off in the form of enhanced solution quality. By leveraging the information gained from exploring infrequent patterns, NFP-DGWO could navigate the search space more effectively, avoiding premature convergence and uncovering higher-quality solutions to the WDP instances.

The comprehensive comparative analysis also demonstrated that the proposed NFP-DGWO algorithm is highly competitive and capable of achieving solution qualities that are on par with or better than other state-of-the-art approaches for a significant portion of the studied benchmark instances, positioning it favorably within the existing literature. This favorable position opens up promising avenues for exploring its applicability and performance on other COPs.

6.6 Chapter summary

In this chapter, we introduced two novel algorithms, DGWO and NFP-DGWO, to solve the WDP. DGWO adapted the original GWO metaheuristic to handle the discrete and combinatorial nature of WDP, while NFP-DGWO integrated a learning mechanism exploiting infrequent patterns via the FP-max ML algorithm to enhance solution diversity and exploration. The promising results highlight the potential benefits of incorporating such techniques into metaheuristic search processes for solving complex optimization problems like the WDP.

Despite the encouraging results, further research is needed to develop more robust and versatile optimization techniques that can effectively solve a diverse range of COPs by leveraging the synergies between ML and metaheuristic methods.

Conclusion and Future Work

Machine Learning and metaheuristic optimization fields have experienced considerable growth in recent times, making them crucial components of AI. Their emergence has been pivotal in today's data-driven world, where extracting information from large amounts of data and making sound decisions is vital. Thus, the synergy between these two paradigms holds immense potential for addressing complex real-world optimization challenges in various. This new line of research can lead to new opportunities for addressing intricate problems, promoting innovation, and facilitating efficient decision-making processes across diverse applications.

This thesis embarks on a novel exploration of the synergistic integration of ML and metaheuristic optimization techniques to solve complex problems efficiently. Through several innovative contributions, we have illuminated the potential of combining these complementary paradigms to enhance the performance of ML models and metaheuristic algorithms. Our research has indicated that metaheuristics can effectively optimize the hyperparameters of ML models, leading to improved performance. Furthermore, we developed robust optimization approaches that harness the learning capabilities of ML models to guide and enhance the search process of metaheuristic algorithms when tackling COPs.

Our thesis consists of two main parts. The first part offers a comprehensive overview of the current state-of-the-art and its challenges. We started by providing an in-depth analysis of metaheuristic algorithms and ML techniques. Subsequently, we reviewed existing methods and outlined potential challenges.

The second part of our thesis is dedicated to the core contributions. Here, we presented our new approaches and methodologies in detail, providing a comprehensive analysis and evaluation of their performance and effectiveness. We discussed the steps we took, the data we used, and the results we obtained, giving the reader a clear picture of our research process and its outcomes.

Specifically, we proposed a new optimization algorithm called the hybrid GWO-MVO algorithm. This algorithm combines the strengths of GWO and MVO metaheuristics

to optimize the hyperparameters of ML models efficiently. GWO-MVO was applied to optimize the dropout probabilities for CNNs and tune the hyperparameters for ELMs. Our proposed approaches outperformed baseline methods and achieved competitive performance on benchmark datasets. Inspired by this success, we introduced a novel optimization approach called qGWO-CNN. This approach incorporates quantum computing principles into the GWO algorithm. Our experiments demonstrated the promising performance of the qGWO-CNN approach in optimizing CNN hyperparameters.

Moreover, we explored the practical implications of integrating ML techniques into metaheuristic algorithms for solving COPs. Our approach, which combines the LR model and the SA algorithm, enhances the search process, leading to improved solution quality for binary COPs, particularly MKP and SSP. In our final contribution, we proposed an intelligent algorithm based on GWO to tackle WDP. Our algorithm incorporates a learning mechanism that utilizes frequent item extraction techniques to efficiently explore rare patterns, resulting in improved solution diversity and helping to prevent stagnation at local optima.

The contributions presented in this thesis have laid a strong foundation for further exploration and advancements in the synergistic integration of ML and metaheuristic optimization techniques. However, significant opportunities for continued research and improvement remain.

In the future, the proposed approaches could be extended to more complex DL architectures and optimization problems to evaluate their scalability and generalizability across a broader range of domains and problem instances. Additionally, investigating the integration of RL techniques into metaheuristic algorithms could yield further performance enhancements, leveraging the power of experience-driven learning to guide and refine the optimization process.

Future work could also apply the proposed methodologies to address dynamic and multi-objective optimization problems. The synergistic combination of ML and metaheuristics could provide adaptive and efficient solutions capable of handling evolving objective functions and constraints, as well as balancing multiple conflicting objectives simultaneously.

Beyond these potential research directions, future efforts could also focus on exploring the use of distributed and parallel computing paradigms to accelerate the training and optimization processes. This would enable handling large-scale datasets and complex problem instances that may be computationally intensive or resource-demanding. By leveraging the computational power of modern hardware architectures

and distributed systems, we could unlock new possibilities for tackling increasingly complex optimization challenges.

Overall, the future work outlined here emphasizes the immense potential for further research and innovation in integrating ML and metaheuristic optimization. This integration can lead to more robust, adaptable, and efficient solutions to complex real-world optimization challenges across various applications.

References

[1] Abed-alguni, B. H. and Barhoush, M. (2018). Distributed grey wolf optimizer for numerical optimization problems. *Jordanian J. Comput. Inf. Technol.(JJCIT)*, 4(03):21.

[2] Abualigah, L. (2020). Multi-verse optimizer algorithm: a comprehensive survey of its results, variants, and applications. *Neural Computing and Applications*, 32(16):12381–12401.

[3] Aha, D. W., Kibler, D., and Albert, M. K. (1991). Instance-based learning algorithms. *Machine Learning*, 6(1):37–66.

[4] Ahyar, L. F., Suyanto, S., and Arifianto, A. (2020). Firefly algorithm-based hyperparameters setting of drnn for weather prediction. In *2020 International Conference on Data Science and Its Applications (ICoDSA)*, pages 1–5.

[5] Ali, I. M., Essam, D., and Kasmarik, K. (2020). A novel design of differential evolution for solving discrete traveling salesman problems. *Swarm and Evolutionary Computation*, 52:100607.

[6] Alipour, M. M., Razavi, S. N., Feizi Derakhshi, M. R., and Balafar, M. A. (2018). A hybrid algorithm using a genetic algorithm and multiagent reinforcement learning heuristic to solve the traveling salesman problem. *Neural Computing and Applications*, 30(9):2935–2951.

[7] Araújo, L. A., Leite E Lopes, I., Oliveira, R. M., Silva, S. H. G., Jarochinski E Silva, C. S., and Gomide, L. R. (2022). Simulated annealing in feature selection approach for modeling aboveground carbon stock at the transition between Brazilian Savanna and Atlantic Forest biomes. *Annals of Forest Research*, 65(1):47–63.

[8] Arel, I., Rose, D. C., and Karnowski, T. P. (2010). Deep machine learning - a new frontier in artificial intelligence research [research frontier]. *IEEE Computational Intelligence Magazine*, 5(4):13–18.

[9] Arnold, F. and Sörensen, K. (2019). What makes a vrp solution good? the generation of problem-specific knowledge for heuristics. *Computers & Operations Research*, 106:280–288.

[10] Bacanin, N., Stoean, R., Zivkovic, M., Petrovic, A., Rashid, T. A., and Bezdan, T. (2021). Performance of a Novel Chaotic Firefly Algorithm with Enhanced Exploration for Tackling Global Optimization Problems: Application for Dropout Regularization. *Mathematics*, 9(21):2705.

[11] Bacanin, N., Tuba, E., Bezdan, T., Strumberger, I., Jovanovic, R., and Tuba, M. (2020). Dropout probability estimation in convolutional neural networks by the enhanced bat algorithm. In *2020 International Joint Conference on Neural Networks (IJCNN)*, pages 1–7.

[12] Bacanin, N., Zivkovic, M., Al-Turjman, F., Venkatachalam, K., Trojovský, P., Strumberger, I., and Bezdan, T. (2022). Hybridized sine cosine algorithm with convolutional neural networks dropout regularization application. *Scientific Reports*, 12(1):6302.

[13] Bartlett, P. (1998). The sample complexity of pattern classification with neural networks: the size of the weights is more important than the size of the network. *IEEE Transactions on Information Theory*, 44(2):525–536.

[14] Bean, J. C. (1994). Genetic Algorithms and Random Keys for Sequencing and Optimization. *ORSA Journal on Computing*, 6(2):154–160.

[15] Bello, I., Pham, H., Le, Q. V., Norouzi, M., and Bengio, S. (2016). Neural Combinatorial Optimization with Reinforcement Learning. Publisher: [object Object] Version Number: 3.

[16] Bengio, Y., Frejinger, E., Lodi, A., Patel, R., and Sankaranarayanan, S. (2019). A learning-based algorithm to quickly compute good primal solutions for Stochastic Integer Programs. arXiv:1912.08112 [cs, math].

[17] Blum, C. and Roli, A. (2003). Metaheuristics in combinatorial optimization: Overview and conceptual comparison. *ACM Comput. Surv.*, 35(3):268–308.

[18] Boser, B. E., Guyon, I. M., and Vapnik, V. N. (1992). A training algorithm for optimal margin classifiers. In *Proceedings of the fifth annual workshop on Computational learning theory*, pages 144–152, Pittsburgh Pennsylvania USA. ACM.

[19] Boughaci, D. (2019). Stochastic local search based feature selection for intrusion detection. In Bramer, M. and Petridis, M., editors, *Artificial Intelligence XXXVI*, pages 404–417, Cham. Springer International Publishing.

[20] Boughaci, D., Benhamou, B., and Drias, H. (2009). A memetic algorithm for the optimal winner determination problem. *Soft Computing*, 13(8-9):905–917.

[21] Boussaïd, I., Lepagnot, J., and Siarry, P. (2013). A survey on optimization metaheuristics. *Information Sciences*, 237:82–117.

[22] Calvet, L., de Armas, J., Masip, D., and Juan, A. A. (2017). Learnheuristics: hybridizing metaheuristics with machine learning for optimization with dynamic inputs. *Open Mathematics*, 15(1):261–280.

[23] Chaturvedi, D. K. (2008). *Soft Computing*, volume 103 of *Studies in Computational Intelligence*. Springer Berlin Heidelberg, Berlin, Heidelberg.

[24] Chen, R., Yang, B., Li, S., and Wang, S. (2020). A self-learning genetic algorithm based on reinforcement learning for flexible job-shop scheduling problem. *Computers & Industrial Engineering*, 149:106778.

[25] Chu, P. and Beasley, J. (1998). A Genetic Algorithm for the Multidimensional Knapsack Problem. *Journal of Heuristics*, 4(1):63–86.

[26] Cortes, C. and Vapnik, V. (1995). Support-vector networks. *Machine Learning*, 20(3):273–297.

[27] De Rosa, G. H., Papa, J. P., and Yang, X.-S. (2018). Handling dropout probability estimation in convolution neural networks using meta-heuristics. *Soft Computing*, 22(18):6147–6156.

[28] Deng, L. (2012). The mnist database of handwritten digit images for machine learning research. *IEEE Signal Processing Magazine*, 29(6):141–142.

[29] Dey, A. K., Gupta, G. P., and Sahu, S. P. (2023). Hybrid meta-heuristic based feature selection mechanism for cyber-attack detection in iot-enabled networks. *Procedia Computer Science*, 218:318–327. International Conference on Machine Learning and Data Engineering.

[30] Dhaenens, C. and Jourdan, L. (2016). *Metaheuristics for Big Data*. Wiley, 1 edition.

[31] Ding, X., Ding, G., Han, J., and Tang, S. (2018). Auto-balanced filter pruning for efficient convolutional neural networks. In *AAAI Conference on Artificial Intelligence*.

[32] Duarte, A., Laguna, M., and Marti, R. (2018). *Metaheuristics for Business Analytics*. EURO Advanced Tutorials on Operational Research. Springer International Publishing, Cham.

[33] Díaz De León-Hicks, E., Conant-Pablos, S. E., Ortiz-Bayliss, J. C., and Terashima-Marín, H. (2023). Addressing the Algorithm Selection Problem through an Attention-Based Meta-Learner Approach. *Applied Sciences*, 13(7):4601.

[34] Elhani, D., Megherbi, A., Zitouni, A., Dornaika, F., Sbaa, S., and Taleb-Ahmed, A. (2023). Optimizing convolutional neural networks architecture using a modified particle swarm optimization for image classification. *Expert Systems with Applications*, 229:120411.

[35] Eskandari, S. and Seifaddini, M. (2023). Online and offline streaming feature selection methods with bat algorithm for redundancy analysis. *Pattern Recognition*, 133:109007.

[36] Fujishima, Y., Leyton-Brown, K., and Shoham, Y. (1999). Taming the computational complexity of combinatorial auctions: Optimal and approximate approaches. In *IJCAI*, volume 99, pages 548–553.

[37] Fukushima, K. (1980). Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. *Biological Cybernetics*, 36(4):193–202.

[38] Glover, F. (1986). Future paths for integer programming and links to artificial intelligence. *Computers & Operations Research*, 13(5):533–549.

[39] Godínez-Bautista, A., Padierna, L. C., Rojas-Domínguez, A., Puga, H., and Carpio, M. (2018). *Bio-inspired Metaheuristics for Hyper-parameter Tuning of Support Vector Machine Classifiers*, pages 115–130. Springer International Publishing, Cham.

[40] Goodfellow, I., Bengio, Y., and Courville, A. (2016). *Deep learning*. Adaptive computation and machine learning. The MIT Press, Cambridge, Massachusetts.

[41] Guo, Y., Lim, A., Rodrigues, B., and Zhu, Y. (2006). Heuristics for a bidding problem. *Computers & Operations Research*, 33(8):2179–2188.

[42] Han, J. and Kamber, M. (2012). *Data mining: concepts and techniques*. Elsevier, Burlington, MA, 3rd ed edition.

[43] Hansen, P. B. (1992). Simulated Annealing.

[44] Hossain, M. T., Teng, S. W., Zhang, D., Lim, S., and Lu, G. (2019). Distortion robust image classification using deep convolutional neural network with discrete cosine transform. In *2019 IEEE International Conference on Image Processing (ICIP)*, pages 659–663.

[45] hua Hao, J., Liu, M., hua Lin, J., and Wu, C. (2016). A hybrid differential evolution approach based on surrogate modelling for scheduling bottleneck stages. *Computers & Operations Research*, 66:215–224.

[46] Huang, G.-B., Zhu, Q.-Y., and Siew, C.-K. (2004). Extreme learning machine: a new learning scheme of feedforward neural networks. In *2004 IEEE International Joint Conference on Neural Networks (IEEE Cat. No.04CH37541)*, volume 2, pages 985–990 vol.2.

[47] Hull, J. (1994). A database for handwritten text recognition research. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 16(5):550–554.

[48] Hutter, F., Kotthoff, L., and Vanschoren, J. (2019). *Automated Machine Learning: Methods, Systems, Challenges*. Springer Publishing Company, Incorporated, 1st edition.

[49] Hutter, F., Stuetzle, T., Leyton-Brown, K., and Hoos, H. H. (2014). ParamILS: An Automatic Algorithm Configuration Framework. Publisher: [object Object] Version Number: 1.

[50] Jia, Y., Shelhamer, E., Donahue, J., Karayev, S., Long, J., Girshick, R., Guadarrama, S., and Darrell, T. (2014). Caffe: Convolutional Architecture for Fast Feature Embedding. Publisher: [object Object] Version Number: 1.

[51] Jung, D., Kang, D., and Kim, J. H. (2018). Development of a Hybrid Harmony Search for Water Distribution System Design. *KSCE Journal of Civil Engineering*, 22(4):1506–1514.

[52] Kaelbling, L. P., Littman, M. L., and Moore, A. W. (1996). Reinforcement Learning: A Survey. Publisher: [object Object] Version Number: 1.

[53] Karimi-Mamaghan, M., Mohammadi, M., Meyer, P., Karimi-Mamaghan, A. M., and Talbi, E.-G. (2022). Machine learning at the service of meta-heuristics for solving combinatorial optimization problems: A state-of-the-art. *European Journal of Operational Research*, 296(2):393–422.

[54] Karimi-Mamaghan, M., Mohammadi, M., Pirayesh, A., Karimi-Mamaghan, A. M., and Irani, H. (2020). Hub-and-spoke network design under congestion: A learning based metaheuristic. *Transportation Research Part E: Logistics and Transportation Review*, 142:102069.

[55] Kellerer, H., Pferschy, U., and Pisinger, D. (2004). *Multidimensional Knapsack Problems*, pages 235–283. Springer Berlin Heidelberg, Berlin, Heidelberg.

[56] Kirkpatrick, S., Gelatt, C. D., and Vecchi, M. P. (1983). Optimization by Simulated Annealing. *Science*, 220(4598):671–680.

[57] Kleinbaum, D. G. (1994). *Logistic Regression*. Springer New York, New York, NY.

[58] Kleinberg, J. and Tardos, E. (2006). *Algorithm design*. Pearson/Addison-Wesley, Boston.

[59] Koopialipoor, M. and Noorbakhsh, A. (2020). Applications of Artificial Intelligence Techniques in Optimizing Drilling. In Azizi, A., editor, *Emerging Trends in Mechatronics*. IntechOpen.

[60] Krizhevsky, A. (2009). Learning multiple layers of features from tiny images.

[61] Krizhevsky, A., Nair, V., and Hinton, G. (2010a). Cifar-10 (canadian institute for advanced research).

[62] Krizhevsky, A., Nair, V., and Hinton, G. (2010b). Cifar-100 (canadian institute for advanced research).

[63] Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2017). ImageNet classification with deep convolutional neural networks. *Communications of the ACM*, 60(6):84–90.

[64] Kumar, V. and Yadav, S. M. (2022). A state-of-the-Art review of heuristic and metaheuristic optimization techniques for the management of water resources. *Water Supply*, 22(4):3702–3728.

[65] Lam, S. K., Pitrou, A., and Seibert, S. (2015). Numba: a LLVM-based Python JIT compiler. In *Proceedings of the Second Workshop on the LLVM Compiler Infrastructure in HPC*, pages 1–6, Austin Texas. ACM.

[66] Lecun, Y., Bottou, L., Bengio, Y., and Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324.

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

[68] Li, H., Kadav, A., Durdanovic, I., Samet, H., and Graf, H. P. (2016). Pruning Filters for Efficient ConvNets. Publisher: [object Object] Version Number: 3.

[69] Li, Y., Lin, X., and Liu, J. (2021). An Improved Gray Wolf Optimization Algorithm to Solve Engineering Problems. *Sustainability*, 13(6):3208.

[70] Li, Y., Lu, G., Zhou, L., and Jiao, L. (2017). Quantum inspired high dimensional hyperparameter optimization of machine learning model. In *2017 International Smart Cities Conference (ISC2)*, pages 1–6.

[71] Liao, H. and Wang, T. (2013). Research project of subset sum problem.

[72] Lin, G. and Li, Z. (2019). A hybrid binary harmony search algorithm for solving the winner determination problem. *Int. J. Innov. Comput. Appl.*, 10(1):59–68.

[73] Lin, G., Zhu, W., and Ali, M. M. (2016). An effective discrete dynamic convexized method for solving the winner determination problem. *Journal of Combinatorial Optimization*, 32(2):563–593.

[74] Liu, S. and Deng, W. (2015). Very deep convolutional neural network based image classification using small training sample size. In *2015 3rd IAPR Asian Conference on Pattern Recognition (ACPR)*, pages 730–734.

[75] Lopes Silva, M. A., de Souza, S. R., Freitas Souza, M. J., and Bazzan, A. L. C. (2019). A reinforcement learning-based multi-agent framework applied for solving routing and scheduling problems. *Expert Systems with Applications*, 131:148–171.

[76] Lu, C., Gao, L., Li, X., Hu, C., Yan, X., and Gong, W. (2020). Chaotic-based grey wolf optimizer for numerical and engineering optimization problems. *Memetic Computing*, 12(4):371–398.

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

[78] López Jaimes, A., Coello Coello, C. A., and Chakraborty, D. (2008). Objective reduction using a feature selection technique. In *Proceedings of the 10th annual conference on Genetic and evolutionary computation*, pages 673–680, Atlanta GA USA. ACM.

[79] Malik, H., Iqbal, A., Joshi, P., Agrawal, S., and Bakhsh, F. I., editors (2021). *Metaheuristic and Evolutionary Computation: Algorithms and Applications*, volume 916 of *Studies in Computational Intelligence*. Springer Singapore, Singapore.

[80] Marcot, B. G. and Hanea, A. M. (2021). What is an optimal value of k in k-fold cross-validation in discrete Bayesian network analysis? *Computational Statistics*, 36(3):2009–2031.

[81] Martin, S., Ouelhadj, D., Beullens, P., Ozcan, E., Juan, A. A., and Burke, E. K. (2016). A multi-agent based cooperative approach to scheduling and routing. *European Journal of Operational Research*, 254(1):169–178.

[82] Masci, J., Meier, U., Cireşan, D., and Schmidhuber, J. (2011). Stacked Convolutional Auto-Encoders for Hierarchical Feature Extraction. In Honkela, T., Duch, W., Girolami, M., and Kaski, S., editors, *Artificial Neural Networks and Machine Learning – ICANN 2011*, volume 6791, pages 52–59. Springer Berlin Heidelberg, Berlin, Heidelberg. Series Title: Lecture Notes in Computer Science.

[83] Mirjalili, S., Mirjalili, S. M., and Hatamlou, A. (2016). Multi-Verse Optimizer: a nature-inspired algorithm for global optimization. *Neural Computing and Applications*, 27(2):495–513.

[84] Mirjalili, S., Mirjalili, S. M., and Lewis, A. (2014). Grey wolf optimizer. *Advances in Engineering Software*, 69:46–61.

[85] Mitchell, T. M. (2013). *Machine learning*. McGraw-Hill series in Computer Science. McGraw-Hill, New York, nachdr. edition.

[86] Mladenović, N. and Hansen, P. (1997). Variable neighborhood search. *Computers & Operations Research*, 24(11):1097–1100.

[87] Mohakud, R. and Dash, R. (2022). Designing a grey wolf optimization based hyper-parameter optimized convolutional neural network classifier for skin cancer detection. *Journal of King Saud University - Computer and Information Sciences*, 34(8, Part B):6280–6291.

[88] Moradi, B. (2020). The new optimization algorithm for the vehicle routing problem with time windows using multi-objective discrete learnable evolution model. *Soft Computing*, 24(9):6741–6769.

[89] Muro, C., Escobedo, R., Spector, L., and Coppinger, R. (2011). Wolf-pack (canis lupus) hunting strategies emerge from simple rules in computational simulations. *Behavioural Processes*, 88(3):192–197.

[90] Nakisa, B., Rastgoo, M. N., Rakotonirainy, A., Maire, F., and Chandran, V. (2018). Long short term memory hyperparameter optimization for a neural network based emotion recognition framework. *IEEE Access*, 6:49325–49338.

[91] Nasiri, M. M., Salesi, S., Rahbari, A., Salmanzadeh Meydani, N., and Abdollai, M. (2019). A data mining approach for population-based methods to solve the JSSP. *Soft Computing*, 23(21):11107–11122.

[92] Nazaré, T. S., da Costa, G. B. P., Contato, W. A., and Ponti, M. (2018). Deep convolutional neural networks and noisy images. In Mendoza, M. and Velastín, S., editors, *Progress in Pattern Recognition, Image Analysis, Computer Vision, and Applications*, pages 416–424, Cham. Springer International Publishing.

[93] Ng, A. Y. (2004). Feature selection, L_1 vs. L_2 regularization, and rotational invariance. In *Twenty-first international conference on Machine learning - ICML '04*, page 78, Banff, Alberta, Canada. ACM Press.

[94] Niu, L.-Y., Wei, Y., and Liu, Y. (2023). Event-driven spiking neural network based on membrane potential modulation for remote sensing image classification. *Engineering Applications of Artificial Intelligence*, 123:106322.

[95] Osaba, E., Villar-Rodriguez, E., Del Ser, J., Nebro, A. J., Molina, D., LaTorre, A., Suganthan, P. N., Coello Coello, C. A., and Herrera, F. (2021). A tutorial on the design, experimentation and application of metaheuristic algorithms to real-world optimization problems. *Swarm and Evolutionary Computation*, 64:100888.

[96] Pintelas, E., Livieris, I. E., and Pintelas, P. E. (2021). A convolutional autoencoder topology for classification in high-dimensional noisy image datasets. *Sensors*, 21(22).

[97] Prates, M. O. R., Avelar, P. H. C., Lemos, H., Lamb, L., and Vardi, M. (2018). Learning to Solve NP-Complete Problems - A Graph Neural Network for Decision TSP. Publisher: [object Object] Version Number: 3.

[98] Preeti and Deep, K. (2022). A random walk grey wolf optimizer based on dispersion factor for feature selection on chronic disease prediction. *Expert Systems with Applications*, 206:117864.

[99] Puchinger, J., Raidl, G. R., and Pferschy, U. (2010). The Multidimensional Knapsack Problem: Structure and Algorithms. *INFORMS Journal on Computing*, 22(2):250–265.

[100] Qing, Y., Zeng, Y., Li, Y., and Huang, G.-B. (2020). Deep and wide feature based extreme learning machine for image classification. *Neurocomputing*, 412:426–436.

[101] Ramos, I., Goldbarg, M., Goldbarg, E., and Neto, A. (2005). Logistic regression for parameter tuning on an evolutionary algorithm. In *2005 IEEE Congress on Evolutionary Computation*, volume 2, pages 1061–1068 Vol. 2.

[102] Rezoug, A., Bader-el den, M., and Boughaci, D. (2022). Application of Supervised Machine Learning Methods on the Multidimensional Knapsack Problem. *Neural Processing Letters*, 54(2):871–890.

[103] Samuel, A. L. (1959). Some studies in machine learning using the game of checkers. *IBM Journal of Research and Development*, 3(3):210–229.

[104] Sandholm, T. (2006). Optimal winner determination algorithms.

[105] Sarker, I. H. (2021). Machine Learning: Algorithms, Real-World Applications and Research Directions. *SN Computer Science*, 2(3):160.

[106] Shaheen, M. A., Hasanien, H. M., and Alkuhayli, A. (2021). A novel hybrid gwo-pso optimization technique for optimal reactive power dispatch problem solution. *Ain Shams Engineering Journal*, 12(1):621–630.

[107] Simonyan, K. and Zisserman, A. (2014). Very Deep Convolutional Networks for Large-Scale Image Recognition. Publisher: [object Object] Version Number: 6.

[108] Singh, P., Chaudhury, S., and Panigrahi, B. K. (2021). Hybrid mpso-cnn: Multi-level particle swarm optimized hyperparameters of convolutional neural network. *Swarm and Evolutionary Computation*, 63:100863.

[109] Sinha, T., Haidar, A., and Verma, B. (2018). Particle swarm optimization based approach for finding optimal values of convolutional neural network parameters. In *2018 IEEE Congress on Evolutionary Computation (CEC)*, pages 1–6.

[110] Smith, K. A. (1999). Neural Networks for Combinatorial Optimization: A Review of More Than a Decade of Research. *INFORMS Journal on Computing*, 11(1):15–34.

[111] Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. (2014). Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(56):1929–1958.

[112] Sun, J., Feng, B., and Xu, W. (2004). Particle swarm optimization with particles having quantum behavior. In *Proceedings of the 2004 Congress on Evolutionary Computation (IEEE Cat. No.04TH8753)*, volume 1, pages 325–331 Vol.1.

[113] Sun, J., Zhang, H., Zhou, A., Zhang, Q., and Zhang, K. (2019). A new learning-based adaptive multi-objective evolutionary algorithm. *Swarm and Evolutionary Computation*, 44:304–319.

[114] Talbi, E.-G. (2009). *Metaheuristics: from design to implementation*. John Wiley & Sons, Hoboken, N.J. OCLC: ocn230183356.

[115] Taye, M. M. (2023). Understanding of Machine Learning with Deep Learning: Architectures, Workflow, Applications and Future Directions. *Computers*, 12(5):91.

[116] Tupe, P. R., Vibhute, P. M., and Sayyad, M. A. (2020). An architecture combining convolutional neural network (cnn) with batch normalization for apparel image classification. In *2020 IEEE International Symposium on Sustainable Energy, Signal Processing and Cyber Security (iSSSC)*, pages 1–6.

[117] Vapnik, V. N. (1964). A note on one class of perceptrons. *Automat. Rem. Control*, 25:821–837.

[118] Vijay, R. K. and Nanda, S. J. (2019). A quantum grey wolf optimizer based declustering model for analysis of earthquake catalogs in an ergodic framework. *Journal of Computational Science*, 36:101019.

[119] Vinyals, O., Fortunato, M., and Jaitly, N. (2015). Pointer Networks. Publisher: [object Object] Version Number: 2.

[120] Wang, Y., Pan, S., Li, C., and Yin, M. (2020). A local search algorithm with reinforcement learning based repair procedure for minimum weight independent dominating set. *Information Sciences*, 512:533–548.

[121] Wang, Y., Zhang, H., and Zhang, G. (2019). cpso-cnn: An efficient pso-based algorithm for fine-tuning hyper-parameters of convolutional neural networks. *Swarm and Evolutionary Computation*, 49:114–123.

[122] Wojtusiak, J., Warden, T., and Herzog, O. (2012). The learnable evolution model in agent-based delivery optimization. *Memetic Computing*, 4(3):165–181.

[123] Wolpert, D. and Macready, W. (1997). No free lunch theorems for optimization. *IEEE Transactions on Evolutionary Computation*, 1(1):67–82.

[124] Wu, J., Fan, M., Liu, Y., Zhou, Y., Yang, N., Yin, M., Information Science and Technology, Northeast Normal University, Changchun, China, School of Science, Beijing University of Posts and Telecommunications, Bejing, China, CHEARI Certification & Testing Co., Ltd., Beijing, China, and Key Laboratory of Applied Statistics of MOE, Northeast Normal University, Changchun, China (2022). A hybrid ant colony algorithm for the winner determination problem. *Mathematical Biosciences and Engineering*, 19(3):3202–3222.

[125] Xie, W., Wang, L., Yu, K., Shi, T., and Li, W. (2023). Improved multi-layer binary firefly algorithm for optimizing feature selection and classification of microarray data. *Biomedical Signal Processing and Control*, 79:104080.

[126] Yamasaki, T., Honma, T., and Aizawa, K. (2017). Efficient optimization of convolutional neural networks using particle swarm optimization. In *2017 IEEE Third International Conference on Multimedia Big Data (BigMM)*, pages 70–73.

[127] Zennaki, M. and Cherif, A. E. (2010). A New Machine Learning based Approach for Tuning Metaheuristics for the Solution of Hard Combinatorial Optimization Problems. *Journal of Applied Sciences*, 10(18):1991–2000.

[128] Zhang, H., Cai, S., Luo, C., and Yin, M. (2017). An efficient local search algorithm for the winner determination problem. *Journal of Heuristics*, 23(5):367–396.

[129] Zhao, F., Zhang, L., Cao, J., and Tang, J. (2021). A cooperative water wave optimization algorithm with reinforcement learning for the distributed assembly no-idle flowshop scheduling problem. *Computers & Industrial Engineering*, 153:107082.

[130] Zheng, S., Zhou, X., Zheng, X., and Ge, M. (2020). Improved quantum-behaved particle swarm algorithm based on levy flight. *Mathematical Problems in Engineering*, 2020:1–10.

[131] Zheng, Y., Fu, X., and Xuan, Y. (2019). Data-driven optimization based on random forest surrogate. In *2019 6th International Conference on Systems and Informatics (ICSAI)*, pages 487–491.

[132] Zhou, Y., Hao, J.-K., and Duval, B. (2022). Frequent pattern-based search: A case study on the quadratic assignment problem. *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, 52(3):1503–1515.

[133] Črepinšek, M., Liu, S.-H., and Mernik, M. (2013). Exploration and exploitation in evolutionary algorithms: A survey. *ACM Computing Surveys*, 45(3):1–33.