الجمهورية الجزائرية الديمقراطية الشعبية
People's Democratic Republic of Algeria
وزارة التعليم العالي والبحث العلمي
Ministry of Higher Education and Scientific Research
جامعة مصطفى اسطمبولي معسكر
Mustapha Stambouli University

Faculty of Exact Sciences
Department of Computer Science

DOCTORAL THESIS

Field : Computer Science
Speciality : Artificial Intelligence and its Application.

Multi-constrained optimization using bio-inspired approaches

Presented by : **Boualem Sid Ahmed El Mahdi**

Le : 10 / 12 / 2024

In front of the jury :

President	Yahyaoui Khadidja	Professor	University of Mascara
Supervisor	Meftah Boudjelal	Professor	University of Mascara
Examiner	Benyahia Kada	MCA	University of Saida
Examiner	Fekir Youcef	MCA	University of Mascara

Academic Year 2024-2025

الجمهورية الجزائرية الديمقراطية الشعبية
République Algérienne Démocratique et Populaire
وزارة التعليم العالي والبحث العلمي
Ministère de l'Enseignement Supérieur et de la Recherche Scientifique
جامعة مصطفى اسطمبولي معسكر
Université Mustapha Stambouli de Mascara

Faculté des Sciences Exactes
Département Informatique

Thèse de Doctorat 3ème cycle

Filière : Informatique
Spécialité : Intelligence Artificielle et ses Applications

Optimisation multi-contraintes par des approches bio-inspirées

Présentée par : **Boualem Sid Ahmed El Mahdi**

Le : 10 / 12 / 2024

Devant le jury :

Présidente	Yahyaoui Khadidja	Professeur	Université de Mascara
Directeur de thèse	Meftah Boudjelal	Professeur	Université de Mascara
Examinateur	Benyahia Kada	MCA	Université de Saida
Examinateur	Fekir Youcef	MCA	Université de Mascara

Année universitaire 2024-2025

Acknowledgements

First and foremost, I want to say ***Alhamdulillah***. I thank Allah for giving me the strength, courage, and determination to complete this thesis.

I would like to express my sincere gratitude to my supervisor, Professor ***Meftah Boudjelal***, for his consistent support, encouragement, and invaluable guidance throughout the course of my PhD. His insightful comments and thoughtful recommendations inspired me to push the boundaries of my thinking and contributed significantly to the quality of this work.

I would also like to express my gratitude to the members of the jury for agreeing to evaluate my thesis:

- Mrs. Khadidja Yahyaoui, Full professor at Mustapha Stambouli University of Mascara, for the honor of being part of the jury and presiding over it.
- Mr. Kada Benyahia, Associated professor at University of Saida, for accepting to be part of the jury and dedicating his time to reading and enriching the thesis.
- Mr. Youcef Fekir, Associated professor at Mustapha Stambouli University of Mascara, for his interest in this work by agreeing to be part of the jury.

I would also like to express my gratitude to Professor Fatima Debbat for her invaluable assistance throughout my doctoral journey. Her insightful guidance and support have been instrumental in shaping this work.

Finally, I would like to extend my heartfelt thanks to my family and friends for their unconditional support during these intense academic years.

Dedication

My dear parents,
Who have always supported me with love, affection, and care. Your sacrifices and efforts have shaped me into who I am today.

My brothers,
For their constant encouragement and unwavering belief in me.

All my family,
For their support and belief in me throughout my journey.

My friends,
For your support, and shared moments that have enriched my journey.

To all those who have known and supported me,
May Allah bless you with health and happiness.

Abstract

This thesis focuses on the optimization of constrained problems using bio-inspired algorithms, a growing field in artificial intelligence and computational optimization. Constrained optimization problems (COPs) are pervasive in real-world applications, requiring innovative methods to efficiently handle the inherent complexities. The main objective of this research is to develop and evaluate nature-inspired techniques, particularly focusing on Differential Evolution (DE) to solve various single objective constrained optimization problems. Also our research explore the application of the grey wolf optimizer bio-inspired algorithm in solving the Traveling Salesman Problem.

This study begins by addressing the key challenges in COPs, focusing on handling constraint violation and maintaining feasibility within complex search spaces. To address these challenges, a novel adaptive coordinate system based on constrained differential evolution is proposed, where the DE is adaptively performed in either an Eigen coordinate system or original coordinate system. This flexibility enables the algorithm to dynamically direct the search, enhancing its ability to explore and exploit promising feasible regions, resulting in improved convergence rates and solution quality. Additionally, an enhanced Greedy Discrete Grey Wolf Optimizer (GD-GWO) is developed for discrete optimization, demonstrating superior performance against multiple benchmark instances of the Traveling Salesman Problem (TSP).

The experimental results highlight the effectiveness of these bio-inspired algorithms in balancing exploration and exploitation, showing competitive performance compared to state-of-the-art techniques. The findings suggest that the proposed methods not only provide promising results for constrained optimization but also offer a foundation for solving more complex, real-world problems. Future work will investigate broader applications, multi-objective optimization, and further enhancements of these algorithms for better scalability and adaptability in diverse problem domains.

keywords : Constrained optimization problems, Bio-inspired algorithms, Differential evolution, Eigen coordinate system, Constraint-handling techniques, GWO, TSP.

Résumé

Cette thèse porte sur l'optimisation des problèmes sous-contraints en utilisant des algorithmes bio-inspirés, un domaine en pleine croissance dans l'intelligence artificielle et l'optimisation computationnelle. Les problèmes d'optimisation sous contraintes (COPs) sont omniprésents dans les applications réelles, nécessitant des méthodes innovantes pour gérer efficacement les complexités inhérentes. L'objectif principal de cette recherche est de développer et d'évaluer des techniques inspirées de la nature, en mettant particulièrement l'accent sur l'évolution différentielle (DE) pour résoudre divers problèmes d'optimisation sous contrainte mono-objectif. De plus, notre recherche explore l'application de l'algorithme bio-inspiré du Grey Wolf Optimizer (GWO) pour résoudre le problème du voyageur de commerce (TSP).

L'étude commence par explorer les défis liés aux COPs, notamment la gestion des violations des contraintes et le maintien de la faisabilité au sein des espaces de recherche. Un nouveau système de coordonnées adaptative basé sur l'algorithme d'évolution différentielle sous contraintes est introduit, où l'algorithme est effectué de manière adaptative dans un système de coordonnées propres "Eigen", ou dans le système de coordonnées d'origine. Cette flexibilité permet à l'algorithme de diriger dynamiquement la recherche, améliorant ainsi sa capacité à explorer et exploiter des régions faisables prometteuses. En outre, la thèse propose une version améliorée du GWO algorithm adaptée à l'optimisation discrète, qui démontre une performance supérieure lors des tests sur le problème de TSP.

Les résultats expérimentaux mettent en évidence l'efficacité de ces algorithmes bio-inspirés dans l'équilibre entre exploration et exploitation, montrant des performances compétitives par rapport à d'autres techniques. Les résultats montrent que les méthodes proposées offrent non seulement des résultats prometteurs pour l'optimisation sous contraintes, mais fournissent également une base pour résoudre des problèmes réels plus complexes. Les travaux futurs examineront des applications plus larges, l'optimisation multi-objectifs, ainsi que des améliorations supplémentaires de ces algorithmes pour une meilleure évolutivité et adaptabilité dans divers domaines problématiques.

Mots-clés : Problèmes d'optimisation sous contraintes, Algorithmes bio-inspirés, Évolution différentielle, Système de coordonnées propres, Techniques de gestion des contraintes, GWO, TSP.

ملخص

تركز هذه الأطروحة على تحسين المشكلات المقيدة باستخدام خوارزميات مستوحاة من الطبيعة، وهو مجال سريع النمو في الذكاء الاصطناعي والتحسين الحسابي. تنتشر مشكلات التحسين المقيدة في العديد من تطبيقات العالم الواقعي، مما يتطلب أساليب مبتكرة لإدارة التعقيدات المتأصلة بفعالية. يتمثل الهدف الرئيسي من هذا البحث في تطوير وتقييم تقنيات مستوحاة من الطبيعة، مع التركيز بشكل خاص على التطور التفاضلي (DE) لحل مختلف مشكلات التحسين المقيدة. كما يستكشف بحثنا أيضًا تطبيق خوارزمية محسّن الذئب الرمادي (GWO) المستوحاة من الطبيعة في حل مشكلة البائع المتجول .(TSP) تبدأ الدراسة باستكشاف التحديات المرتبطة بمشكلات التحسين المقيدة، بما في ذلك كيفية التعامل مع انتهاكات القيود والحفاظ على إيجاد حلول قابلة للتنفيذ في فضاءات البحث. يتم تقديم نظام إحداثيات تكيفي جديد لخوارزمية DE المقيدة، حيث يتم إجراء التقاطع في نظام إحداثيات ،Eigen مما يعزز قدرة الخوارزمية على استكشاف واستغلال المناطق القابلة للتطبيق. بالإضافة إلى ذلك، تقترح الأطروحة تحسينًا لخوارزمية GD-GWO المخصصة للتحسين المنفصل، والتي أظهرت أداءً متفوقًا عند اختبارها على عينات قياسية من مشكلة البائع المتجول. تسلط النتائج التجريبية الضوء على فعالية هذه الخوارزميات المستوحاة من الطبيعة في تحقيق التوازن بين الاستكشاف والاستغلال، مما يظهر أداءً تنافسيًا مقارنة بأحدث التقنيات. تشير النتائج إلى أن الأساليب المقترحة لا توفر نتائج واعدة فقط لتحسين المشكلات المقيدة، ولكنها توفر أيضًا أساسًا لحل مشكلات أكثر تعقيدًا في العالم الواقعي. ستتناول الأبحاث المستقبلية تطبيقات أوسع، وتحسينات متعددة الأهداف، وتعزيز هذه الخوارزميات لتحسين قابليتها للتوسع والتكيف مع مجالات مشكلات متنوعة.

الكلمات الرئيسية: مشاكل التحسين المقيدة، الخوارزميات المستوحاة من الأحياء، التطور التفاضلي، نظام إحداثيات إيجن، تقنيات التعامل مع القيود، ،TSP ،GWO

Contents

List of Figures

List of Tables

General Introduction

Context and Motivation

Optimization plays a crucial role in a variety of fields such as engineering, finance, logistics, and artificial intelligence. Many real-world problems require finding the best solutions while adhering to certain restrictions, known as constraints. This is critical for decision-making processes, whether it's minimizing the cost of constructing a bridge while ensuring its safety or managing a company's resources within budget limitations. These are examples of constrained optimization problems (COPs), which occur when solutions must meet specific conditions or limits. Constraints can range from simple, such as staying within a budget, to complex, such as meeting safety regulations while keeping a product affordable. In practice, COPs are encountered in many industries, making it essential to find efficient methods to solve them. However, traditional optimization methods often struggle with complex constraints, especially when they are non-linear or dynamic. As a result, finding reliable and accurate solutions in such situations is challenging. This has led to a growing interest in alternative approaches like bio-inspired algorithms, which mimic natural processes like evolution and collective animal behavior. These algorithms offer flexibility and adaptability in searching for solutions, particularly in large and complex solution spaces, making them highly effective for constrained optimization problems. However, despite the progress in bio-inspired algorithms, many existing methods still face challenges in balancing the optimization of the objective function while respecting all constraints. This gap in performance, particularly in real-world applications, serves as the main motivation for the research presented in this thesis.

Objectives

To address the aforementioned challenges, this thesis proposes several techniques based on bio-inspired algorithms that aim to provide optimal solutions or high-quality approximations for constrained optimization problems. The primary objectives of the research are:

- Develop adaptive bio-inspired algorithms that efficiently solve constrained optimization problems, ensuring solutions respect the constraints while optimizing the objective function.
- Apply bio-inspired algorithms to complex real-world problems to demonstrate their effectiveness in handling discrete, nonlinear, and dynamic constraints.

Contributions

This thesis has led to two major contributions, with the first being of particular importance:

The first contribution introduces a novel adaptive differential evolution algorithm specifically designed to tackle constrained optimization problems. The algorithm leverages an adaptive coordinate system to improve the search process and effectively navigate complex solution spaces. It was rigorously evaluated on standard constrained optimization benchmarks and demonstrated superior performance, not only in terms of finding optimal or near-optimal solutions but also in reducing constraint violations. This innovative approach addresses a key gap in the field by offering a flexible and efficient solution for handling difficult constraints. The significance of this work was recognized through its publication in a Q1-ranked journal, highlighting its impact and contribution to the advancement of constrained optimization techniques.

The second contribution focuses on applying bio-inspired algorithms to real-world constrained optimization problems, specifically the TSP. We developed the Greedy Discrete Grey Wolf Optimizer (GD-GWO), an enhanced version of the Grey Wolf Optimizer, tailored for discrete optimization. This algorithm demonstrated competitive performance on several TSP instances and has the potential for broader applications in other routing problems. A part of this work has been presented at an international conference, reflecting its relevance and effectiveness in addressing practical optimization challenges.

Thesis Plan

This thesis is structured as follows:

Chapter I introduces the fundamental concepts of optimization, with a focus on constrained optimization problems (COPs). It covers the classification of optimization problems, types of constraints, challenges in solving COPs, and the motivation for using bio-inspired algorithms. The chapter sets the stage for the structure and contributions of the thesis.

Chapter II reviews bio-inspired algorithms and their application to optimization. It provides an overview of key algorithms like genetic algorithms, differential evolution, and

grey wolf optimizer, highlighting their effectiveness in handling constraints. The chapter also discusses common constraint-handling techniques in bio-inspired algorithms.

Chapter III details the methodology behind the proposed improvements to Differential Evolution for constrained optimization. It explains the modifications, such as the adaptive coordinate system, and evaluates the improved DE's performance against benchmark problems, along with a comparative analysis of other methods.

Chapter IV explores the application of the Grey Wolf Optimizer to the Traveling Salesman Problem. It describes how the algorithm is adapted for discrete optimization and introduces the greedy discrete grey wolf optimizer (GD-GWO). The chapter presents the results of its application on TSP benchmarks and compares its performance with other algorithms.

Finally, General conclusion is presented, summarizing the key findings and contributions of the thesis. It also discusses the practical implications and potential future research directions, including extending the proposed methods to other complex optimization problems.

Chapter I

Optimization: Theory and Practice

Chapter I
Optimization: Theory and Practice

I.1 Introduction

This chapter provides a comprehensive overview of the fundamental concepts and challenges associated with optimization, particularly in the context of constrained problems. The primary goal is to establish a strong theoretical foundation that will support the subsequent discussions on the methodologies and approaches employed in this thesis. By defining key terms, categorizing optimization problems, and exploring the general basics of an optimization process, this chapter lays the groundwork for understanding the complexities involved in constrained optimization and the need for innovative solutions. This background is crucial for appreciating the relevance and application of bio-inspired algorithms that will be discussed in the next chapter.

I.2 Overview of Optimization

Optimization is a fundamental process used across various fields to identify the most efficient, effective, or optimal solution from a set of possible alternatives. At its core, (Fig.I.1) optimization seeks to improve performance, reduce costs, or enhance outcomes based on specific criteria, making it invaluable in areas ranging from engineering design and resource allocation to machine learning and finance (Sioshansi et al, 2017). The optimization process revolves around finding the best solution by adjusting variables within a system to achieve a desired objective, all while adhering to any imposed limitations or constraints (Diwekar, 2020).

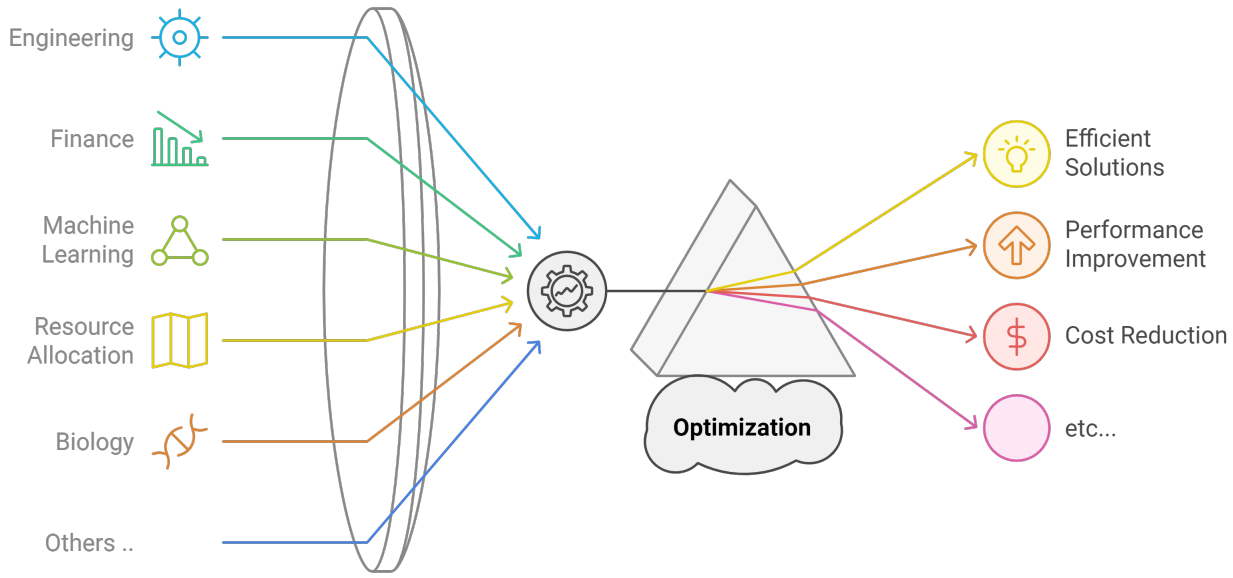


Figure I.1: Optimization Application

I.2.1 Fundamental Components of Optimization

To understand how optimization functions in practice, it's essential to break it down into its fundamental components. These include the objective function, decision variables, and constraints, all of which form the backbone of any optimization process Andréasson et al (2020).

1. Objective Function

The objective function is the mathematical expression that needs to be optimized (minimized or maximized). It quantifies the performance or efficiency of a system or process. The choice of the objective function is critical, as it determines the criteria for evaluating different solutions (Rao, 2019).

2. Decision Variables

These are the variables that can be adjusted or controlled to optimize the objective function. The values of decision variables determine the specific solution to the problem. The dimensionality and nature (continuous or discrete) of the decision variables significantly influence the complexity of the optimization problem Diwekar (2020).

3. Constraints

Constraints are conditions or limitations that the solutions must satisfy. They can be in the form of equalities or inequalities. Constraints play a vital role in optimization

problems as they define the feasible region within which the optimal solution must lie Diwekar (2020).

I.2.2 Local and Global Optimum

Optimization problems often require the identification of the best solution from a set of feasible solutions. These solutions can be categorized as either local or global optimum (Boyd and Vandenberghe, 2004). Understanding the distinction between local and global optima is crucial in optimization theory, as it directly impacts the effectiveness of optimization algorithms in finding the best possible solution. Below is a detailed discussion of local and global optima, along with their mathematical formulations and representative figure Fig.I.2.

1. Global Optimum

A global optimum refers to the absolute best solution in the entire feasible region of an optimization problem. In other words, it is the point at which the objective function reaches its minimum or maximum value, depending on whether the problem is a minimization or maximization task. The global optimum can be represented mathematically as follows:

For a minimization problem, the global optimum x^* is defined as:

$$x^* = \text{argmin}\, f(x), \quad \text{for all } x \in S \tag{I.1}$$

where S is the feasible solution space, and $f(x)$ is the objective function.

For a maximization problem, the global optimum x^* is defined as:

$$x^* = \text{argmax}\, f(x), \quad \text{for all } x \in S \tag{I.2}$$

2. Local Optimum

A local optimum, on the other hand, refers to the best solution within a smaller region of the feasible space (Boyd and Vandenberghe, 2004). It may not be the absolute best solution (global optimum), but it represents the optimal solution in a specific neighborhood around a point. A local optimum can be mathematically defined as follows:

For a minimization problem, a local optimum satisfies the following condition:

$$f(x^*) \leq f(x), \quad \text{for all } x \text{ in a neighborhood of } x^* \tag{I.3}$$

Similarly, for a maximization problem, a local optimum x^* satisfies the condition:

$$f(x^*) \geq f(x), \quad \text{for all } x \text{ in a neighborhood of } x^* \tag{I.4}$$

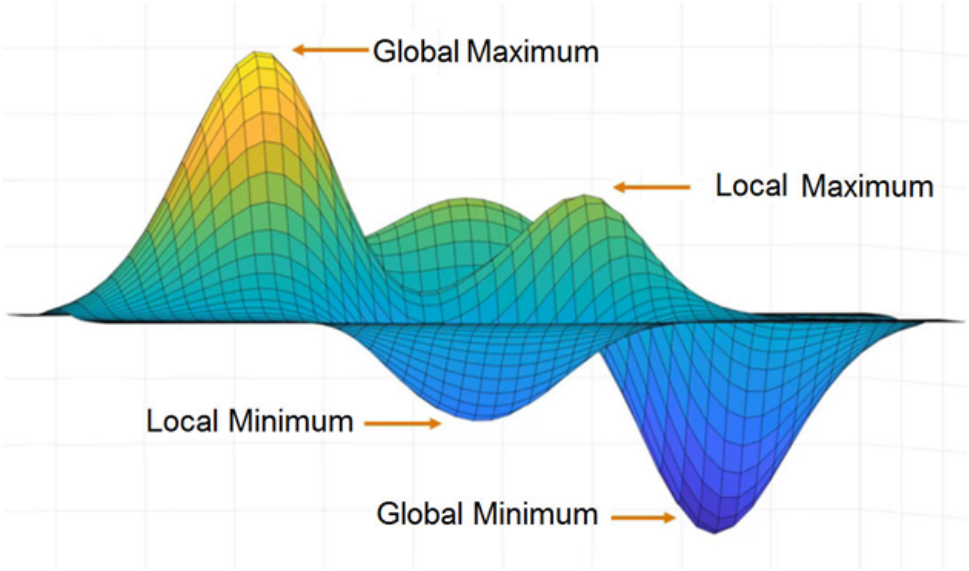


Figure I.2: local and global optimum (mathworks.com)

It is important to note that while every global optimum is also a local optimum, the reverse is not necessarily true. In many optimization problems, particularly non-convex ones (Fig.I.3), there may be multiple local optima, and optimization algorithms must be designed to avoid getting trapped in these sub-optimal solutions.

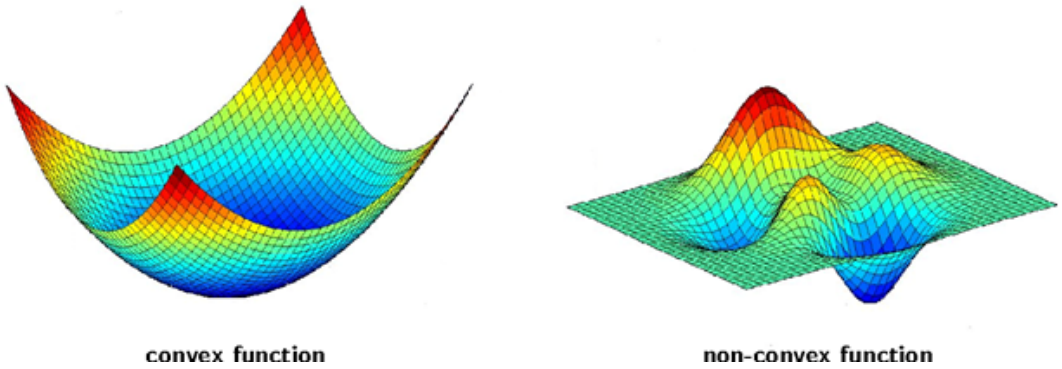


Figure I.3: 3D convex and non-convex function illustration

I.2.3 Linear and Non-Linear Optimization

Linear and non-linear optimization are two fundamental categories of optimization problems that differ based on the relationships between the decision variables (Fletcher, 2013).

1. Linear Optimization

Linear optimization, also known as linear programming, involves problems where the objective function and constraints are linear (Fletcher, 2013). This means that all relationships in the model can be expressed as linear equations or inequalities. The general

form of a linear optimization problem can be stated as follows:

$$\text{maximize or minimize } z = c^T x \tag{I.5}$$

subject to

$$Ax \leq b \tag{I.6}$$

$$x \geq 0 \tag{I.7}$$

where:

- z is the objective function to be maximized or minimized,
- c is a vector of coefficients,
- x is a vector of decision variables,
- A is a matrix of coefficients representing the constraints,
- b is a vector representing the right-hand side of the constraints.

Linear optimization is widely used in various fields, including finance, logistics, and manufacturing, due to its efficiency and the availability of well-established solution methods such as the Simplex algorithm.

2. Non-Linear Optimization

Non-linear optimization involves problems where the objective function or at least one of the constraints is non-linear (Fletcher, 2013). Non-linear problems can be more complex and challenging to solve compared to linear problems due to the potential for multiple local optima and the need for specialized algorithms. The general form of a non-linear optimization problem can be expressed as follows:

$$\text{maximize or minimize } z = f(x) \tag{I.8}$$

subject to

$$g_i(x) \leq 0, \quad i = 1, \ldots, m \tag{I.9}$$

$$h_j(x) = 0, \quad j = 1, \ldots, p \tag{I.10}$$

where:

- $f(x)$ is the non-linear objective function,
- $g_i(x)$ are the non-linear inequality constraints,
- $h_j(x)$ are the non-linear equality constraints.

Non-linear optimization problems arise in many applications, such as engineering design, economics, and machine learning. Due to their complexity, various methods, including gradient-based approaches, evolutionary algorithms, and heuristic methods, are often employed to find solutions.

I.2.4 Continuous and Discrete Optimization

Continuous and discrete optimization are two fundamental types of optimization problems distinguished by the nature of the decision variables (Andréasson et al, 2020).

1. Continuous Optimization

Continuous optimization refers to optimization problems in which the decision variables can take any value within a specified range. This type of optimization is typically characterized by continuous functions, meaning that small changes in the decision variables result in small changes in the objective function (Andréasson et al, 2020).

2. Discrete Optimization

Discrete optimization, on the other hand, involves problems where the decision variables can only take specific discrete values, such as integers or binary values. This type of optimization often arises in combinatorial problems where the solution involves selecting from a finite set of options (Rajeev and Krishnamoorthy, 1992).

I.3 Combinatorial Optimization

Combinatorial optimization is a specialized area within the broader field of optimization, intersecting with operations research, algorithm theory, and computational complexity theory (Papadimitriou and Yannakakis, 1991). It is dedicated to discovering the optimal grouping, ordering, or arrangement of discrete elements using mathematical approaches. The primary objective in combinatorial optimization is to identify the best possible solution from a finite set of feasible solutions. These solutions are inherently discrete or can be transformed into discrete forms. The optimal solution in such problems is typically represented by an integer, a subset, a permutation, or a graph structure. Combinatorial optimization finds widespread application in various engineering domains, where problems such as routing, task allocation, and scheduling are often modeled as combinatorial optimization problems. These problems are then solved using a range of methods that fall into two distinct categories: exact methods (Nemhauser and Wolsey, 1988) and stochastic methods (Talbi, 2009). Exact methods are deterministic, providing a guarantee of finding the optimal solution, but they can be computationally expensive, especially for large-scale

problems. On the other hand, stochastic methods, while not guaranteeing an exact optimal solution, they offer a more practical approach by providing high-quality approximate solutions within reasonable computational limits. These methods are particularly useful in scenarios where exact methods are infeasible due to the problem's complexity or size.

I.4 Unconstrained Optimization

Unconstrained optimization refers to the process of finding the maximum or minimum of an objective function without any restrictions or constraints on the decision variables (Andrei, 2008). This type of optimization is fundamental in various fields, including mathematics, economics, and engineering, as it simplifies the problem-solving process by removing the complexities introduced by constraints.

In unconstrained optimization, the objective function $f(x)$ is typically defined over a continuous domain, where x represents the decision variables. The goal is to find the optimal solution x^* that either maximizes or minimizes the function, represented mathematically as:

$$x^* = \operatorname{argmax} f(x) \quad \text{or} \quad x^* = \operatorname{argmin} f(x) \tag{I.11}$$

I.4.1 Resolution Approaches

Various methods are employed to solve unconstrained optimization problems, including:

- **Gradient Descent:** This iterative method uses the gradient (or derivative) of the objective function to guide the search for optimal solutions. It updates the current solution by moving in the direction of the steepest descent until a minimum is reached (Bertsekas and Tsitsiklis, 1999).
- **Newton's Method:** This method improves convergence by using second-order derivative information (the Hessian matrix) to find optimal points. It is more efficient than gradient descent for functions that are well-approximated by a quadratic model (Wright and Nocedal, 1999).
- **Stochastic Methods:** Techniques like simulated annealing and genetic algorithms can also be applied to unconstrained optimization problems, particularly when the objective function is complex or noisy (Marti et al, 2008).

I.4.2 Neighboring Search

Neighboring search (or neighborhood search) is an important concept in optimization, particularly relevant for solving discrete and combinatorial optimization problems Hansen

and Mladenović (2001). It involves exploring the solution space by evaluating solutions that are in close proximity to a current solution, in the hopes of improving it. The idea is to move from one solution to a nearby solution, or "neighbor," based on predefined rules, to find better solutions incrementally. The working mechanism is as follows:

1. **Neighborhood Structure:** A neighborhood structure defines what constitutes a "neighbor" of a solution (e.g., swapping two elements in a solution sequence).

2. **Local Search:** A common neighboring search method that iteratively explores neighboring solutions to find improvements. When a better solution is discovered, it becomes the new current solution, and this process repeats until no further improvements are possible, resulting in a local optimum (Lourenço et al, 2019).

Several optimization algorithms use neighboring search in different ways, depending on how they explore the solution space. Common methods include:

1. **Hill Climbing:** This is a simple neighboring search method where the algorithm selects the best neighboring solution and moves to it, climbing toward better solutions. However, hill climbing can get stuck in a local optimum if no better neighbors exist(Selman and Gomes, 2006).

2. **Simulated Annealing:** This technique introduces randomness to escape local optima by occasionally allowing worse moves (to a less optimal neighbor), simulating the annealing process in metallurgy. As the algorithm progresses, the probability of accepting worse solutions decreasesVan Laarhoven et al (1987).

3. **Tabu Search:** Tabu search enhances the basic neighboring search by keeping track of recently visited solutions, or "tabu" moves, to avoid cycling back to them. This memory mechanism helps the algorithm explore new regions of the solution space (Glover and Laguna, 1998).

I.5 Constrained Optimization

Constrained optimization is a critical area of study within the broader field of optimization, particularly because real-world problems are rarely unconstrained (Fioretto et al, 2018; Uryasev, 2013). Most practical optimization problems involve constraints—conditions or limitations that any feasible solution must satisfy. These constraints can stem from physical laws, resource limitations, safety requirements, or other practical considerations (Zhang et al, 2006).

Constrained optimization involves finding the best solution (i.e., minimizing or maximizing an objective function) while ensuring that all the constraints of the problem are met. Mathematically, a constrained optimization problem can be formulated as follows:

$$\min/\max\ f(\mathbf{x}) \tag{I.12}$$

$$\text{subject to: } \begin{cases} h_j(\mathbf{x}) = 0, & i = 1, \ldots, n_h \\ g_j(\mathbf{x}) \leq 0, & j = 1, \ldots, n_g \end{cases} \tag{I.13}$$

The criterion to be optimized is evaluated through an objective function $f(\mathbf{x})$, where $f : D^{(n_x)} \to \mathbb{R}^m$, $n_x \geq 1$, $m \geq 1$. This objective function can take on any mathematical form, whether linear, convex, or otherwise. It may even be treated as a black-box function, as long as it assigns a value from the objective space $\mathbb{R}^m$ to a valid solution $\mathbf{x} \in D^{(n_x)}$.

In cases where the objective space is $\mathbb{R}^m$ with $m \geq 2$, the problem is referred to as multi-objective optimization; if $m = 1$, it is termed single-objective optimization. Valid solutions exist within the variable space $D^{(n_x)}$, which can be discrete, continuous, or a combination of both. The dimensionality of a valid solution within this space may be fixed or variable, depending on the problem at hand.

The optimization problem can be framed as either a minimization or maximization of f. Notably, any maximization problem can be converted into a minimization problem by multiplying the objective function by -1, and vice versa. The global optimal solution $\mathbf{x}^*$ is found when $f(\mathbf{x}^*) \leq f(\mathbf{x}) \quad \forall \mathbf{x} \in D^{(n_x)}$.

I.5.1 Types of Constraints

Constraints in optimization problems can be broadly categorized into different types:

- **Boundary Constraints:** These define the upper and lower bound values for each decision variable of the optimization problem. The boundaries for each dimension enforce a hyper-cube within which possible problem solutions must be contained.

- **Inequality Constraints:** These are conditions where the constraint function must be either less than or equal to, or greater than or equal to, a specific value. For instance, in a budgeting problem, an inequality constraint might require that the total cost does not exceed a certain budget limit:

  $$g_i(\mathbf{x}) \leq 0 \tag{I.14}$$

  Inequality constraints define boundaries within which the solution must lie, often creating a more complex feasible region with potentially disjointed or non-convex shapes.

- **Equality Constraints:** These are conditions where the constraint function must equal a specific value. For example, in an engineering design problem, an equality constraint might specify that a component's weight must exactly match a given

target:

$$h_i(\mathbf{x}) = 0 \tag{I.15}$$

Equality constraints are often more restrictive and can complicate the optimization process, as they reduce the feasible region to a lower-dimensional space. They can also be transformed into an inequality constraint through the transformation:

$$|\ h_i(\mathbf{x})| - \epsilon \leq 0 \tag{I.16}$$

In real-world optimization problems, solutions must satisfy a set of constraints, which typically consist of equality constraints $h_i(\mathbf{x}) = 0$ and inequality constraints $g_i(\mathbf{x}) \leq 0$. These constraints define the feasible region within the variable space, where only solutions that meet all the constraints are considered valid.

I.5.2 Constraint Handling Techniques

In optimization problems, particularly those involving constraints, finding solutions that satisfy both the objective function and the constraints is crucial. Various methods have been developed to handle constraints effectively, each with its own strengths and applicability depending on the problem's nature. This section delves into some of the most widely used constraint handling techniques: Penalty Methods, Barrier Methods, Lagrange Multiplier Method, and Heuristic and Metaheuristic Techniques (Askarzadeh, 2016).

1. Penalty Methods

Penalty methods are a classic approach to handling constraints by transforming a constrained optimization problem into an unconstrained one (Luenberger et al, 2016). This is achieved by adding a penalty term to the objective function, which imposes a cost for violating any constraints. The transformed objective function, known as the penalized objective function, can be expressed as:

$$F(\mathbf{x}) = f(\mathbf{x}) + \sum_i \lambda_i \max(0, g_i(\mathbf{x}))^p + \sum_j \mu_j |h_j(\mathbf{x})|^q \tag{I.17}$$

Where $F(\mathbf{x})$ is the original objective function, $g_i(\mathbf{x})$ and $h_j(\mathbf{x})$ represent inequality and equality constraints respectively, and λ_i, μ_j, p, and q are penalty parameters. The values of these parameters influence the severity of the penalty for constraint violations. Penalty methods are versatile and can be tailored for various optimization problems, but careful calibration of the penalty parameters is essential for ensuring effective constraint handling.

2. Barrier Methods

Barrier methods approach constraint handling by adding a barrier function to the objective function that becomes infinite as the solution approaches the boundary of the feasible region (Wright, 1992). This method ensures that the optimization process remains within the feasible region, making it particularly useful for problems with inequality constraints. The barrier-augmented objective function can be formulated as:

$$F(\mathbf{x}) = f(\mathbf{x}) - \sum_i \frac{1}{g_i(\mathbf{x})} \tag{I.18}$$

Where $g_i(\mathbf{x}) \geq 0$ are the inequality constraints. The barrier function effectively "traps" the solution within the feasible region by imposing an infinite cost for approaching the constraint boundary. Barrier methods are particularly useful when it is crucial to maintain feasibility during the optimization process, but their application is somewhat limited to specific types of constraints.

3. Lagrange Multiplier Method

The Lagrange Multiplier Method is a powerful technique for handling constraints, particularly when dealing with equality constraints (Birgin and Martínez, 2014). This method introduces Lagrange multipliers, which serve as additional variables that enforce the constraints while optimizing the objective function. The Lagrangian function is defined as:

$$L(\mathbf{x}, \lambda, \mu) = f(\mathbf{x}) + \sum_i \lambda_i g_i(\mathbf{x}) + \sum_j \mu_j h_j(\mathbf{x}) \tag{I.19}$$

where λ_i and μ_j are the Lagrange multipliers corresponding to the inequality $g_i(\mathbf{x}) \geq 0$ and equality $h_j(\mathbf{x}) = 0$ constraints, respectively. The optimization process involves finding the saddle point of the Lagrangian, where the derivatives with respect to both the original variables $\mathbf{x}$ and the multipliers λ and μ are zero. The Lagrange Multiplier Method is a fundamental tool in constrained optimization, particularly in theoretical analysis and in situations where constraints are explicitly defined.

4. Heuristic and Metaheuristic Techniques

Heuristic and Metaheuristic Techniques form a broad category of methods designed to find high-quality solutions for complex optimization problems, particularly when traditional methods struggle with non-linearity, large search spaces, or intricate constraints. These techniques offer flexibility in addressing constraints either directly or indirectly, making them effective tools for constrained optimization.

Heuristics are problem-specific strategies aimed at generating good solutions within a reasonable timeframe. Common heuristic approaches include greedy algorithms, local

search, and constructive methods (Vince, 2002). Although heuristics do not guarantee optimal solutions, they are effective in managing constraints by embedding constraint-handling mechanisms within the search process or by using repair strategies to correct infeasible solutions.

Metaheuristics, on the other hand, are higher-level frameworks that guide the search process and can be applied to a wide range of problems. Well-known metaheuristics include Genetic Algorithms (GAs), Simulated Annealing (SA), Particle Swarm Optimization (PSO), and Differential Evolution (DE). These methods often handle constraints using techniques such as penalty functions, repair mechanisms, and feasibility rules, which will be discussed in detail in the next chapter.

Heuristic and metaheuristic approaches are particularly valuable for tackling real-world optimization problems with complex or difficult-to-model constraints. Their flexibility and adaptability make them indispensable tools in constrained optimization.

I.5.3 Search Landscape

The search landscape represents the multidimensional space formed by the parameters, objectives, and constraints of an optimization problem. This landscape defines the environment within which an optimization algorithm operates, seeking the optimal solution (Zou et al, 2022).

For visualization Figure.I.4, in problems with two variables, the search space can be depicted as a 2D plane, while in three-variable problems, it forms a 3D cube. As the number of variables increases beyond three, the search space becomes a hypercube. However, if the range of each variable differs, the search space takes the shape of a hyper-rectangle rather than a cube, adding complexity to the optimization process.

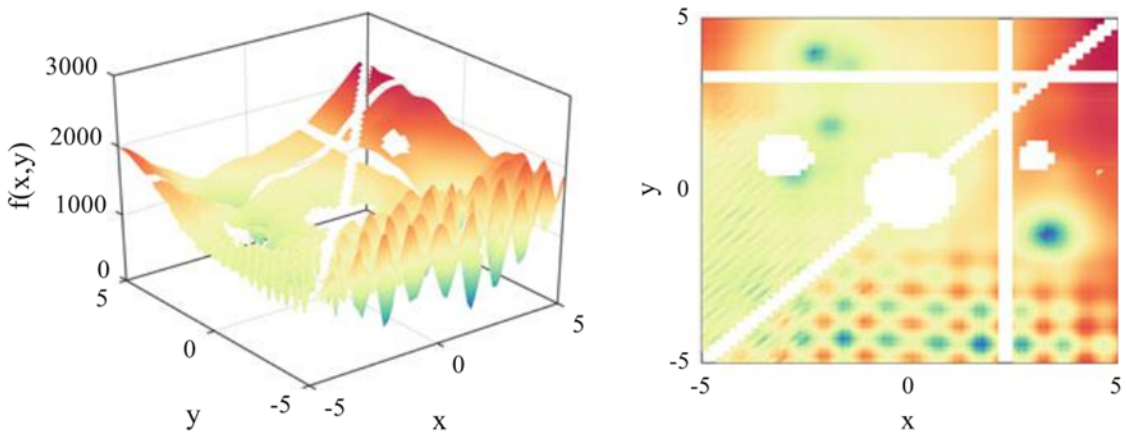


Figure I.4: A search landscape with several constraints (Mirjalili and Mirjalili, 2019)

The nature of the search landscape depends on the relationship between the inputs and outputs. It can be unimodal or multimodal. A unimodal landscape has a single peak

or valley, representing the global optimum with no local optima to mislead the search. In contrast, a multimodal landscape contains multiple peaks and valleys, where the global optimum is one of several possible solutions, making it challenging to identify the best one (Mirjalili and Mirjalili, 2019). Real-world problems often exhibit multimodal landscapes with dynamic properties, requiring optimization algorithms to adapt and track shifting optima over time. Additionally, some landscapes feature multiple global optimum with identical objective values, further complicating the search process.

Constraints significantly influence the structure of the search landscape, introducing gaps or 'islands' that an optimization algorithm must navigate. These constraints can divide the landscape into isolated regions, making it crucial for the algorithm to explore and identify promising optima within these regions. Boundaries set by constraints also restrict the search space, and the algorithm must be capable of managing particles that stray outside these feasible regions, relocating them appropriately (Sallam et al, 2020).

The challenges presented by the search landscape are especially pronounced in real-world scenarios, where the landscape is typically more complex. Such landscapes are characterized by a high number of parameters, numerous constraints, multiple local optima, deceptive slopes, isolated global optima, dynamic changes, and uncertainties. Effective optimization requires algorithms equipped with sophisticated mechanisms to navigate these complexities and reliably converge on the optimal solution.

I.5.4 Challenges in Constrained Optimization

Constrained optimization involves optimizing an objective function subject to a set of constraints, which could be equality or inequality constraints. These constraints represent the limits or requirements that the solution must satisfy, making the optimization process more complex compared to unconstrained problems. Several challenges arise when dealing with constrained optimization, making it a highly intricate and demanding field of study.

1. **Complexity of the Feasible Region:** One of the primary challenges in constrained optimization is the complexity of the feasible region, which is the set of all solutions that satisfy the constraints. The feasible region can be highly irregular, discontinuous, or even disconnected, especially in the case of non-linear constraints. This complexity can make it difficult for optimization algorithms to efficiently explore the search space and find feasible solutions that also optimize the objective function. Traditional optimization methods often struggle with this aspect, as they may become trapped in local optima or fail to find a feasible solution altogether Zhang and Xu (2023).

2. **Trade-Off Between Objective Function and Constraints:** Another significant challenge is the inherent trade-off between optimizing the objective function

and satisfying the constraints. In many cases, the optimal solution for the objective function may lie outside the feasible region, requiring a balance between achieving the best possible objective value and adhering to the constraints. This trade-off is particularly challenging in multi-objective optimization problems, where multiple conflicting objectives must be optimized simultaneously while still satisfying all constraints. The ability to effectively manage this trade-off is crucial for the success of any constrained optimization algorithm (Wang et al, 2020).

3. **Constraint Violation and Infeasibility:** Constraint violation and the potential infeasibility of solutions are major hurdles in constrained optimization (Zhang and Xu, 2023; Petsagkourakis et al, 2020). A solution is considered infeasible if it does not satisfy all the given constraints. In many optimization problems, especially those involving complex or non-linear constraints, finding a feasible solution can be as challenging as optimizing the objective function. This is particularly true when the constraints are highly restrictive, leaving only a small portion of the search space as feasible. Algorithms need to incorporate mechanisms to handle constraint violations, either by penalizing infeasible solutions or by repairing them to bring them back into the feasible region.

4. **Sensitivity to Initial Conditions and Parameter Setting:** Many optimization algorithms, especially those that are iterative in nature, are sensitive to the initial conditions and parameter settings. In constrained optimization, this sensitivity can lead to drastically different outcomes depending on the starting point or the chosen parameters. Poor initial conditions might lead to early convergence to suboptimal or infeasible solutions. Similarly, inappropriate parameter settings, such as the choice of penalty factors in penalty methods, can either overly restrict the search space or fail to enforce constraint satisfaction adequately (Schulte and Nissen, 2020).

5. **Computational Complexity:** The computational complexity of constrained optimization is another significant challenge. Evaluating the feasibility of solutions and calculating the objective function can be computationally expensive, particularly when dealing with large-scale problems or problems with complex constraints. The need to perform these evaluations repeatedly throughout the optimization process can result in high computational costs, making it impractical to apply certain algorithms to large or real-time problems. This challenge necessitates the development of more efficient algorithms that can handle constraints without excessive computational overhead Haeser et al (2019).

6. **Handling Dynamic and Uncertain Constraints:** In real-world applications, constraints are not always static; they can change over time or be subject to uncertainty. Dynamic constraints add another layer of complexity, requiring algo-

rithms to adapt in real-time as the constraints evolve. Similarly, uncertain constraints, where the exact form or parameters of the constraints are not known in advance, require robust optimization techniques that can handle variability and unpredictability. Addressing these challenges is crucial for the successful application of constrained optimization algorithms in practical scenarios Petsagkourakis et al (2020).

I.5.5 Constrained Optimization Approaches

Constrained optimization problems present unique challenges that require specialized approaches to address effectively. Traditional optimization methods, while foundational, often encounter limitations when faced with complex, high-dimensional problems involving multiple constraints. These methods typically struggle with issues such as handling non-linearity, navigating fragmented feasible regions, and efficiently finding global optima.

In light of these challenges, recent advancements have introduced more versatile techniques designed to overcome the constraints faced by traditional methods. Bio-inspired algorithms, which mimic natural processes to solve optimization problems, have emerged as powerful tools in this context. These algorithms leverage evolutionary principles, swarm behaviors, and other natural phenomena to explore complex search spaces and handle constraints more effectively. This section explores the shortcomings of traditional approaches and highlights the potential of nature-inspired algorithms as a powerful and innovative alternative.

Traditional optimization methods, such as gradient-based techniques, linear programming (LP), and branch-and-bound methods, have long been the go-to tools for solving optimization problems. However, these methods exhibit significant limitations when applied to complex constrained optimization problems.

- **Gradient-Based Methods:** Gradient-based methods, which rely on the calculation of derivatives, are well-suited for smooth, continuous optimization problems (Liu et al, 2021). However, they face significant challenges in constrained optimization, particularly when the objective function or constraints are non-differentiable, non-linear, or contain discontinuities. These methods are also prone to getting trapped in local optima, especially in non-convex problems, limiting their ability to find the global optimum. Furthermore, gradient-based methods often require feasible starting points and can struggle to find feasible solutions in highly constrained problems.

- **Linear Programming (LP):** Linear programming techniques are effective for problems where the objective function and constraints are linear (Asorey-Cacheda et al, 2017). However, many real-world optimization problems involve non-linear

relationships, rendering LP methods inadequate. Additionally, LP methods are not well-suited for problems with complex or non-convex feasible regions, where the feasible solutions form irregular or disconnected spaces.

- **Branch-and-Bound and Other Exact Methods:** Exact methods, such as branch-and-bound, guarantee finding the global optimum by systematically exploring the search space. However, their applicability is limited to small or moderately-sized problems due to their exponential time complexity. As the problem size increases, these methods become computationally infeasible, especially when dealing with large-scale problems or those with a high number of constraints (Morrison et al, 2016).

In traditional optimization methods, constrained problems often prove difficult, especially in complex, high-dimensional spaces where multiple constraints interact. These approaches commonly face challenges like handling non-linearity, navigating fragmented feasible regions, and efficiently finding global optima. In light of these challenges, the limitations of conventional methods motivate the search for more adaptable solutions. Bio-inspired algorithms, drawing on natural processes like evolution and swarm behaviors, have emerged as effective alternatives (Yang, 2020). In the next chapter, we will explore these innovative techniques, examining how they can better handle the complexities of constrained optimization compared to traditional approaches.

I.6 Multi-Objective Optimization

Multi-objective optimization (MOO) is a branch of optimization that involves problems with more than one objective function to be optimized simultaneously (Deb et al, 2016). Unlike single-objective optimization, where the goal is to find a solution that maximizes or minimizes a single objective, MOO focuses on optimizing multiple, often conflicting, objectives. This type of problem is common in real-world scenarios, where decision-makers must balance trade-offs between competing goals.

I.6.1 Definition and Formulation

A general multi-objective optimization problem can be mathematically formulated as:

$$\text{minimize} \quad \mathbf{f}(\mathbf{x}) = [f_1(\mathbf{x}), f_2(\mathbf{x}), \ldots, f_k(\mathbf{x})] \tag{I.20}$$

subject to:

$$\begin{aligned} h_j(\mathbf{x}) &= 0, \quad j = 1, \ldots, m \\ g_j(\mathbf{x}) &\leq 0, \quad j = 1, \ldots, p \end{aligned} \tag{I.21}$$

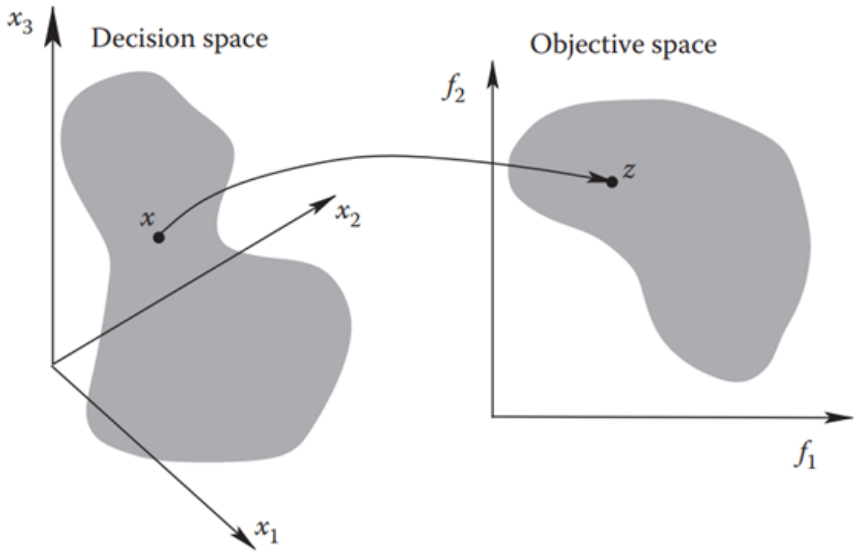


Figure I.5: multi-objective optimization decision and objective space

where $f_i(\mathbf{x}), i = 1, \ldots, k$ are the objective functions, which are to be minimized (or maximized). The objective functions $f_1(\mathbf{x}), f_2(\mathbf{x}), \ldots, f_k(\mathbf{x})$ often conflict, meaning improving one objective may lead to the deterioration of another. For example, in engineering design, reducing cost might lead to lower performance, while maximizing performance may increase costs.

I.6.2 Pareto Optimality

In multi-objective optimization, there is usually no single solution that simultaneously optimizes all objectives Xue et al (2003). Instead, a set of optimal solutions, known as the Pareto front or Pareto optimal solutions, is sought. A solution $\mathbf{x}^*$ is considered Pareto optimal if no other solution $\mathbf{x}$ exists that improves at least one objective without worsening another. Formally, a solution $\mathbf{x}^*$ is Pareto optimal if for every $\mathbf{x}$:

$$f_i(\mathbf{x}) \leq f_i(\mathbf{x}^*) \quad \forall i \quad \text{and} \quad \exists j \text{ such that } f_j(\mathbf{x}) < f_j(\mathbf{x}^*) \tag{I.22}$$

This represents the trade-offs between conflicting objectives where any improvement in one objective would lead to the degradation of at least one other objective.

I.6.3 Pareto Front

The Pareto front is the set of all Pareto optimal solutions in the objective space. It provides a visual representation of the trade-offs between objectives. Decision-makers often use the Pareto front to select a preferred solution based on their priorities. The solutions on the Pareto front are non-dominated, meaning no other solutions outperform them across all objectives. The nature of the Pareto front can vary depending on the

problem; it may be convex, concave, or discontinuous, influencing the difficulty of finding the optimal set of solutions.

I.6.4 Solution Techniques for Multi-objective Optimization

Several approaches have been developed to solve multi-objective optimization problems. These methods can be broadly categorized as scalarization techniques, evolutionary algorithms, and hybrid methods.

1. Scalarization Techniques

Scalarization methods convert a multi-objective problem into a single-objective problem by combining the objectives into a weighted sum or another scalar function (Giagkiozis and Fleming, 2015). The main scalarization techniques are:

- **Weighted Sum Method:** Each objective is multiplied by a weight factor representing its relative importance, and the sum of the weighted objectives is minimized (Bazgan et al, 2022). The problem becomes:

$$\text{minimize} \quad f(\mathbf{x}) = \sum_{i=1}^{k} \lambda_i f_i(\mathbf{x}) \tag{I.23}$$

  where λ_i are the weight factors. Varying the weights generates different Pareto optimal solutions. However, this method may fail to capture non-convex Pareto fronts.

- **Epsilon-Constraint Method:** One objective is optimized while others are treated as constraints with upper bounds. This approach can produce Pareto optimal solutions for non-convex problems, but finding an appropriate epsilon ϵ for each constraint is challenging (Fan et al, 2016).

- **Goal Programming:** The decision-maker specifies target values for each objective, and the goal is to minimize the deviation from these targets (Tanino et al, 2013).

Evolutionary Algorithms (EAs) are widely employed as solution techniques for tackling multi-objective optimization problems, thanks to their ability to explore complex search spaces and generate diverse solution sets. While this chapter provides an overview of the fundamental concepts, a detailed discussion of EAs, their mechanisms, and their applications as effective optimization tools will be covered in the next chapter.

I.7 Conclusion

In this chapter, we have established the foundational principles necessary for understanding the intricacies of optimization, particularly within the realm of constrained problems. By defining key concepts, categorizing optimization problems, and examining the general structure of an optimization process, we have created a theoretical framework that underpins the discussions in subsequent chapters. The exploration of constrained optimization has highlighted both the complexities involved and the need for innovative approaches to tackle these challenges effectively. This chapter sets the stage for the deeper investigation into advanced techniques that will follow, with a particular emphasis on bio-inspired algorithms in the next chapter, where we will explore their application to constrained problems in greater detail.

Chapter II
Bio-Inspired Algorithms

Chapter II
Bio-Inspired Algorithms

II.1 Introduction

This chapter delves into the fundamental principles and applications of Bio-Inspired algorithms, exploring their emergence as robust tools for solving complex optimization problems, particularly in constrained environments. Building on the theoretical foundation established in the previous chapter, this section provides a comprehensive overview of the various Bio-Inspired techniques that have gained prominence in recent years.

Bio-Inspired algorithms, which draw inspiration from biological processes such as evolution, swarming behaviors, and natural selection, have proven to be highly effective in addressing optimization challenges where traditional methods often fall short. Their inherent adaptability and ability to handle large, complex search spaces make them particularly suitable for constrained optimization problems, where satisfying constraints while optimizing the objective function presents a significant challenge.

In this chapter, we will first explore the key concepts and mechanisms that underpin Bio-Inspired algorithms, focusing on their unique strengths in balancing exploration and exploitation, handling diverse constraints, and maintaining robustness in uncertain environments. We will then delve into specific algorithms, such as genetic algorithms (GAs), particle swarm optimization (PSO), ant colony optimization (ACO), etc, and particularly differential evolution (DE) and grey wolf optimizer (GWO), which will be the primary focus of the subsequent chapters.

By examining both the theoretical and practical aspects of these algorithms, this chapter sets the stage for the development and application of novel strategies that improve constraint handling in optimization processes. The insights gained here will form the basis for understanding the contributions made in this research, where advanced versions of differential evolution will be employed to address the complexities of constrained

optimization. Moreover, the application of GWO algorithm on real-world constrained optimization problem.

II.2 Stochastic and Deterministic Approaches

Stochastic optimization algorithms, which incorporate randomness in their search processes, provide a flexible and robust approach to exploring complex search spaces (Liberti and Kucherenko, 2005). This randomness enables them to avoid the common pitfall of deterministic algorithms: getting stuck in local optima. Deterministic algorithms follow a predefined path and, while predictable, often suffer from search stagnation, especially in high-dimensional or complex landscapes where finding the global optimum is challenging.

In contrast, bio-inspired algorithms such as genetic algorithms, particle swarm optimization, and differential evolution are a subset of stochastic optimization techniques. They utilize stochastic operators inspired by natural processes to explore search spaces more freely. By mimicking evolution, social behavior, or other biological phenomena, these algorithms combine randomness with structured search strategies, increasing their ability to escape local optima and find more globally optimal solutions.

This comparison highlights the advantages of bio-inspired algorithms in constrained optimization. While deterministic algorithms may offer theoretical guarantees under certain conditions, bio-inspired approaches benefit from their stochastic nature, allowing them to adapt more efficiently to complex, real-world problems where the landscape is dynamic or uncertain.

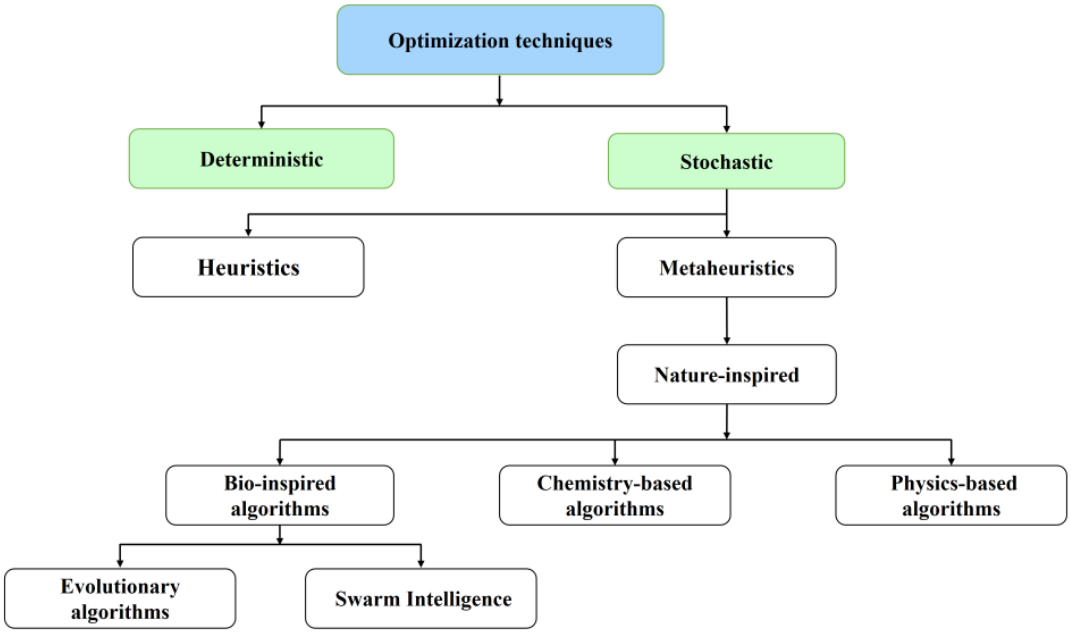


Figure II.1: Resolution method

II.3 Overview of Bio-Inspired Algorithms

Bio-inspired algorithms have emerged as a powerful approach to optimization by drawing inspiration from the biological processes observed in nature (Kar, 2016). These algorithms replicate the behaviors of living organisms, converting them into computational models that can effectively solve complex problems. In nature, species collaborate and interact to perform critical tasks like survival, hunting, defense, and foraging. By studying the behavior of biological systems such as fish schools, ant colonies, and bird flocks, researchers have discovered how these systems find optimal solutions in their environment. This has led to the creation of optimization techniques that mimic nature's problem-solving abilities, refined over millions of years.

Bio-inspired algorithms are not limited to mimicking animal behavior but also include mechanisms based on broader natural processes, such as biological evolution. Evolutionary algorithms, for instance, simulate natural selection and genetic diversity to explore solutions effectively. These varied biological processes have proven to be highly adaptable for solving a wide range of optimization problems in computational contexts.

As population-based metaheuristics, bio-inspired algorithms often employ principles like self-organization, coevolution, and learning to produce high-quality results. These algorithms, which represent the most prominent category of bio-inspired techniques, leverage the unique characteristics of biological systems to tackle complex computational tasks.

Population-based algorithms evolve a set of initial solutions over successive iterations by balancing two critical processes: exploration (diversification) and exploitation (intensification). These processes work together to navigate the search space and converge on optimal solutions (Črepinšek et al, 2013). Exploration involves probing various regions of the search space to discover promising areas that may contain the global optimum. During this phase, the algorithm makes significant or frequent changes to candidate solutions to ensure a broad search across different areas. The primary goal is to identify regions that hold potential, rather than to fine-tune any specific solution. For example, in Particle Swarm Optimization (PSO), the inertia weight encourages particles to maintain their current direction, thereby promoting exploration. Similarly, in Genetic Algorithms (GA), a high mutation rate introduces random variations, ensuring the population explores diverse regions of the search space. Exploitation, on the other hand, focuses on refining and improving the best solutions identified during the exploration phase. This phase involves making smaller, more precise adjustments to candidate solutions, honing in on the optimal solution within a promising region. The objective here is to intensify the search around the most successful solutions, maximizing the chances of finding the global optimum. In PSO, this is achieved by reducing the inertia weight, which increases the tendency of particles to move toward the best-known solutions. In GA, crossover plays a key role in exploitation by combining the traits of the best individuals, leading to in-

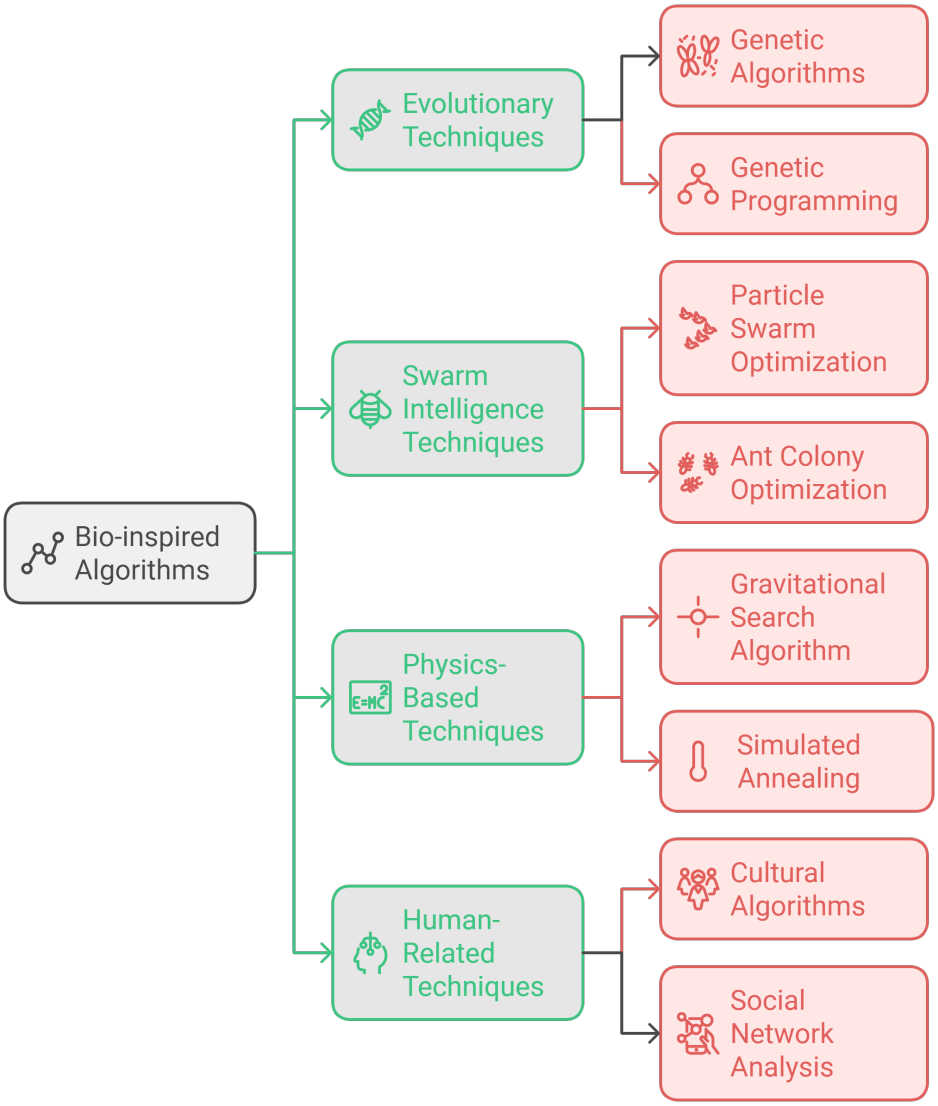


Figure II.2: Taxonomy of nature-inspired algorithms

cremental improvements and convergence on an optimal solution. Balancing exploration and exploitation are essential for the effectiveness of population-based algorithms. While exploration prevents premature convergence by ensuring the search space is adequately covered, exploitation ensures that the algorithm can effectively refine and optimize the best solutions found. This dynamic interplay allows these algorithms to solve complex optimization problems efficiently.

Bio-inspired algorithms are classified into two dominant classes (Fan et al, 2020): evolutionary algorithms and swarm intelligence algorithms Fig.II.2. In the following paragraphs, we will explore the different characteristics of each class, and we will give a detailed overview of their working principles.

II.4 Evolutionary Algorithms

Evolutionary algorithms (EAs) represent a key subclass of evolutionary computation, situated within the broader category of stochastic algorithms. These algorithms have a rich history and are widely recognized for their effectiveness in solving diverse optimization problems (Yu and Gen, 2010). EAs draw inspiration from the principles of natural evolution, including processes such as reproduction, mutation, and selection. This biological metaphor enables EAs to tackle complex, multi-dimensional, nonlinear, and discrete problems without necessitating an in-depth understanding of the underlying mathematical structures. The process begins with the generation of one or more initial solutions within the problem's search space. EAs operate on a population of individuals, each corresponding to a potential solution to the given problem. The solutions created in this initial phase are often referred to as the set of candidate solutions or initial random guesses, as they are generated using random operators. Once these candidate solutions are established, EAs work iteratively to enhance their quality, employing techniques that mimic the evolutionary processes of nature. This iterative improvement continues until a specified termination condition is met. One of the standout advantages of evolutionary algorithms is their simplicity and versatility. They are problem-independent, making them applicable to a wide range of optimization scenarios. Additionally, EAs are known for their ability to avoid local optima, a common pitfall in optimization tasks. However, the effective deployment of EAs requires careful specification of algorithmic parameters, as their performance can be significantly influenced by these values. Furthermore, EAs often entail substantial computational overhead, which can be a limiting factor in large-scale applications. The family of evolutionary algorithms encompasses several well-known optimization techniques, including Genetic Algorithm (GA), Differential Evolution (DE), Evolutionary Strategies (ES), and Evolutionary Programming (EP). In the subsequent sections, we will delve into the details of two of the most prominent algorithms in the literature: Genetic Algorithm (GA) and Differential Evolution (DE), exploring their unique characteristics and applications.

II.4.1 Genetic Algorithm

Genetic Algorithms (GAs) are among the most widely utilized optimization techniques within the family of evolutionary algorithms. First introduced to the scientific community in 1975 by John Holland, GAs are population-based algorithms that emulate the principles of natural selection and evolution as articulated by Darwin (Kramer and Kramer, 2017). These principles include selection, recombination (or crossover), and mutation of genes, enabling GAs to effectively explore complex solution spaces.

In a Genetic Algorithm, each candidate solution is represented as a chromosome,

which is composed of a series of genes. Typically, these genes are encoded as binary strings, where each gene can take a value of either one or zero. This binary representation simplifies the manipulation of candidate solutions and allows for straightforward genetic operations. The fundamental concept of GAs lies in the idea of survival of the fittest: the algorithm strives to evolve the fittest chromosomes over successive generations through the application of reproductive, mutational, and selection operators.

The optimization process begins with the random generation of an initial population of candidate solutions. Each chromosome in this population represents a potential solution to the problem at hand. Once the initial population is established, each chromosome is evaluated using an objective function, which quantifies the quality of the solution it represents. This evaluation forms the basis for the selection process, where the algorithm identifies the fittest individuals to contribute to the next generation.

To facilitate genetic diversity and improve the overall quality of solutions, various selection mechanisms are employed, including:

- **Roulette Wheel Selection:** Probability-based selection where individuals are chosen in proportion to their fitness.
- **Tournament Selection:** A subset of individuals is randomly selected, and the fittest among them is chosen for reproduction.
- **Rank Selection:** Individuals are ranked based on their fitness, and selection is performed according to their rank rather than their absolute fitness values.

Once the selection process is complete, the chromosomes of the selected parents are combined to generate offspring through the crossover operator. This crossover process can take various forms, including:

- **Single-Point Crossover:** A single crossover point is chosen, and the genetic material from the parents is exchanged at this point to create two offspring.
- **Double-Point Crossover:** Two crossover points are selected, leading to a more complex exchange of genetic material between parents.

Following crossover, the newly created chromosomes undergo a mutation process, where one or more genes are randomly altered. The mutation operator introduces variability into the population, preventing premature convergence and helping the algorithm escape local optima. This mechanism is crucial for maintaining diversity within the population, allowing for a more thorough exploration of the solution space.

The iterative application of selection, crossover, and mutation continues until a predefined termination condition is met—such as a maximum number of generations or a satisfactory level of solution quality. Upon termination, the Genetic Algorithm returns

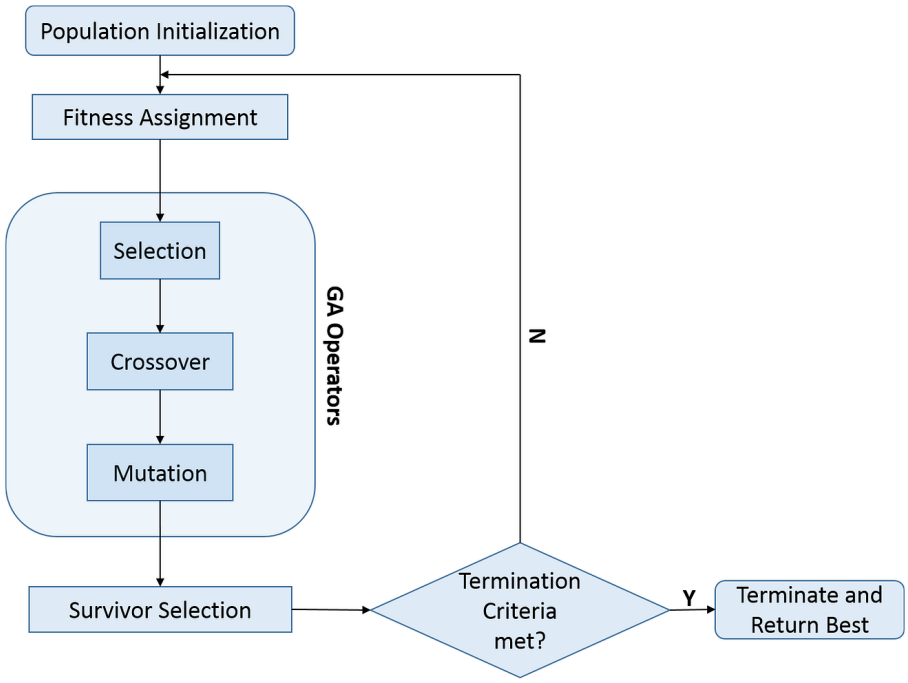


Figure II.3: GA Flowchart

the best chromosome from the final population, which serves as the best approximation of the global optimum for the given problem.

II.4.2 Evolution Strategies

Evolution Strategies (ESs) were developed by Schwefel (1965); Rechenberg (1973). These strategies are centered around the concept of intermediate-sized mutations, which are typically modeled using multivariate normal (Gaussian) distributions. In ESs, a parent individual at generation t, denoted as m_t, generates its k-th offspring $\mathbf{x}_k \in \mathbb{R}^n$ according to:

$$\mathbf{x}_k \sim \mathcal{N}(\mathbf{m}, \sigma^2\mathbf{C}) = \mathbf{m} + \sigma \cdot \mathcal{N}(0, \mathbf{C}) \tag{II.1}$$

where $\mathbf{C} \in \mathbb{R}^{n\times n}$ is a positive definite covariance matrix, and σ is the mutation step size. The parent $\mathbf{m}$ serves as the mean of the current mutation distribution.

In each iteration of the $(\mu + \lambda)$-ES, a set of λ offspring is generated from μ parent individuals. In the "plus" selection strategy, denoted as $(\mu + \lambda)$, the best μ individuals are selected from the combined pool of μ parents and λ offspring to become the parents of the next generation. In contrast, in the "comma" selection strategy, denoted as (μ, λ), the parent population does not carry over to the next generation, and only the best μ offspring out of λ are chosen to continue.

Research on ES primarily focuses on developing methods to adapt the covariance

matrix $\mathbf{C}$ and, more importantly, the step size σ during the optimization process. The goal is to find an optimal way to parameterize this adaptation. For a detailed and comprehensive review of Evolution Strategies, readers are referred to (Hansen et al, 2015).

II.4.3 Covariance Matrix Adaptation Evolution Strategy

The Covariance Matrix Adaptation Evolution Strategy (CMA-ES) was introduced by Hansen and Ostermeier Hansen and Ostermeier (2001) and has become a widely-used standard for continuous black-box optimization. The key advantage of CMA-ES over other evolution strategies lies in its use of correlated mutations rather than axis-aligned ones. By adapting the covariance matrix $\mathbf{C}$, CMA-ES learns an effective mutation distribution, enhancing the likelihood of repeating successful search directions.

In the $(\mu/\mu_w, \lambda)$-CMA-ES, the step size σ is adapted using a procedure similar to CSA-ES and is controlled by the evolution path p_σ. However, in CMA-ES, successful steps are tracked within a coordinate system defined by the principal components of $\mathbf{C}$. This allows a transformation of sampled points $\mathbf{x}_k$ back into the sampling space, using the inverse square root of the covariance matrix, denoted as $\mathbf{C}^{-\frac{1}{2}}$.

The covariance matrix update consists of two key components: a rank-one update and a rank-μ update. The rank-one update tracks the evolution path p_c of successful moves of the mean, $\mathbf{m}_{t+1} - \mathbf{m}_t$, and is adjusted similarly to the evolution path p_σ. A rank-μ update then uses weighted covariances of the top μ individuals to adjust the covariance matrix $\mathbf{C}$, accumulating information from these successful steps. Additionally, in the weighted active CMA-ES variant, information from unsuccessful steps is incorporated through a rank-μ update with a negative weight (Hansen and Ros, 2010). The coefficients c_1, c_μ, and c_- are chosen such that their sum remains below or equal to one to maintain stability in the adaptation process. Although the optimal parameter settings for CMA-ES remain an open question, most parameters are chosen based on empirical observations and the robustness of the algorithm in solving benchmark functions like the Sphere and rotated Ellipsoid functions (Hansen et al, 2015).

In this thesis, particularly in Chapter III, we utilize Covariance Matrix Adaptation as the baseline optimization algorithm. This approach serves as a foundational element for developing our proposed coordinate system, with the goal of enhancing optimization techniques in single-constrained scenarios. Our focus is on improving constraint handling and overall algorithm performance to achieve more effective solutions.

II.4.4 Differential Evolution Algorithm

Differential Evolution (DE) is recognized as one of the most effective evolutionary computing techniques. Since its introduction by Storn and Price (1997), DE has become a preferred optimization method across a diverse range of applications in various scientific

and engineering fields. The core operating principle of DE is akin to that of standard evolutionary algorithms ; however, it uniquely employs the differences between parameter vectors to explore the objective function landscape more effectively.

Similar to Genetic Algorithms, DE seeks to identify a global optimum solution within a D-dimensional search space. The optimization process commences with the random generation of an initial population, which consists of several candidate solutions known as parameter vectors or genomes. Each vector represents a potential solution to the optimization problem.

1. Initialization

The algorithm starts by initializing a population P^0 of N_p candidate solutions (individuals). Each individual x_i^0 is a D-dimensional vector, where D is the number of decision variables. The initial population is generated randomly within the predefined bounds for each dimension:

$$P^0 = \{x_1^0, x_2^0, \ldots, x_{N_p}^0\} \tag{II.2}$$

2. Mutation

The mutation process is a pivotal step in DE, where new candidate solutions, referred to as mutant vectors, are generated. For each individual in the population, a mutant vector is computed using various mutation strategies. Some common mutation strategies include:

- **DE/rand/1:**

$$\mathbf{v}_i^t = \mathbf{x}_{r_1}^t + F \cdot (\mathbf{x}_{r_2}^t - \mathbf{x}_{r_3}^t) \tag{II.3}$$

- **DE/rand/2:**

$$\mathbf{v}_i^t = \mathbf{x}_{rand_1}^t + F \cdot (\mathbf{x}_{r_2}^t - \mathbf{x}_{r_3}^t) + F \cdot (\mathbf{x}_{r_4}^t - \mathbf{x}_{r_5}^t) \tag{II.4}$$

- **DE/rand-to-best/1:**

$$\mathbf{v}_i^t = \mathbf{x}_{r_1}^t + F \cdot (\mathbf{x}_{best}^t - \mathbf{x}_{r_1}^t) + F \cdot (\mathbf{x}_{r_2}^t - \mathbf{x}_{r_3}^t) \tag{II.5}$$

- **DE/current-to-best/1:**

$$\mathbf{v}_i^t = \mathbf{x}_i^t + F \cdot (\mathbf{x}_{best}^t - \mathbf{x}_i^t) + F \cdot (\mathbf{x}_{r_1}^t - \mathbf{x}_{r_2}^t) \tag{II.6}$$

- **DE/current-to-rand/1:**

$$\mathbf{v}_i^t = \mathbf{x}_i^t + F \cdot (\mathbf{x}_{r_1}^t - \mathbf{x}_i^t) + F \cdot (\mathbf{x}_{r_2}^t - \mathbf{x}_{r_3}^t) \tag{II.7}$$

Here, $\mathbf{x}_{best}^t$ denotes the best individual in the current population, $\mathbf{x}_i^t$ represents the current individual, and $\mathbf{x}_{r_k}^t$ are randomly chosen individuals. F is the scaling factor that controls the mutation step size. These mutation strategies significantly impact the algorithm's performance on optimization problems.

3. Crossover

The mutant vector is combined with the target individual to yield a trial vector through a crossover operation. The crossover process is defined as follows:

$$u_{i,j}^t = \begin{cases} v_{i,j}^t & \text{if } \text{rand}_{i,j} \in (0,1) \leq CR \\ x_{i,j}^t & \text{otherwise} \end{cases} \tag{II.8}$$

Here, $\mathbf{x}_{i,j}^t$ and $\vec{v}_{i,j}^t$ are the j-th dimensions of $\vec{x}_i^t$ and $\mathbf{v}_i^t$ respectively, j_{rand} is a random number uniformly generated between 0 and 1, CR denotes the crossover probability, and rand generates a random value between 0 and 1.

4. Selection

The trial vector competes with the target individual. If the trial vector is superior, it substitutes the target individual in the next generation:

$$\vec{x}_i^t = \begin{cases} \vec{u}_i^t & \text{if } f(\vec{u}_i^t) < f(\vec{x}_i^t) \\ \vec{x}_i^t & \text{otherwise} \end{cases} \tag{II.9}$$

II.5 Swarm Intelligence (SI)

Swarm Intelligence (SI) was first introduced to the field of optimization in 1993 when researchers leveraged the collective behavior of swarms to develop cellular robotic systems. SI represents a groundbreaking paradigm within artificial intelligence, particularly for solving complex optimization problems (Kennedy, 2006). It has opened up a novel and promising direction in optimization research by mimicking the intelligent behaviors observed in biological swarms, such as ant colonies, bee hives, bird flocking, and fish schooling.

In nature, these species demonstrate an ability to co-evolve and cooperate through local interactions with each other and their environment. The individuals within a swarm, often referred to as search agents, operate under simple rules and interact locally, without any centralized control mechanism. Despite the simplicity of their individual behaviors, these interactions can lead to the emergence of complex global patterns and solutions—patterns that are often far beyond the capabilities of any single agent.

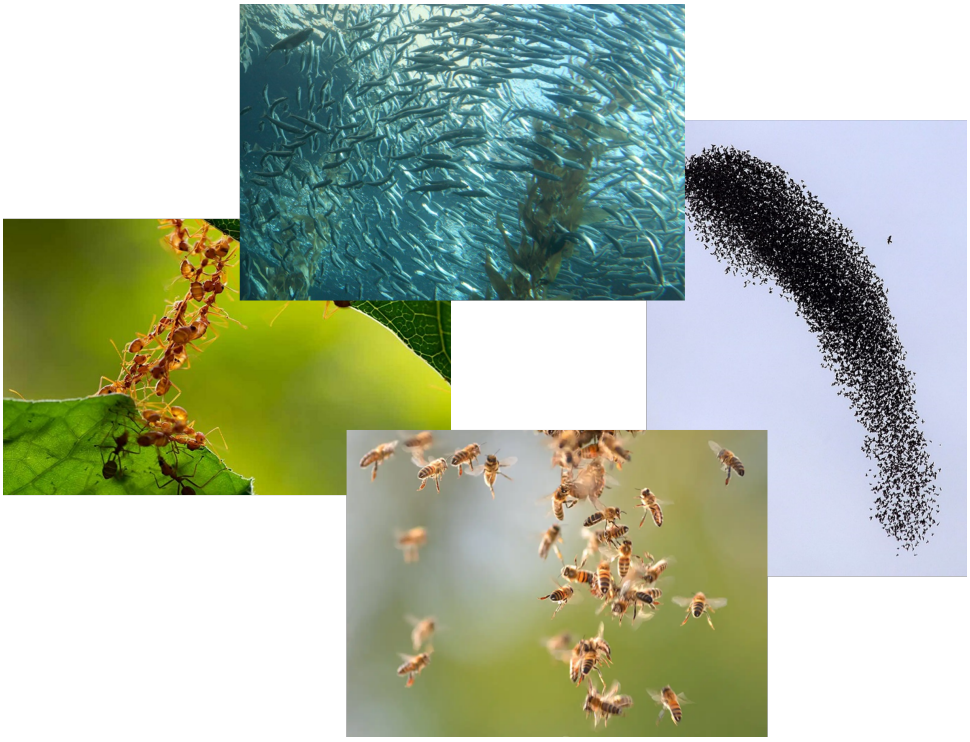


Figure II.4: Swarm intelligence examples

Swarm Intelligence, as defined by Kennedy in 2006, refers to a problem-solving capability that emerges from the interactions of simple information-processing units. The term ”swarm” implies multiplicity, stochasticity, randomness, and an inherent messiness, while ”intelligence” suggests that these problem-solving methods are successful in achieving their goals. The processing units within a swarm can vary widely, from insects, birds, and humans to mechanical devices, computational elements, or even abstract mathematical entities. Despite this diversity, the key characteristic of swarm intelligence is the interaction among these units, which collectively contribute to the problem-solving process.

This concept has inspired researchers to develop a variety of optimization techniques by imitating the social behaviors observed in different animal societies Fig.II.4. To harness this intelligence, researchers focus on understanding the local rules governing interactions within the swarm, which collectively lead to the emergence of social intelligence. As a result, swarm intelligence methods have gained recognition as a powerful alternative to traditional deterministic techniques, often demonstrating superior performance in solving complex optimization problems.

Swarm intelligence algorithms are a subset of bio-inspired algorithms, specifically population-based metaheuristics, where a population of agents is maintained and evolved during the optimization process. The success of these algorithms largely stems from the shared information among the agents, which facilitates self-organization—leading to the emergence of higher-level structures and solutions.

Several factors contribute to the growing popularity of SI-based algorithms:

1. **Ease of Implementation:** SI-based algorithms are generally straightforward to implement, making them accessible for a wide range of applications.
2. **Minimal Parameter Tuning:** Most SI-based algorithms require relatively few parameters to be adjusted, simplifying the optimization process.
3. **Memory Utilization:** These algorithms often incorporate memory to retain the best solutions found throughout the optimization process, enhancing their effectiveness.
4. **Simple Operators:** The simplicity of the operators in SI algorithms enables them to approximate the global optimum efficiently and within a reasonable timeframe.

Due to these advantages, SI-based algorithms have become a favored choice in optimization, offering a robust and flexible approach to solving complex, real-world problems.

II.5.1 Particular Swarm Optimization

Particle Swarm Optimization (PSO) is a swarm intelligence algorithm inspired by the observation of swarm behaviors in ecological systems (Kennedy and Eberhart, 1995). The classical PSO algorithm was first proposed by Kennedy in 1995. In PSO, solutions are represented as particles, each holding two vectors: the position vector and the velocity vector, which represent the particle's evolutionary state in the search space. For a problem in a D-dimensional space, particle i holds the position vector $\vec{X}_i$ and the velocity vector $\vec{V}_i$:

$$\vec{X}_i = (x_i^1, x_i^2, \ldots, x_i^D) \tag{II.10}$$

$$\vec{V}_i = (v_i^1, v_i^2, \ldots, v_i^D). \tag{II.11}$$

During each iteration of the algorithm, these vectors are updated according to the following rules:

$$\vec{V}_i^{(t+1)} = w \cdot \mathbf{V}_i^{(t)} + c_1 \cdot r_1 \cdot (\mathbf{pbest}_i^{(t)} - \mathbf{X}_i^{(t)}) + c_2 \cdot r_2 \cdot (\mathbf{gbest}^{(t)} - \mathbf{X}_i^{(t)}) \tag{II.12}$$

$$\vec{X}_i^{(t+1)} = \mathbf{X}_i^{(t)} + \mathbf{V}_i^{(t)} \tag{II.13}$$

Here, $\mathbf{pbest}_i$ is the personal best position discovered by the i^{th} particle, and $\mathbf{gbest}$ is the best so far position found by the entire swarm. c_1 and c_2 are acceleration coefficients

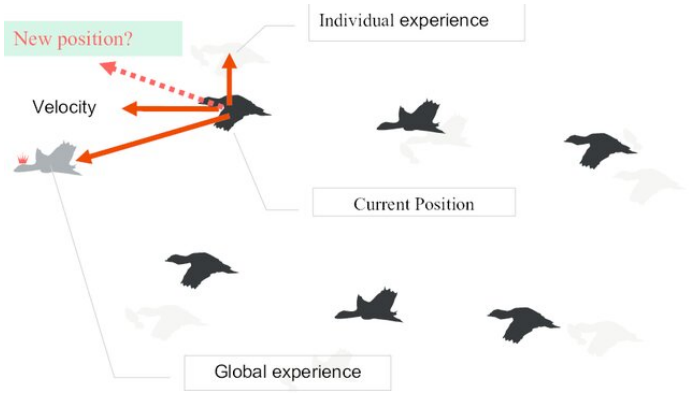


Figure II.5: Schematics of PSO algorithm

that balance personal and social experiences. r_1 and r_2 are random numbers uniformly distributed in the range $[0, 1]$, and w is the inertia weight. Experimental analysis has shown that a higher w promotes exploration, while a smaller w enhances exploitation.

Fig II.5 shows the collective behavior of individuals and the pseudo-code of the basic PSO is presented in algorithm 1

Algorithm 1 Particle Swarm Optimization (PSO)

Initialize the swarm of particles with random positions and velocities
for each particle **do**
    Initialize particle's position randomly in the search space
    Initialize particle's velocity randomly
    Set the particle's personal best position (`pBest`) to its initial position
    if f(`pBest`) $< f$(`gBest`) **then**
        Update `gBest` to particle's position
    end if
end for
while stopping criterion not met **do**
    for each particle **do**
        Update particle's velocity:
        $V_i = w \cdot V_i + c_1 \cdot r_1 \cdot (pBest_i - X_i) + c_2 \cdot r_2 \cdot (gBest_i - X_i)$
        Update particle's position
        $X_i = X_i + V_i$
        Evaluate the fitness of the particle's current position
        if fitness(X_i) $<$ fitness($pBest_i$) **then**
            Update $pBest_i$ to the current position
            if fitness($pBest_i$) $<$ fitness($pBest$) **then**
                Update `gBest` to $pBest_i$
            end if
        end if
    end for
end while
Return the global best position (`gBest`) as the optimal solution

II.5.2 Whale Optimization Algorithm

The Whale Optimization Algorithm (WOA) is a bio-inspired optimization technique that mimics the social hunting behavior of humpback whales, particularly their unique bubble-net feeding strategy. WOA is designed to balance exploration and exploitation by adapting mechanisms based on the whales' natural hunting behaviors.

1. Encircling Prey

Humpback whales have an instinctive ability to detect prey and form an encircling formation around it. In WOA, the optimal solution in the search space is not predetermined; thus, the algorithm assumes that the best candidate solution at any iteration is likely close to the global optimum. The other search agents in the population then update their positions to move closer to this identified best solution. This encircling behavior is represented by the following equations:

$$D = |C \cdot \vec{X}^*(t) - \vec{X}(t)| \tag{II.14}$$

$$\vec{X}(t+1) = \vec{X}^*(t) - A \cdot D \tag{II.15}$$

where:

- t denotes the current iteration,
- A and C are coefficient vectors,
- $\vec{X}^*$ is the position vector of the current best solution,
- $\vec{X}$ is the position vector of an individual whale,

The position of $\vec{X}^*$ is updated in each iteration if a better solution is found. The coefficient vectors A and C are computed as follows:

$$A = 2a \cdot r - a \tag{II.16}$$

$$C = 2 \cdot r \tag{II.17}$$

where:

- a is a parameter that linearly decreases from 2 to 0 over the course of the iterations, balancing exploration and exploitation, and
- r is a random vector within the range $[0, 1]$.

2. Bubble-Net Attacking Method (Exploitation Phase)

To model the bubble-net behavior observed in humpback whales, two approaches are implemented in WOA: the *shrinking encircling mechanism* and the *spiral updating position*. These approaches enable the algorithm to concentrate the search around promising solutions.

- Shrinking Encircling Mechanism: In this approach, as the value of a decreases, the range of A is confined within $[-a, a]$. This restriction allows the search agents to position themselves between their current locations and the best solution in a controlled manner, depending on the value of A. When A is randomly set between $[-1, 1]$, the new position of a search agent can move closer to the current best solution.

- Spiral Updating Position : This approach simulates the helix-shaped movement of humpback whales around their prey. The spiral movement is modeled by calculating the distance between a whale at position $\vec{X}(t)$ and the prey at $\vec{X}^*(t)$ as follows:

$$\vec{X}(t+1) = D' \cdot e^{bl} \cdot \cos(2\pi l) + \vec{X}^*(t) \tag{II.18}$$

  where:

  - $D' = |\vec{X}^*(t) - \vec{X}(t)|$ is the distance between the whale and the prey,
  - b is a constant defining the shape of the logarithmic spiral,
  - l is a random number in the range $[-1, 1]$, and

  To capture the simultaneous shrinking and spiraling behavior, a probability p (50%) is introduced to switch between the shrinking encircling mechanism and the spiral model. The position update rule becomes:

$$\vec{X}(t+1) = \begin{cases} \vec{X}^*(t) - A \cdot D & \text{if } p < 0.5 \\ D' \cdot e^{bl} \cdot \cos(2\pi l) + \vec{X}^*(t) & \text{if } p \geq 0.5 \end{cases} \tag{II.19}$$

  where p is a randomly generated number within $[0, 1]$.

3. Search for Prey (Exploration Phase)

In the exploration phase of the Whale Optimization Algorithm (WOA), humpback whales search for prey by moving towards diverse areas in the search space, inspired by the behavior of whales observing the positions of others in their group. This phase encourages global search and helps the algorithm avoid local optima.

To achieve exploration, the same variation of the A vector used in the exploitation phase is applied here with adjustments. Specifically, A is assigned random values greater than 1 or less than -1, which forces search agents to move farther away from a reference whale, thereby encouraging them to explore the broader search space. Unlike in the exploitation phase, where search agents are guided towards the best solution found so far, in the exploration phase, each search agent updates its position based on a randomly selected agent from the population. This mechanism, along with setting $|A| > 1$, enhances the algorithm's exploration abilities and enables it to conduct a more global search. The equations governing this behavior are:

$$D = |C \cdot \vec{X}_{\text{rand}} - \vec{X}| \tag{II.20}$$

$$\vec{X}(t+1) = \vec{X}_{\text{rand}} - A \cdot D \tag{II.21}$$

where:

- $\vec{X}_{\text{rand}}$ represents the position vector of a randomly chosen whale from the current population, and
- D denotes the distance between the current whale's position $\vec{X}$ and the randomly selected whale's position $\vec{X}_{\text{rand}}$.

II.5.3 Ant Colony Optimization (ACO)

Ant Colony Optimization (ACO) is a prominent algorithm in the realm of swarm intelligence, first introduced by Marco Dorigo and his colleagues in 1996. The algorithm is inspired by the foraging behavior of ants, particularly their method of finding and optimizing paths to food sources (Dorigo et al, 2006).

In the natural world, ants exhibit a remarkable ability to discover the most efficient routes to food sources Fig.II.6. This is achieved through the use of a chemical substance known as pheromone, which ants deposit along their paths. When an ant finds food, it returns to the nest, laying down a trail of pheromone proportional to the quality of the food. This pheromone trail serves as a guide for other ants in the colony.

As other ants encounter these pheromone trails, they are more likely to follow paths with stronger pheromone concentrations, thereby reinforcing those trails. Over time, as more ants traverse a path and lay down additional pheromones, the colony collectively converges on the most efficient route—usually the shortest path. The longer paths are gradually abandoned because the pheromone on these paths evaporates more quickly, leading to lower pheromone concentrations compared to shorter, more frequently traveled routes.

The ACO algorithm models this natural behavior to solve optimization problems. In the algorithm, each ant represents a potential solution. As ants move from one point to

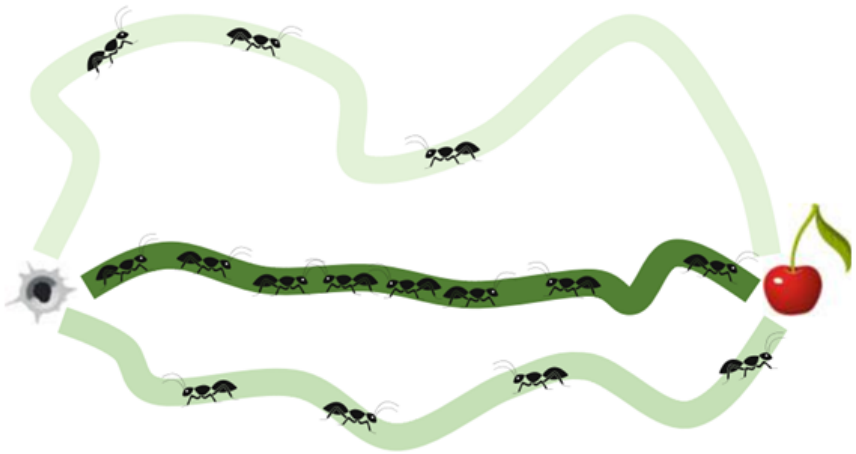


Figure II.6: Ant colony: Foraging behavior (Dorigo et al, 2006)

another in the search space, they choose their next step in a probabilistic manner based on a transition rule that considers both the pheromone level on the connecting edge and a heuristic value associated with that path.

1. Transition Probability

The transition probability for an ant to move from point i to point j is calculated using the following formula:

$$p_{i,j} = \frac{(\tau_{i,j})^{\alpha}(\eta_{i,j})^{\beta}}{\sum_k (\tau_{i,k})^{\alpha}(\eta_{i,k})^{\beta}} \tag{II.22}$$

Here, $\tau_{i,j}$ represents the intensity of the pheromone on the edge between points i and j, while $\eta_{i,j}$ is the heuristic value of the path. The parameters α and β determine the relative influence of the pheromone trail and the heuristic information on the ant's decision.

2. Pheromone Update

Pheromone levels are not static; they decrease over time through a process called evaporation, governed by an evaporation coefficient ρ where $0 < \rho < 1$. This evaporation helps to prevent the algorithm from converging too quickly on suboptimal solutions. After each iteration, the pheromone levels on each path are updated according to the following rule:

$$\tau_{i,j} = (1 - \rho)\tau_{i,j} + \Delta\tau_{i,j} \tag{II.23}$$

The term $\Delta\tau_{i,j}$ represents the amount of pheromone deposited by all the ants that have traversed the edge (i, j) in the current iteration. The sum of these deposits influences the next generation of ants, guiding them towards potentially better solutions.

II.5.4 The Bees Algorithm

The Bees Algorithm (BA) was introduced by Pham et al (2006) as a bio-inspired optimization technique, emulating the foraging behavior of honeybees in their search for food sources. In BA, bees are divided into two key groups: scout bees and forager bees. Scouts are tasked with exploring new areas to locate rich food sources. They begin by randomly navigating the surroundings of the hive in search of promising flower patches. Upon discovering a valuable food source, a scout memorizes its location and returns to the hive to share this information with the foragers. This communication occurs through a unique behavior known as the *waggle dance*, which effectively conveys the location of the discovered food source. The more promising patches, rich in nectar (indicating higher fitness), attract a larger number of foragers. Subsequently, the scout returns to the identified flower patch, now accompanied by recruited foragers to gather nectar. Upon their return to the hive, foragers may also perform the waggle dance, directing others toward the identified flower patch.

The artificial BA begins by randomly placing n_s scout bees within the search space. Each scout's position is then evaluated using a fitness function. During each iteration, the n_b scouts that discover the highest fitness solutions perform the waggle dance to recruit foragers. Among these, the top-rated n_e solutions attract the largest number of

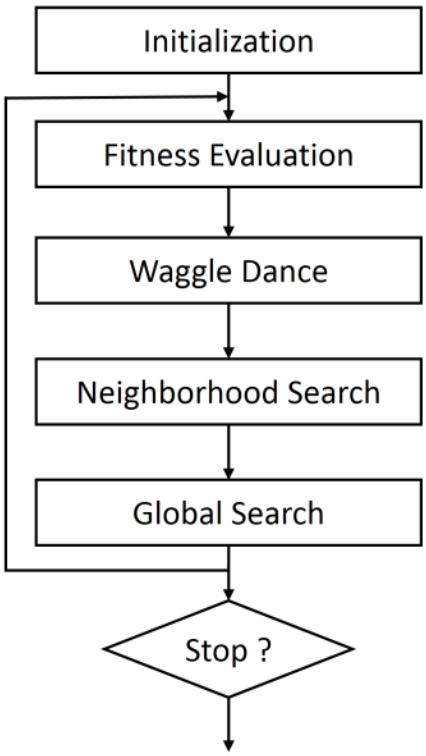


Figure II.7: BA flowchart

foragers (n_{re}). Meanwhile, the remaining scout bees recruit fewer foragers (n_{rb}), with $n_{rb} < n_{re}$. This recruitment strategy ensures that a greater number of bees focus on the most promising areas in the search space.

Forager bees are distributed randomly within the neighborhood of the identified flower patches. A flower patch represents a promising region, defined by the size of its neighborhood (ngh). If a forager discovers a solution in the neighborhood that outperforms the one found by the scout, the forager assumes the role of the new scout, taking over the waggle dance in the next iteration. The remaining scout bees ($n_s - n_b$), which have located less promising food sources, are reallocated randomly to explore new regions within the search space. These steps are repeated until an optimal solution is reached or a predefined maximum number of iterations is completed. A visual representation of the BA's workflow is provided in Figure II.7.

II.5.5 Grey Wolf Optimization Algorithm

Inspired by the social hierarchy and hunting behavior of grey wolves (Fig.II.9), the Grey Wolf Optimization (GWO) algorithm was developed by Mirjalili et al. in 2014. GWO mimics the leadership and hunting strategies of grey wolves to solve optimization problems effectively (Mirjalili et al, 2014). The algorithm selects three leaders in each iteration: the fittest wolf, denoted as X_α, the second-best X_β, and the third X_σ. The remaining wolves, represented as X_ω, follow these leaders during the hunting process.

Figure II.8: Grey wolves in nature (Mirjalili et al, 2014)

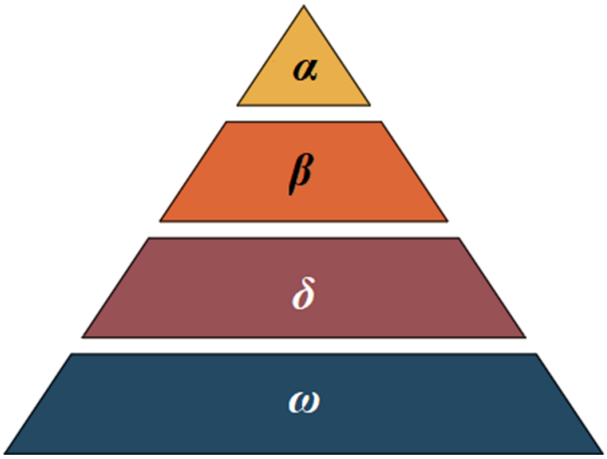


Figure II.9: Grey wolf hierarchy (Mirjalili et al, 2014)

1. Encircling

The encircling process of grey wolves during the hunt is calculated from the following mathematical equations:

$$\vec{D} = \left|\vec{C} \cdot \vec{X}_p(t) - \vec{X}(t)\right| \tag{II.24}$$

$$\vec{X}(t+1) = \vec{X}_p(t) - \vec{A} \cdot \vec{D} \tag{II.25}$$

Where t presents the iteration number, $\vec{x}$ and $\vec{x_p}$ and indicates the positions vector of a grey wolf and the prey respectively. Here, a coefficient vectors $\vec{A}$ and $\vec{C}$ calculated as follows:

$$\vec{A} = 2\vec{a} \cdot \vec{r}_1 - \vec{a} \tag{II.26}$$

$$\vec{C} = 2 \cdot \vec{r}_2 \tag{II.27}$$

2. Hunting

In order to mathematically simulate the hunting behavior of grey wolves, it is assumed that the three best candidate solutions alpha, beta, and delta have better knowledge about the potential location of prey. Therefore, the remaining wolves can update their positions according to the following expressions:

$$\begin{aligned} \vec{D}_\alpha &= \left|\vec{C}_1 \cdot \vec{X}_\alpha - \vec{X}\right| \\ \vec{D}_\beta &= \left|\vec{C}_2 \cdot \vec{X}_\beta - \vec{X}\right| \\ \vec{D}_\delta &= \left|\vec{C}_3 \cdot \vec{X}_\delta - \vec{X}\right| \end{aligned} \tag{II.28}$$

$$\begin{aligned}\vec{X}_1 &= \vec{X}_\alpha - \vec{A} \cdot \left(\vec{D}_\alpha\right) \\ \vec{X}_2 &= \vec{X}_\beta - \vec{A}_2 \cdot \left(\vec{D}_\beta\right) \\ \vec{X}_3 &= \vec{X}_\delta - \vec{A}_3 \cdot \left(\vec{D}_\delta\right)\end{aligned} \tag{II.29}$$

$$\vec{X}(t+1) = \frac{\vec{X}_1 + \vec{X}_2 + \vec{X}_3}{3} \tag{II.30}$$

Where D_α, D_β, and D_δ are the estimated distance between grey wolf and the three best solutions X_α, X_β, and X_δ respectively at t-th iteration Eqs. II.28 and II.29 . $\vec{A}1$, $\vec{A}2$, and $\vec{A}3$ are calculated as in Eq. II.26, $\vec{C}1$, $\vec{C}2$, and $\vec{C}3$ are calculated as in Eq. II.27.

3. Attacking

When the prey stops moving indicates that the hunt has finished and wolves the attacking process begins. This process can mathematically model by the value $\vec{a}$ which is linearly decreased from 2 to 0 over the course of iterations controlling the exploration and exploitation. In this phase, the search agents take their new positions randomly between their current position and the position of the prey.

Algorithm 2 The Basic GWO

```
Initialize population of wolves x_i(i = 1, 2, ..., n)
Initialize the parameters a, A and C
calculate the fitness of each wolf
X_α = the best search agent
X_β = the second best search agent
X_δ = the third best search agent
while t ≥ MaxIter do
    for each search agent do
        update the position of current wolf
    update the values of a, A and C
    calculate the fitness for all search agent
    update best wolves X_α, X_β and X_δ
    t = t + 1
end while
return X_α
```

The Grey Wolf Optimization algorithm has several notable advantages:

- **Simplicity**: GWO is easy to implement due to its straightforward mathematical formulations.
- **Fewer Parameters**: The algorithm requires fewer parameters to be tuned compared to other optimization methods, such as Genetic Algorithms or Particle Swarm Optimization.

- **Exploration and Exploitation Balance**: The dynamic adjustment of A allows GWO to effectively balance exploration and exploitation throughout the optimization process.
- **Flexibility**: GWO can be adapted for various optimization problems, including continuous and discrete optimization tasks.

GWO has been successfully applied in various domains, including engineering design, feature selection, machine learning, and more (Makhadmeh et al, 2023). Its ability to converge quickly and find high-quality solutions makes it a powerful tool in the optimization toolbox.

In Chapter IV, we will explore the application of the Grey Wolf Optimization algorithm in solving the Traveling Salesman Problem (TSP), where the unique properties of GWO will be leveraged to improve solution quality and computational efficiency.

II.5.6 Other bio-inspired algorithms

The literature provides a significant number of bio-inspired. The number of the algorithms is high to the extent that it is not possible to list all the existing algorithms in one table. Therefore, the most popular and recent algorithms found in the literature are listed in the following Table II.1:

Table II.1: Bio-inspired algorithms

Algorithms	**References**
Adaptive Social Behavior Optimization (ASBO)	Singh (2012)
African Wild Dog Algorithm (AWDA)	Subramanian et al (2013)
Animal Behavior Hunting (ABH)	Naderi et al (2014)
Animal Migration Optimization Algorithm (AMO)	Li et al (2014b)
Ant Lion Optimizer (ALO)	Mirjalili (2015)
Archerfish Hunting Optimizer (AHO)	Zitouni et al (2022)
Artificial Butterfly Optimization (ABO)	Qi et al (2017)
Artificial Coronary Circulation System (ACCS)	Kaveh and Mohsen (2019)
Artificial Electric Field Algorithm (AEFA)	Yadav et al (2019)

Algorithms	**References**
Artificial Jellyfish Search Optimizer (AJSO)	Chou and Truong (2021)
Artificial Reaction Algorithm (ARA)	Melin et al (2013)
Binary Whale Optimization Algorithm (bWOA)	Reddy K et al (2019)
Biology Migration Algorithm (BMA)	Zhang et al (2019)
Bird Mating Optimization (BMO)	Askarzadeh (2014)
Brain Storm Optimization (BSO)	Shi (2011)
Cat Swarm Optimization (CSO)	Chu et al (2006)
Chicken Swarm Optimization (CSO)	Meng et al (2014)
Circle Search Algorithm (CSA)	Qais et al (2022)
Coronavirus Herd Immunity Optimizer (CHIO)	Al-Betar et al (2021)
Coronavirus mask protection algorithm	Yuan et al (2023)
Electric eel foraging optimization	Zhao et al (2024)
Firefly Algorithm (FA)	Yang (2009)
Flower Pollination Algorithm (FPA)	Yang (2012)
Green anaconda optimization	Dehghani et al (2023)
Golden Ball Algorithm (GB)	Osaba et al (2014)
Gradient-Based Optimizer (GBO)	Ahmadianfar et al (2020)
Harmony Search (HS)	Lee and Geem (2005)
Harris Hawks Optimizer (HHO)	Heidari et al (2019)
Imperialist Competitive Algorithm (ICA)	Atashpaz and Lucas (2007)
Jaya Algorithm (JA)	Rao (2016)
Kril Herd (KH)	Gandomi and Alavi (2012)
Lévy Flight Distribution (LFD)	Houssein et al (2020)
Multi-Verse Optimizer (MVO)	Mirjalili et al (2016a)

Algorithms	References
Newton-Raphson-based optimizer	Sowmya et al (2024)
Pigeon Inspired Optimization (PIO)	Duan and Qiao (2014)
Salp Swarm Algorithm (SSA)	Mirjalili et al (2017)
Simulated Annealing (SA)	Kirkpatrick et al (1983)
Tree Seed Algorithm (TSA)	Kiran (2015)
Water Cycle Algorithm (WCA)	Eskandar et al (2012)

II.6 Constraint-Handling Techniques

The inclusion of constraints in optimization problems significantly increases the complexity of the optimization process, particularly in ensuring that these constraints are satisfied (Mezura-Montes and Coello, 2011). Various challenges inherent in real-world optimization, such as vast search spaces, noise in objective functions, and the intricacy of the modeling process, with constraints being a critical factor. Constraints can shift the focus of the search from optimizing the objective function to merely finding a feasible (i.e., valid) solution.

Bio-inspired algorithms (BIAs), in their original form, were primarily developed to navigate unconstrained search spaces. However, the need to handle constraints has led to the development and integration of specialized constraint-handling techniques into BIAs. These techniques are crucial for directing the search towards regions that contain feasible solutions while maintaining the algorithm's exploratory capabilities.

Constraint-handling techniques for bio-inspired algorithms can be categorized in various ways. Previous reviews by Mezura-Montes and Coello (2011) and Coello (2002) have proposed different classifications for these techniques in the context of nature-inspired algorithms. Based on these works, a simplified taxonomy of the primary categories includes:

1. **Penalty Functions**: Penalty functions transform a constrained optimization problem into an unconstrained one by augmenting the objective function with terms that penalize constraint violations. This approach allows standard optimization algorithms to address constraints indirectly by heavily penalizing infeasible solutions, thereby guiding the search towards feasible regions.

2. **Decoders**: Decoder methods encode solutions in a way that inherently represents

the feasible domain of the problem. By designing the genotype-to-phenotype mapping such that the decoded phenotypes remain within the feasible region, the optimization process is confined to valid solutions, ensuring that all candidates satisfy the problem's constraints.

3. **Special Operators**: Special operators are tailored mechanisms that prioritize feasible solutions over infeasible ones. These operators adjust the search dynamics of the algorithm to promote the exploration and exploitation of feasible regions, thus improving convergence towards optimal solutions within the permissible search space.

4. **Separation Approaches**: Unlike penalty functions, this approach maintains a clear distinction between the objective function and constraints. The optimization process addresses these components separately, allowing for more sophisticated strategies that balance the trade-off between optimizing the objective and satisfying the constraints, often leading to more effective and nuanced solutions.

In the following subsections, we will introduce five common techniques that exemplify these categories.

II.6.1 Penalty Methods

Penalty functions have historically been the most prevalent approach for incorporating constraints into optimization problems, both in evolutionary algorithms and traditional mathematical programming. Originally proposed in the 1940s, these methods have been extensively studied and expanded upon due to their simplicity and effectiveness (Smith et al, 1997).

The fundamental idea behind penalty methods is to modify the fitness landscape by adding a penalty value to the objective function for each violation of the constraints. This alteration discourages the selection of infeasible solutions during the optimization process. In their general form, penalty functions can be expressed as:

$$\psi = f(x) + \sum_{i=0}^{p} r_i \cdot \max(0, g_i(x))^{\alpha} + \sum_{j=1}^{q} c_j \cdot |h_j(x)|^{\alpha} \tag{II.31}$$

where:

- ψ is the modified fitness function to be minimized,
- $f(x)$ is the original objective function,
- $g_i(x)$ and $h_j(x)$ represent inequality and equality constraints respectively,
- r_i and c_j are positive penalty factors,

- α is typically set to 1 or 2.

Although this general implementation is straightforward, it requires tuning multiple penalty parameters corresponding to each constraint, which can be impractical for problems with a large number of constraints. To address this issue, several variations of penalty methods have been developed, including static, dynamic, and adaptive penalty functions.

1. Static Penalty Functions

The static penalty function is the simplest and most widely used form of penalty methods. It consolidates all constraint violations into a single term, reducing the number of parameters that need tuning (Kulkarni et al, 2018). The static penalty function is defined as:

$$\psi = f(x) + r \cdot \phi(x) \tag{II.32}$$

where:

- r is a constant penalty coefficient,
- $\phi(x)$ represents the overall constraint violation calculated as:

$$\phi(x) = \sum_{i=0}^{p} \max(0, g_i(x))^{\alpha} + \sum_{j=1}^{q} |h_j(x)|^{\alpha} \tag{II.33}$$

Setting an appropriate value for r is crucial. If r is too low, the search may favor regions where the objective function is minimized but constraints are violated, leading to infeasible solutions. Conversely, if r is too high, the search prioritizes feasibility over optimality, which can cause premature convergence to local optima, especially in disjointed or highly constrained search spaces. Ideally, r should be just sufficient to ensure feasibility without overly constraining the search, a concept known as the minimum penalty rule.

2. Dynamic Penalty Functions

Dynamic penalty functions address the limitations of static penalties by allowing the penalty coefficient r to vary throughout the evolutionary process (Yoo et al, 2021). The general principle is to use lower penalty values in the early stages to encourage exploration of the search space and higher values in later stages to enforce feasibility. This adaptive adjustment aims to balance exploration and exploitation effectively.

While dynamic penalties can be effective, their performance heavily depends on the chosen schedule for updating r. Designing an appropriate schedule adds complexity and

requires additional parameters, potentially reducing the method's attractiveness compared to static penalties. Moreover, this approach tends to work best when the unconstrained global optimum is close to the constrained global optimum, which may not always be the case.

3. Adaptive Penalty Functions

Adaptive penalty functions further refine the penalty approach by utilizing information gathered during the search process to adjust the penalty coefficient dynamically. These methods aim to automate the tuning of r by responding to the current state of the search, thereby reducing the need for user-defined parameters.

Implementation of adaptive penalties is relatively straightforward, and they can provide a more balanced search by continuously adapting to the landscape of the problem. However, studies have shown that adaptive methods may require a significant number of iterations to converge to the optimal solution (Barbosa et al, 2015).

II.6.2 Stochastic Ranking

Stochastic Ranking (SR), developed by Runarsson and Yao (2000), is a method that balances the importance of the objective function and constraint violations in optimization. Instead of using fixed penalties, SR ranks individuals in the population through a probabilistic process. Two individuals are compared based on their objective function with a certain probability P_f; if not, their ranking is determined by their constraint violations.

Once ranked, the top individuals are selected for recombination, ensuring that both high-performing and feasible solutions contribute to the next generation. This method helps the optimization process focus on improving both the objective function and meeting the constraints simultaneously.

II.6.3 Feasibility Rules

Feasibility rules are a constraint-handling technique that prioritizes feasible solutions over infeasible ones. Unlike penalty functions, which combine information from both constraint violations and the objective function, feasibility rules treat these two aspects separately (Mezura-Montes et al, 2004; Deb, 2000). This approach, also known as lexicographical order, relies on a binary tournament selection process based on the following criteria:

1. Any feasible solution is always preferred over any infeasible solution.
2. Among two feasible solutions, the one with the better objective function value is preferred.

3. Among two infeasible solutions, the one with the smaller constraint violation is preferred.

II.6.4 Epsilon Constrained

To address the challenges associated with feasibility rules in constrained problems, Takahama and Sakai (2010) proposed the ϵ-constrained method for evolutionary algorithms. This method allows a relaxation of constraints to explore constrained regions more effectively. The tolerance level of this relaxation, referred to as the ϵ level, defines the threshold below which solutions are considered feasible. Once feasibility is determined using the ϵ level, the lexicographical order (i.e., Deb's feasibility rules) is employed to select individuals for the next generation. This technique has proven particularly effective in highly constrained problems, such as those involving equality constraints, because the controlled relaxation in the early generations facilitates the exploration of regions that would otherwise be unreachable using strict feasibility rules.

The main drawback of this method is the challenge of setting the ϵ parameter. While the ϵ level allows for effective exploration of the search space during early generations, it is crucial that ϵ eventually reaches 0 during the evolutionary process to ensure feasible solutions. For that a dynamic control of the ϵ level to address this issue:

$$\epsilon = \begin{cases} \epsilon_0 \cdot (1 - t/T)^{c_p} & \text{if } t \leq p \cdot T \\ 0 & \text{otherwise} \end{cases} \tag{II.34}$$

The parameter c_p is intricately computed by:

$$c_p = -\frac{\log(\epsilon_0) + \lambda}{\log(1 - p)} \tag{II.35}$$

Here, ϵ_0 symbolizes the initial threshold, tailored to reflect the maximum degree of constraint violation within the initial population. T denotes the maximum number of generations, with its specific value dictated by the context of the study. The parameter λ contributes to the calculation of c_p, while p governs the degree to which information from the objective function is utilized. This mathematical framework shows the adaptive nature of ϵ-constrained optimization and its ability to dynamically respond to changing population dynamics.

II.7 Conclusion

In this chapter, we have explored various popular bio-inspired algorithms, with a particular focus on those that will be employed in this thesis. By examining their origins, inspirations, and applications, we have established a foundational understanding of each

algorithm, which enables their effective use as benchmark algorithms for the subsequent studies. In the following chapter, we will introduce a methodology that designates Differential Evolution (DE) as the primary algorithm for optimizing constraints. DE is selected due to its robust performance across a wide array of applications and its remarkable adaptability to various constraint types. The upcoming sections will thoroughly investigate the state of the art in Differential Evolution, emphasizing its core principles, recent advancements, and specific strategies for effectively managing constraints. This focused analysis will not only highlight DE's relevance to our research problem but also validate its selection as the methodological cornerstone of this study. Additionally, in Chapter IV, we will discuss an application of the Grey Wolf Optimizer (GWO) to the well-known Traveling Salesman Problem (TSP), which serves as a classic example of a constrained optimization problem. This application will further illustrate the diverse capabilities of bio-inspired algorithms in addressing complex optimization challenges. Having explored the broader category of bio-inspired algorithms and their inherent strengths in addressing complex optimization problems, it is evident that these nature-inspired methods provide a rich and versatile toolkit for tackling constrained optimization challenges. Among these algorithms, Differential Evolution (DE) distinguishes itself as one of the most effective and versatile approaches available. Its innovative mechanisms for generating new solutions through mutation, crossover, and selection make DE particularly adept at optimizing problems where traditional methods may struggle.

Chapter III

Adaptive Coordinate Systems for Constrained Differential Evolution

Chapter III
Adaptive Coordinate Systems for Constrained Differential Evolution

---

III.1 Introduction

Constrained Optimization Problems (COPs) are prevalent in various real-world applications, requiring the optimization of an objective function subject to a set of constraints. As previously discussed in Chapter II, COPs introduce additional complexities due to the need to balance objective optimization and constraint satisfaction. The landscape of these problems can be highly intricate, characterized by narrow feasible regions and disconnected spaces, making the design of efficient optimization algorithms a significant challenge. Among the techniques available for addressing COPs, Evolutionary Algorithms (EAs) have gained prominence for their flexibility and robustness. Differential Evolution (DE) has emerged as one of the most effective algorithms, known for its simplicity and strong performance across diverse optimization scenarios. However, when applied to COPs, DE faces the challenge of navigating complex search landscapes where constraints and objective functions interact in non-trivial ways. To address these challenges, constraint-handling techniques (CHTs) have been integrated into DE, enabling it to better explore feasible regions and avoid infeasible solutions. However, most existing DE variants rely on fixed coordinate systems, which limit their adaptability to dynamic and complex constraint landscapes. Furthermore, recent advances, such as the use of Eigen coordinate systems, have demonstrated promise in improving DE's adaptability, yet most research has focused solely on unconstrained problems, neglecting the role of constraint violation information. This chapter introduces a novel approach, Adaptive Coordinate System for Constrained Differential Evolution (ACSCDE), which addresses these limitations by incorporating dual Eigen coordinate systems one focusing on ob-

jective function optimization and the other on constraint violation. These coordinate systems are constructed using an archive-based covariance matrix that captures information about both feasible and objective landscapes, improving the algorithm's ability to balance exploration and exploitation. In addition, ACSCDE utilizes an adaptive selection process based on the Upper Confidence Bound (UCB) method, dynamically choosing the most appropriate coordinate system at each stage of the evolutionary process. By maintaining the original coordinate system to preserve diversity and applying a tailored mutation strategy, ACSCDE significantly enhances DE's performance on COPs. The contributions of this work are:

1. Development of two Eigen coordinate systems utilizing an archive-based covariance matrix. This approach effectively harnesses population distribution data by incorporating both objective function values and constraint violation information.

2. Definition of a novel selection process for three coordinate systems based on the upper confidence bound method and an innovative rewarding mechanism.

3. Achievement of a balanced approach between handling constraints and optimizing the objective function, leading to enhanced performance of DE.

4. Comprehensive experiments conducted on widely recognized benchmark sets to evaluate the effectiveness of ACSCDE, with comparisons made against advanced constrained DE variants.

In this chapter, we will review the relevant related work, present the ACSCDE method in detail, provide experimental results, and conclude with a summary and discussion of the findings.

III.2 Related work

During the past decade, constrained DE has attracted much attention by the researchers. The current studies mainly focus on balancing between constraints and objective function while maintaining the convergence and diversity of the search. Consequently, numerous methods including different constraint-handling techniques, mutation strategies and control parameter setting have been presented to strike that balance.

Wang et al (2020) combine the correlation between the objective function and constraints with the penalty function technique to update the search direction. Wang et al (2016b) integrates the information of the objective function to the feasibility rule by archiving the discarded individuals with better objective functions for a later replacement mechanism process. Li et al (2023a) presents an adaptive penalty coefficient mechanism based on the correlation between constraints and objective function, enhanced

by a method of updating the search direction using an exponentially weighted average. Qiao et al (2022) introduces a DE variant based on a self-adaptive resources allocation method, which incorporates three collaborative mutation strategies and dynamically assigned it to individuals based on their performance feedback. Liang et al (2021) introduced a rankings-based fitness function that combines two rankings: one based on the ϵ constraint-handling technique and the other on the objective function. The final fitness is dynamically weighted, balancing constraint satisfaction and objective optimization. The method also uses three differential evolution strategies to enhance diversity and convergence. Wang et al (2019) proposed a composite DE based on three different trial vector and two stage selection mechanism based on two different constraint-handling techniques. Mohamed and Sabry (2012) proposed a modified DE variant for solving constrained optimization problems. This version introduces a novel mutation scheme, an updated parameter selection strategy, and a new constraint-handling approach. The mutation involves adding the base vector to the scaled difference between the global best and worst vectors. The selection process is modified to select a trial based on three criteria.

Recent studies have explored the use of dynamic coordinate systems based on population distribution information to enhance the performance of EAs optimization algorithms. These approaches help algorithms adapt to diverse function landscapes and generate more promising solutions. However, the majority of this work has been limited to unconstrained optimization problems.

Wang et al (2014) enhances DE by incorporating covariance matrix learning to establish a suitable Eigen coordinate system for the crossover operator which makes the algorithm more rotation-invariant and better suited for problems with high variable correlation.Wang et al (2016a), propose cumulative population distribution information is used to build an Eigen coordinate system via covariance matrix adaptation. the algorithm creates two trial vectors—one from the original coordinate system and one from the Eigen coordinate system—for each target vector, selecting the best one for survival. An eigenvector-based crossover operator was proposed by Guo and Yang (2015), where the operator utilizes the eigenvectors of the covariance matrix derived from individual solutions. This approach makes the crossover rotationally invariant by projecting donor vectors onto an eigenvector basis, providing an alternative coordinate system. In (Liu et al, 2019) author introduces an Eigen coordinate system based on cumulative population distribution information. This is achieved through a covariance matrix adaptation strategy and an archiving mechanism. the proposed framework dynamically tunes between the Eigen and original coordinate systems using a probability vector, which is updated based on performance feedback from offspring. This mechanism enables NIOAs to better identify the modality of the function landscape. Gao et al (2022) proposes a Markov decision model to adaptively choose between the Eigen and original coordinate systems. The Eigen coordinate system is constructed using cumulative population distribution

information from an archive-based covariance matrix, while the selection process is controlled via reinforcement learning. CMA-ES Hansen and Ostermeier (2001) introduces the well-established covariance matrix adaptation evolution strategy. In this approach, offspring are generated by sampling a multivariate normal distribution from an Eigen coordinate system that is continuously updated using information from both past and current generations.

While these methods demonstrate significant improvements in optimization, they are primarily designed for unconstrained problems and often neglect the complexity introduced by constraints. Our proposed approach extends the use of dynamic coordinate systems to constrained optimization. this research present an adaptive coordinate systems for DE to strike the balance between objective and constraints, and exploration and exploitation.

III.3 Proposed approach

III.3.1 Motivation

Recently, research has shown the potential of using population distribution information to guide the search process, where the distribution of high-quality solutions can provide valuable insights into the structural characteristics of the problem landscape. There are a few works involving the population distribution information during the search process of DE (Li et al, 2023b; Liang et al, 2020) in which they successfully build an Eigen coordinate system for the crossover phase that can dynamically adapt to the problem landscape to enhance the optimization performance. However, their Eigen coordinate systems are built based on the distribution information of the objective function only and are addressed to unconstrained optimization problems. That is to say, the information of the whole search process in constrained optimization (objective function, constraint violation) is underutilized, and consequently, an unreliable Eigen coordinate system may be generated to misguide the evolution direction.

Motivated by the above considerations, in this chapter, we propose the ACS-CDE in which two Eigen coordinate systems based on archive population distribution information are introduced. Specifically, we employ the information of the objective function to construct an objective-Eigen coordinate system, and the information of the constraint violation to establish a constrained-Eigen coordinate system to enhance the population evolution in both feasible and infeasible regions.

As the main operator of DE, the crossover is sensitive to the rotation of the coordinate system. The proposed coordinate system serves as an alternative to the original coordinate system employed by individuals during the crossover phase. To maintain the diversity of the population and prevent premature convergence, we implement the

crossover in both the original and Eigen coordinate systems by choosing a proper coordinate system in an adaptive manner. This scheme can increase the diversity of the population and prevent premature convergence.

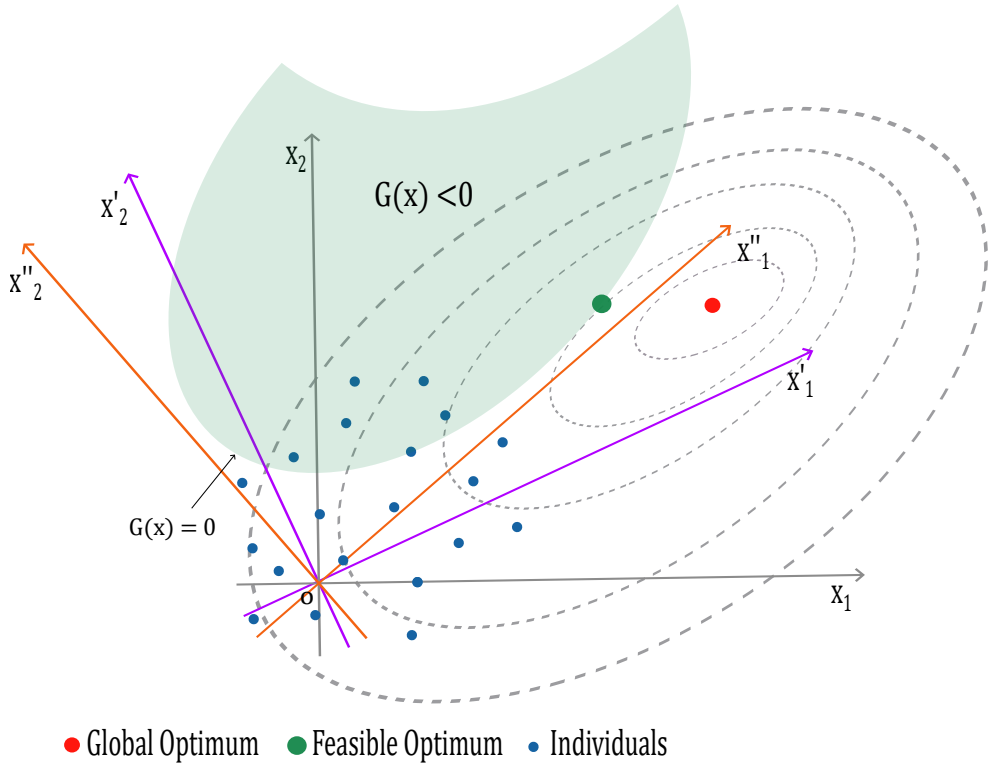


Figure III.1: Illustration of finding the optimum on different coordinate system for a multi-modal function in 2D space.

Fig.III.1 introduced the utilization of three coordinate systems. As demonstrated in Fig.III.1, the distribution information of the population can reflect the features characteristics of the function landscape and the direction to the feasible region to a certain degree.

The contours of the objective function are not aligned with the original coordinate system ox_1x_2, whereas they are aligned parallely with the objective eigen coordinate system $ox'_1x'_2$. Additionally, the constrained Eigen coordinate system $ox''_1x''_2$ is more suitable for the feasibility direction compared to the original coordinates ox_1x_2.

Consequently, it is anticipated that DE implemented within the Eigen coordinate system will be able to determine the function landscape's modality and the direction of the search. The generated offspring will fall close to the global optimum . This implies that the Eigen coordinate system has a higher probability of obtaining the optimum than the original system. It is hoped that the solutions use the objective eigen coordinate system to guide the solution toward a promising region with a better objective function, and also use the constrained eigen coordinate system to help the solution enter the feasible space in the later stage.

In the following sections, we present the proposed strategy of the ACS-CDE. Specifically, we first introduce how to build the proposed two eigen coordinate systems in Section 3.2. Then, we explain the adaptive selection mechanism to select a proper coordinate system among the eigen one and the original one in Section 3.3. Afterwards, we formulate a modified search strategy and a restart mechanism in Section 3.4.

III.3.2 Eigen coordinate system

Eigen coordinate systems are defined by the columns of an orthogonal matrix B, derived from the Eigen decomposition of the covariance matrix C:

$$C = BDB^T \tag{III.1}$$

Here, B^T is the transpose of an orthogonal matrix B, where D is a diagonal matrix. Each column of B represents an eigenvector of C, and diagonal elements of D are the square root of the eigenvalues of C.

In our approach, we involve the population information of a certain number of previous generations along with current generation solutions to better estimate the covariance matrix of both Eigen coordinate systems. For each generation t, an archive A^t that preserves these solutions selected for estimation is defined as follows:

$$A^t = S^t \cup S^{t-1} \cup \cdots \cup S^{t-k} \tag{III.2}$$

Here S^t is the set of solutions at generation t, and k denotes the length of the archive A.

The rank-μ-update strategy is employed to update the covariance matrix, benefiting from the large size of A for improved estimation. Since we have two eigen coordinate systems based on different information, two covariance matrices will be constructed. In which, we initialize each covariance matrix C with an identity matrix I formulated as $C^g = I$.

The mean vector $\vec{m}^{(g+1)}$ of the search distribution is initialized with a randomly generated point in the search space. At each subsequent generation g, $\vec{m}^{(g+1)}$ is updated with (III.3).

$$\vec{m}^{(g+1)} = \sum_{i=1}^{\mu} \omega_i \vec{v}_{(i:\mathrm{AS})} \tag{III.3}$$

where the weight coefficient ω_i is computed as:

$$\omega_i = \frac{\ln(\mu + 0.5) - \ln i}{\mu \ln(\mu + 0.5) - \sum_{j=1}^{\mu} \ln j}, i = 1, \ldots, \mu \tag{III.4}$$

Here, μ is the number of selected individuals, $\vec{v}_{(i:\mathrm{AS})}$ the i-th best individual from A, and ω_i is the i-th weight coefficient. The updated mean vector $\vec{m}^{(g+1)}$ is a weighted average of the best μ individuals from archive A, with higher quality individuals receiving higher weight coefficients.

It's noteworthy that the procedure for constructing the covariance matrix of both eigen coordinate systems is the same, thus we are only presenting one procedure approach. However, a key distinction arises when selecting the optimal individual from the archive for each coordinate. In the constrained-Eigen coordinate system, the best individuals are selected based on their minimal constraint violation degree, i.e.,

$$g(\vec{v}_{1:AS}) < g(\vec{v}_{2:AS}) < \cdots < g(\vec{v}_{\mu:AS}). \tag{III.5}$$

Conversely, in the objective eigen coordinate system, they are chosen based on the best objective function values, with,

$$f(\vec{v}_{1:AS}) < f(\vec{v}_{2:AS}) < \cdots < f(\vec{v}_{\mu:AS}). \tag{III.6}$$

This selection plays a primary role in identifying the landscape of constraints and the objective function. Additionally, the archive-based covariance matrix $C^{(g+1)}$ is estimated and updated as follows:

$$C_{\mu}^{(g+1)} = \sum_{i=1}^{\mu} \omega_i (\vec{a}_{i:AS} - \vec{m}^g)(\vec{a}_{i:AS} - \vec{m}^g)^T \tag{III.7}$$

$$C^{(g+1)} = (1 - c_\mu) C^g + c_\mu C_{\mu}^{(g+1)} \tag{III.8}$$

where $C_\mu = \frac{1}{3} \times \frac{\mu_{\mathrm{eff}}}{D^2}$, $\mu_{\mathrm{eff}} = \left(\sum_{i=1}^{\mu} \omega_i^2\right)^{-1}$, and D denotes the learning rate, the variance effective selection mass, and the dimension of the search space, respectively.

Finally, to execute the crossover in the Eigen system, we begin by transforming the target vector $\vec{x}_i$ and mutant vector $\vec{v}_i$ from the original coordinate system to the eigen one using Eqs.(III.9)-(III.10).

$$\vec{x}_i' = B^T \vec{x}_i \tag{III.9}$$

$$\vec{v}_i' = B^T \vec{v}_i \tag{III.10}$$

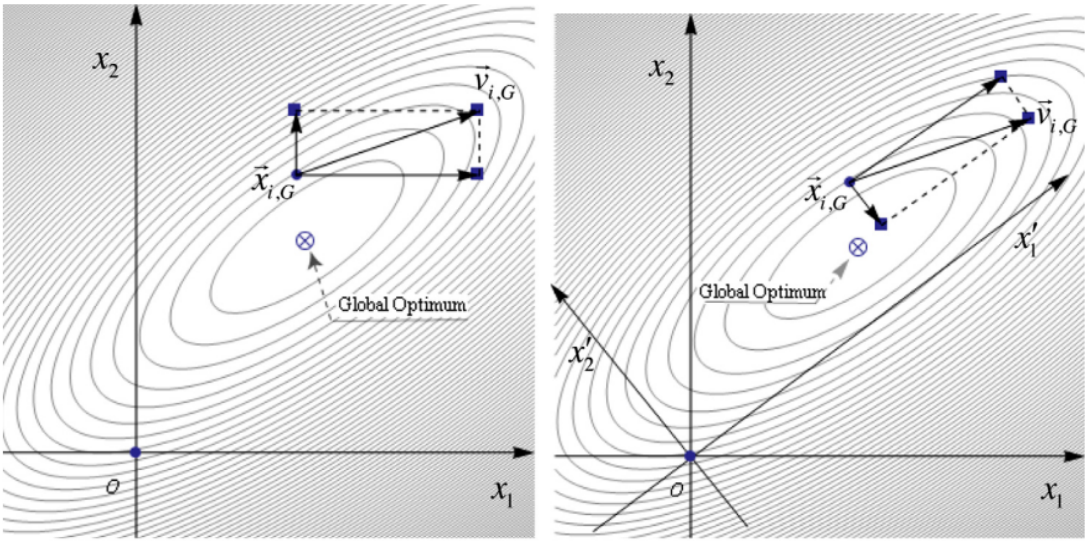
Subsequently, we apply the crossover operator to these transformed vectors, resulting

a trial vector $\vec{u}'_i$ within the eigen coordinate system as per Eq. (III.11).

$$u'_{i,j} = \begin{cases} v'_{i,j}, & \text{if } rand_j \leq CR \text{ or } j = j_{rand} \\ x'_{i,j}, & \text{otherwise.} \end{cases} \tag{III.11}$$

$$\vec{u}_i = B\vec{u}'_i \tag{III.12}$$

Finally, the trial vector $\vec{u}'_i$ is converted back to the original system using Eq. (III.12) for fitness evaluation. This process ensures that the crossover operation is performed in the eigen coordinate system, facilitating a more effective exploration of the search space while maintaining the integrity of the original coordinate system for fitness evaluation.



(a) (b)

Figure III.2: Crossover in the original coordinate system ox_1x_2 and the eigen coordinate system $ox'_1x'_2$, with $x_{i,G}$ as the target vector, $v_{i,G}$ as its mutant vector, and square markers indicating possible trial vectors.

Fig. III.2 shows the differences between the crossover operator in the original coordinate system (Fig. III.2(a)) and the crossover operator in the Eigen coordinate system (Fig. III.2(b)) for a problem with variable correlation. Suppose that the Eigen coordinate system (i.e.,$ox'_1x'_2$) is obtained after analyzing the distribution of the population. From Fig. III.2, it is clear that crossover in the Eigen coordinate system is more promising to find the global optimum, since the trial vectors generated by the crossover in the Eigen coordinate system may be more close to the global optimum than the trial vectors created by the crossover in the original coordinate system.

III.3.3 Selecting coordinate system

In optimization problems, selecting an appropriate coordinate system plays a crucial role in guiding the search process effectively. Different coordinate systems offer unique strengths and weaknesses (Liu et al, 2019; Gao et al, 2022), making them better suited for different problem landscapes and optimization phases. For example, the original coordinate system provides a straightforward representation of the problem space, while the Eigen coordinate system can align with principal directions to facilitate a more targeted search. However, choosing the right coordinate system at each stage is a challenging decision that can significantly impact convergence speed and solution quality.

1. Adaptive Selection Methods

Adaptive selection methods are common in optimization to determine the best strategies or coordinate systems. Several approaches have been proposed in the literature, including reinforcement learning (RL)-based strategies (Hu and Gong, 2022; Gao et al, 2022), random selection Wang et al (2014), probabilistic methods (Palakonda and Mallipeddi, 2020), and Bandit-based methods (Li et al, 2014a). RL-based strategies, for example, utilize reward-based frameworks to learn and adapt over time, while random selection methods aim to balance exploration by stochastically alternating between different strategies. However, RL-based methods can be computationally expensive and require significant training periods (Dulac-Arnold et al, 2021), making them less practical for every problem. Meanwhile, simple random or probabilistic approaches lack structured exploitation and may fail to efficiently leverage useful coordinate systems.

2. Bandit-Based Methods

Bandit-based methods, particularly those based on Multi-Armed Bandit (MAB) algorithms (Auer et al, 2002), have gained traction due to their ability to balance exploration and exploitation. In MAB-based approaches, the goal is to allocate resources between different competing strategies (or "arms") whose properties are partially known and evolve with feedback over time. These methods aim to exploit promising strategies while ensuring sufficient exploration of other options that might prove beneficial later in the search.

The Upper Confidence Bound (UCB) algorithm is a notable MAB approach known for its recent advancements and asymptotic optimality guarantees. In UCB-based MAB algorithms, each arm possesses an empirical quality estimate $q_i^{(t)}$, and a confidence interval based on the number of times it has been sampled $n_i^{(t)}$. At each time point t, the arm maximizing the following function is selected:

$$q_i^{(t)} + C \times \sqrt{\frac{2 \times \ln\left(\sum_{j=1}^{K} n_j^{(t)}\right)}{n_i^{(t)}}} \tag{III.13}$$

Here, C serves as a scaling factor, balancing the trade-off between exploitation (favoring arms with superior empirical rewards) and exploration (favoring less-sampled arms).

To overcome the limitations of existing strategies, we introduce an enhanced Upper Confidence Bound (UCB) method that balances exploration and exploitation. This approach involves two key components which will be further discussed (Li et al, 2014a):

1. **A credit assignment scheme** which assigns reward values to coordinate systems based on offspring improvement on both fitness f and constraint violation g aspects.

2. **A selection mechanism** that dynamically selects the coordinate system using the assigned credits.

Various strategies for credit assignment have been proposed, differing mainly in the following aspects: (i) how to measure the impact of strategy application; (ii) how to allocate credit based on these impact assessments. The most common measure of strategy application impact is the improvement in fitness resulting from generating new offspring (Li et al, 2014a). However, for constrained optimization, it is challenging to measure the performance of the new offspring due to the quality of the offspring depending on both fitness and constraint violation. Also, the range of fitness improvements varies from problem to problem and even at different stages of an optimization process.

In this work, we introduce a new strategy that measures the quality of the offspring improvement. The collected information derived from the offspring when the coordinate system performs better can be categorized into three ranks ie., (1), (2) and (3) based on f (function value) and g (constraint violation) which can be introduced as follow:

(1) Offspring is better in both f and g

(2) Offspring is better in g but not f.

(3) Offspring is better in f but not g.

In this strategy, we first calculate the reward value received by each improvement based on its rank, noted as improvement rank reward (IRR). The value of IRR is flexible and influenced by the overall improvement distribution of the population at each generation, where it reflects the changing performance and adjusts accordingly. Here, for each generation t, we sum the number of candidate solutions that has been successfully improved with the ith rank (ie., 1, 2 or 3), denoted as S_i^t. Then, we define the measured reward value for each rank of improvement using a cumulative sum as follows:

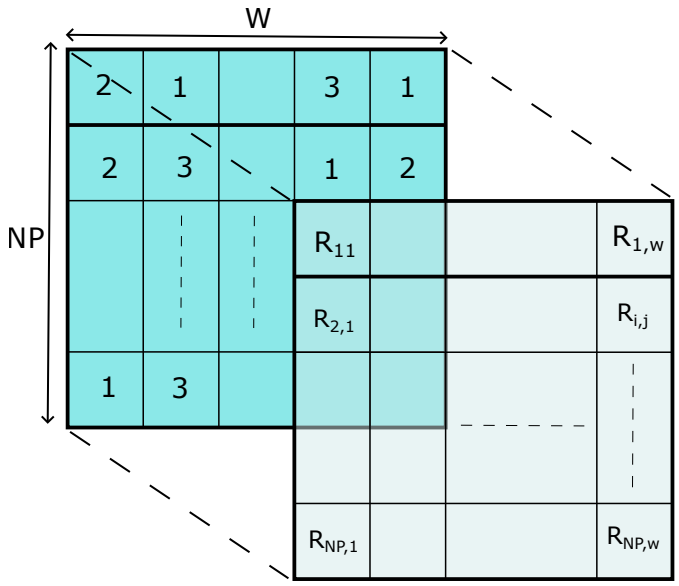


Figure III.3: A 2D Sliding Window that store individual based reward of each coordinate system .

$$IRR^t_{rank_{1..3}} = \frac{\sum_{i=rank}^{n} S_i^t}{\sum_{i=1}^{n} S_i^t} \tag{III.14}$$

Clearly, the smaller the rank, the larger the reward value it gets. Also, all improvements in the same rank will receive the same IRR value. Each individual i that successfully applied a coordinate system j that generated a new offspring solution ranked r will receive a reward value $R_{ij}^t = IRR_r^t$ for that coordinate system.

A sliding window with $NP \times W \times 2$ dimensions is used to store reward values R_{ij} of the recently used coordinate system by each individual. It is organized as a FIFO queue, ensuring that the stored performance information is relevant to the current search situation. Figure III.3 shows the structure of a sliding window: the first layer stores the coordinate system indexes (1, 2, 3) used by each individual, and their associated reward values are stored in the second layer.

As for the credit value of a coordinate system, it is the average sum of its recent received reward R in the current sliding window. Based on that, each individual i has a credit value for each coordinate system j, denoted as C_{ij}^t determined at each generation t. The selection of the coordinate system for each individual is then based on a bandit-based selection scheme similar to the UCB algorithm. The index of the selected coordinate system is defined as follows:

$$\Omega_i = \underset{j=1,2,3}{\operatorname{argmax}} \left(C_{ij}^t + \delta \times \sqrt{\frac{2 \times \ln \sum n_{ij}^t}{n_{ij}^t}} \right) \tag{III.15}$$

$$C_{ij}^t = \frac{\sum_{t-k}^{t} R_{ij}^t}{n_{ij}^t} \tag{III.16}$$

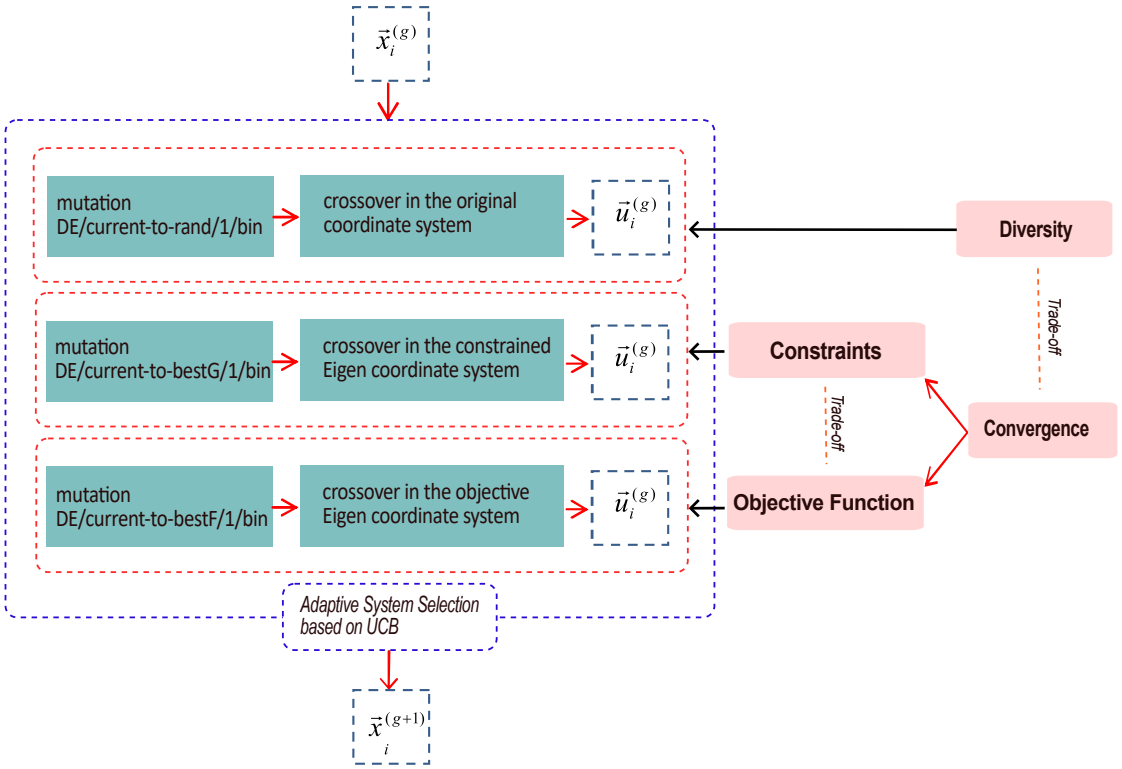


Figure III.4: Principal of the proposed approach in achieving the trade-off constraints vs objective and objective vs convergence.

with Ω_i is the selected coordinate system for individual i, δ a scaling factor balancing exploration and exploitation and n_{ij}^t is the number of times system j has been selected recently by individual i. It's worth noting that this method is implemented only after each system has been tried at least once, ensuring a fair exploration phase.

III.3.4 Diversity and convergence

For further enhancing the convergence and diversity of the search and motivating the balance between constraints and objective function, we employed a distinct search algorithm for each coordinate system to enhance their specialized performance Fig. III.4. For the original coordinate system, we utilize the `DE/current-to-rand/1` mutation strategy (II.7). In this approach, the target vector learns from information of a randomly selected individual. This strategy reinforces population diversity since the original coordinate system exhibits no bias towards specific search directions. While for both Eigen coordinate systems, we employ the `DE/current-to-best/1/bin` mutation strategy (II.6). This choice allows us to utilize information from the best individual in the population to guide the search. However, the criteria for selecting the best individual differ between the constrained and objective Eigen coordinate systems:

1. *Constraint Eigen Coordinate System:* The best individual selected with the least degree of constraint violation. This selection aims to prioritize the satisfaction of

constraints.

2. *Objective Eigen Coordinate System:* The best individual is selected based on the best value of objective function. This choice aims to focus the search on optimizing the objective function.

Moreover, to avoid the risk of population stagnation within the highly nonlinear and multi-modal infeasible regions, we incorporate a restart mechanism, similar to the one proposed in (Wang et al, 2019). This mechanism is based on the standard deviation of the constraint violation which is calculated and compared to a defined threshold to identify premature convergence within the infeasible region. A random population will be generated when stagnation appears.

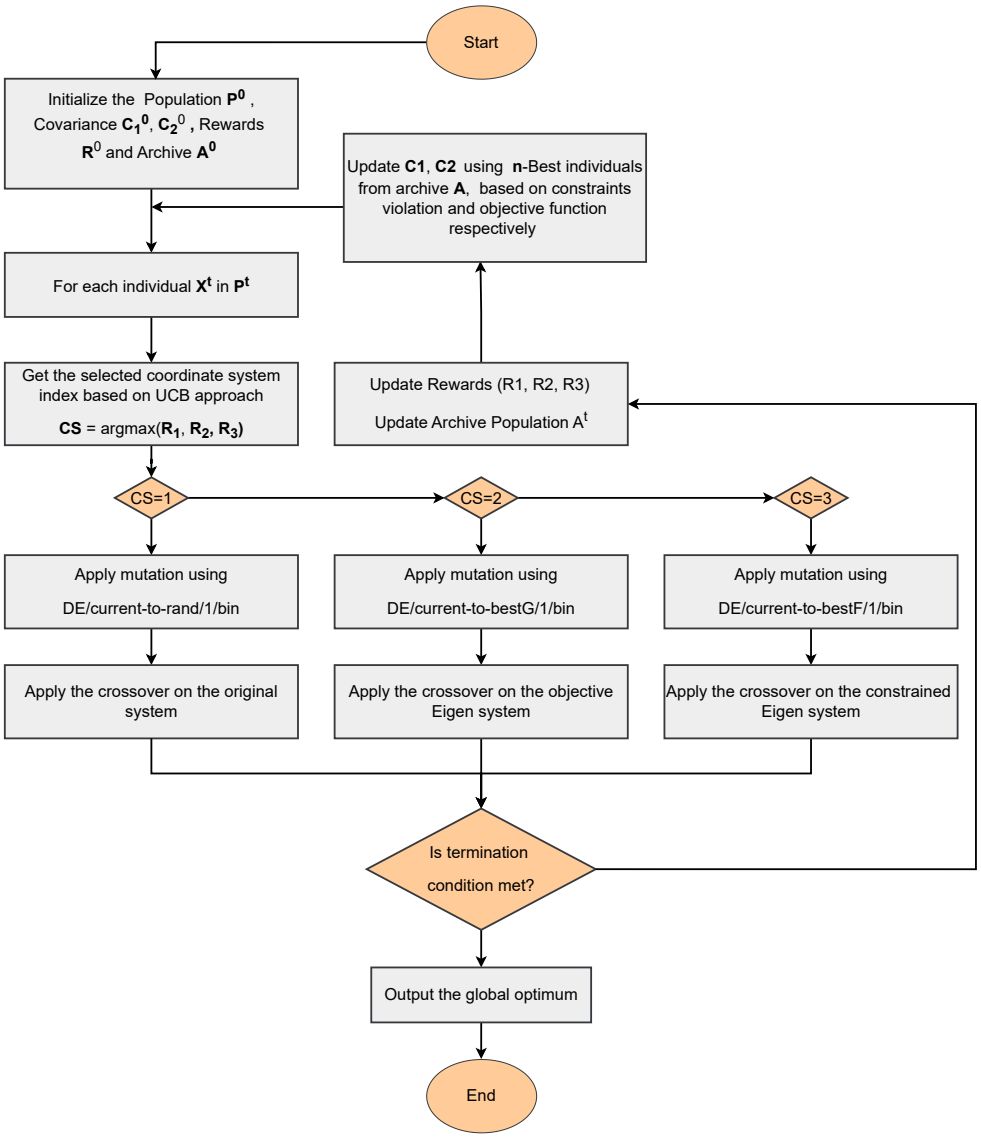


Figure III.5: A flowchart of ACS-CDE procedure.

Algorithm 3 The procedure of ACS-CDE

Require: $NP, D, MaxFEs$
Ensure: Optimal solution $\vec{x}_{best}$
Initialize $FEs = NP$, $SW, IRR^0 = [1, 2/3, 1/3]$
Initialize $\mathbf{C}_{obj}^{(0)} = \mathbf{B}_{obj}^{(0)} = \mathbf{I}$, $\mathbf{C}_{cv}^{(0)} = \mathbf{B}_{cv}^{(0)} = \mathbf{I}$
Initialize $\mathbf{\Omega} = [1, 3, .., 2]_{1\times NP}$ $\mathbf{P}^{(0)} = [\vec{x}_1, .., \vec{x}_{NP}]_{D\times NP}$ $\mathbf{A}^{(0)} = \varnothing$
Calculate $F(\mathbf{x}_i^{(g)})$ by the objective function, where $i = (1, 2, \ldots, N)$
while $FEs \leq MaxFEs$ **do**
    for $i = 1 : NP$ **do**
        if $\Omega_i == 1$ **then**
            $\vec{\mathbf{v}}_i$ generated by Eq. (II.7)
            $\vec{\mathbf{u}}_i$ generated by Eq. (II.8)
        else if $\Omega_i == 2$ **then**
            $\vec{\mathbf{v}}_i$ generated by Eq. (II.6) with $\vec{\mathbf{x}}_{best}$ (ie., based on objective function)
            $\vec{\mathbf{u}}_i$ generated by Eq. (III.11) with $\mathbf{B}_{obj}$
        else if $\Omega_i == 3$ **then**
            $\vec{\mathbf{v}}_i$ generated by Eq. (II.6) with $\vec{\mathbf{x}}_{best}$ (ie., based on constraint violation)
            $\vec{\mathbf{u}}_i$ generated by Eq. (III.11) with $\mathbf{B}_{cv}$
        end if
        Evaluate the offspring $F(\vec{u}_i)$ & $G(\vec{u}_i)$;
        Apply the ϵ-constrained method to compare $\vec{\mathbf{x}}_i$ and $\vec{\mathbf{u}}_i$ Eq. (II.9)
        store the selected one into $\mathbf{P}^{(t+1)}$;
        FEs = FEs + 1;
    end for
    Update $\mathbf{A}^t$ using Eq. (III.2)
    Sort $\mathbf{A}^t$ based on objective function values
    Update $\mathbf{C}_{obj}^{t+1}$ to obtain $\mathbf{B}_{obj}^{t+1}$ Eq. (III.8)
    Sort $\mathbf{A}^t$ based on constraint violation values
    Update $\mathbf{C}_{cv}^{t+1}$ to obtain $\mathbf{B}_{cv}^{t+1}$ Eq. (III.8)
    compute the reward $\mathbf{IRR}^t$ associated with each improvement rank by Eq. (III.14)
    Compute the credit value $\mathbf{CSC}_{ij}$ of each coordinate system by Eq. (III.16)
    Update $\mathbf{\Omega}$ ie., $\Omega = \arg\max_{j=1,2,3}\left(CSC^t + \delta \times \sqrt{\frac{2\times\ln\sum n_{ij}^t}{n_{ij}^t}}\right)$
end while

A pseudo-code summary of the ACS-CDE framework is provided in Algorithm 3, outlining the key steps, including Eigen system construction, adaptive selection, and population evolution. Additionally, a flowchart III.5 illustrates the main steps of the algorithm, offering a visual overview of the framework.

III.4 Experimental study

III.4.1 Benchmark problems and parameter settings

Two benchmark test suites, comprising test functions with different features and complexities, were chosen in order to verify the performance of ACS-CDE. Specifically, IEEE CEC2010 (Mallipeddi and Suganthan, 2010) and the IEEE CEC2017 (Wu et al, 2017) competition for single objective constrained optimization, respectively.

Table III.1 describes the population size (NP) and the maximum number of function evaluations (MaxFEs)

Test Functions	**MaxFEs**	**NP**
10D-CEC2010	2.0E+05	60
30D-CEC2010	6.0E+05	80
50D-CEC2017	1.0E+06	80
100D-CEC2017	2.0E+06	100

Table III.1: Parameter settings for MaxFEs and NP.

Twenty-five (25) separate runs were made for every test function. Also, the value 0.0001 is set for the tolerance value δ as recommended in (Wang et al, 2016b), Parameter are set the same for the compared algorithms. Furthermore, parameters μ = 10E-8 and $Tc = 0.5 \times T$ for the restart scheme and epsilon-constrained methods respectively.

The Friedman and the multi-problem Wilcoxon tests at significance level of 0.05 were carried out for testing the statistical significance using KEEL software (Alcalá-Fdez et al, 2009).

III.4.2 Experiment on IEEE CEC2010 benchmark

Firstly, our proposed approach ACS-CDE was tested on 18 test functions for 10-dimensional (10D) and 18 test functions for 30-dimensional (30D). The performance of ACS-CDE compared with five competitive COEAs recently proposed: C2oDE (Wang et al, 2019), FROFI (Wang et al, 2016b), CORCO (Wang et al, 2020), DeCODE (Wang et al, 2021), ITLBO (Wang et al, 2018). Given that (Mallipeddi and Suganthan, 2010) does not provide the true optimum of CEC2010 test functions, the record of 25 independent runs is used to obtain the average and standard deviation of the objective function values. Following that, multi-problem Wilcoxon's and Friedman's statistical tests were used to compare the algorithm's performance in concurrently.

In terms of the statistical results on the test functions with 10D and 30D, Tables III.4 and III.7 summarize the experimental results of the mean objective function values and standard deviations for the optimal solutions obtained by various algorithms over 25 independent runs denoted as "Mean OFV", "Std Dev". For each test function, "+", "−", and "≈" denote the performance of ACS-CDE is better than, worse than, and similar to that of other COEAs respectively. The symbol "▽" indicates that an algorithm failed to find feasible solutions in all 25 runs.

As show in Table III.4, for 10D test functions ACS-CDE outperform C2oDE, FROFI, CORCO, DeCODE, ITLBO on seven, eight, six, six and eight test functions. Conversely, it performs less than these algorithms on five, two, four, three and three test functions. Furthermore, results of the Wilcoxon's signed ranks test in Table III.3 reflect that ACS-CDE performs better than five compared algorithms. It can be observed that, the values of R+ are superior than R- in all cases. Additionally, the Friedman's test shown in Table III.2 illustrate that, ACS-CDE achieved the first rank among all the algorithm (2.8611) followed by CORCO(3.3611).

Algorithm	**Ranking**
ACS-CDE	2.8611
CORCO	3.3611
DeCODE	3.5833
C2oDE	3.5833
FROFI	3.7778
ITLBO	3.8333

Table III.2: Ranking of ACS-CDE and Five Methods Using Friedman's Test on IEEE CEC2010 10D

Vs	R^+	R^-	p-value	$\alpha = 0.1$	$\alpha = 0.05$
C2oDE	107.5	63.5	≥ 0.2	No	No
FROFI	124.0	47.0	9.874E-02	Yes	No
CORCO	103.0	68.0	≥ 0.2	No	No
DeCODE	100.0	53.0	≥ 0.2	No	No
ITLBO	108.5	44.5	1.389E-01	No	No

Table III.3: Wilcoxon Test Results for ACS-CDE vs. Five Methods on 18 IEEE CEC2010 10D

Table III.4: Results of ACS-CDE and Comparison Methods on IEEE CEC2010 10D Test Suite

CEC2010	C2oDE	FROFI	CORCO	DeCODE	ITLBO	ACS-CDE
D10	Mean OFV ± Std	Mean OFV ± Std	Mean OFV ± Std	Mean OFV ± Std	Mean OFV ± Std	Mean OFV ± Std
C01	-7.33E-01 ± 2.81E-02 (+)	-7.47E-01 ± 1.20E-12 (≈)	-7.46E-01 ± 4.57E-03 (+)	-7.47E-01 ± 1.35E-03 (≈)	-7.47E-01 ± 2.82E-16 (≈)	-7.47E-01 ± 3.65E-16
C02	-2.26E+00 ± 3.35E-02 (−)	-1.91E+00 ± 2.34E-01 (+)	-2.15E+00 ± 1.22E-01 (−)	-2.19E+00 ± 1.37E-01 (−)	-2.04E+00 ± 7.90E-02 (+)	-2.11E+00 ± 1.03E-01
C03	7.10E-01 ± 2.46E+00 (+)	0.00E+00 ± 0.00E+00 (≈)	0.00E+00 ± 0.00E+00 (≈)	0.00E+00 ± 0.00E+00 (≈)	0.00E+00 ± 0.00E+00 (≈)	0.00E+00 ± 0.00E+00
C04	-1.00E-05 ± 0.00E+00 (≈)	-1.00E-05 ± 0.00E+00 (≈)	-1.00E-05 ± 0.00E+00 (≈)	-1.00E-05 ± 7.79E-16 (≈)	-1.00E-05 ± 1.76E-15 (≈)	-1.00E-05 ± 2.43E-09
C05	-4.84E+02 ± 3.48E-13 (≈)	-4.84E+02 ± 3.16E-07 (≈)	-4.84E+02 ± 3.48E-13 (≈)	-4.84E+02 ± 3.48E-13 (≈)	-4.84E+02 ± 8.71E-12 (≈)	-4.84E+02 ± 8.00E-10
C06	-5.79E+02 ± 1.58E-02 (≈)	-5.79E+02 ± 1.67E-03 (≈)	-5.79E+02 ± 4.58E-09 (≈)	-5.79E+02 ± 1.47E-13 (≈)	-5.79E+02 ± 6.12E-08 (≈)	-5.79E+02 ± 4.98E+00
C07	3.19E-01 ± 1.10E+00 (+)	1.59E-01 ± 7.97E-01 (+)	0.00E+00 ± 0.00E+00 (≈)	0.00E+00 ± 0.00E+00 (≈)	0.00E+00 ± 0.00E+00 (≈)	0.00E+00 ± 0.00E+00
C08	8.53E+00 ± 7.04E+00 (+)	7.90E+00 ± 4.75E+00 (−)	8.55E+00 ± 4.27E+00 (+)	8.62E+00 ± 4.40E+00 (+)	7.72E+00 ± 5.96E+00 (−)	7.96E+00 ± 4.39E+00
C09	1.59E+00 ± 2.16E+00 (+)	2.90E+01 ± 3.88E+01 (+)	0.00E+00 ± 0.00E+00 (≈)	4.38E+00 ± 1.83E+01 (+)	0.00E+00 ± 0.00E+00 (≈)	0.00E+00 ± 0.00E+00
C10	3.84E+01 ± 1.16E+01 (+)	4.01E+01 ± 8.35E+00 (+)	2.01E-01 ± 9.83E-01 (+)	4.17E+01 ± 2.17E-14 (+)	1.67E+00 ± 8.35E+00 (+)	0.00E+00 ± 0.00E+00
C11	-1.52E-03 ± 4.78E-11 (≈)	-1.52E-03 ± 4.84E-15 (≈)	-1.52E-03 ± 2.56E-18 (≈)	-1.52E-03 ± 3.85E-18 (≈)	-1.50E-03 ± 3.37E-05 (+)	-1.52E-03 ± 6.25E-04
C12	-9.53E+01 ± 1.58E+02 (−)	-4.00E+02 ± 2.30E+02 (−)	-1.33E+01 ± 6.09E+01 (−)	-4.76E+00 ± 2.28E+01 (−)	-2.29E+01 ± 1.14E+02 (−)	-1.75E+00 ± 2.49E+00
C13	-6.84E+01 ± 2.81E-14 (≈)	-6.84E+01 ± 1.73E-07 (≈)	-6.84E+01 ± 2.97E-14 (≈)	-6.84E+01 ± 2.90E-14 (≈)	-6.54E+01 ± 1.41E+00 (+)	-6.84E+01 ± 3.52E-02
C14	0.00E+00 ± 0.00E+00 (≈)	0.00E+00 ± 0.00E+00 (≈)	4.78E-01 ± 1.32E+00 (+)	0.00E+00 ± 0.00E+00 (≈)	4.34E-01 ± 2.17E+00 (+)	0.00E+00 ± 0.00E+00
C15	3.27E+00 ± 1.24E+00 (+)	3.23E+00 ± 1.22E+00 (+)	1.32E+00 ± 4.07E+00 (+)	3.09E+00 ± 1.37E+00 (+)	3.18E+00 ± 8.22E+00 (+)	0.00E+00 ± 0.00E+00
C16	0.00E+00 ± 0.00E+00 (−)	1.43E-02 ± 2.45E-02 (+)	2.94E-03 ± 1.44E-02 (+)	6.36E-04 ± 3.18E-03 (+)	2.61E-01 ± 3.27E-01 (+)	3.43E-06 ± 2.17E-02
C17	0.00E+00 ± 0.00E+00 (−)	9.83E-02 ± 2.21E-01 (+)	0.00E+00 ± 0.00E+00 (−)	5.89E-11 ± 1.36E-10 (+)	4.06E-01 ± 4.95E-01 (+)	3.23E-16 ± 1.84E-01
C18	0.00E+00 ± 0.00E+00 (−)	1.65E-25 ± 7.48E-25 (+)	0.00E+00 ± 0.00E+00 (−)	0.00E+00 ± 0.00E+00 (−)	0.00E+00 ± 0.00E+00 (−)	1.04E-25 ± 3.76E-25
+	7	8	6	6	8	/
−	5	2	4	3	3	/
≈	6	8	8	9	7	/

In term of 30D test functions Table III.7 illustrate that ACS-CDE has an edge over C2oDE, FROFI, CORCO, DeCODE, ITLBO on 12, 11, 7, 12 and 13 test functions. However, C2oDE, FROFI, CORCO, DeCODE, ITLBO surpass ACS-CDE on one, three, four, three and two test functions. Similarly, the results of multi-problem Wilcoxon's and Friedman's tests were summarized in Table III.6 and Table III.5, respectively. In Table III.6 all, R+ values are superior than R- values in all cases and the significant difference at $\alpha = 0.05$ can be observed in four cases. Additionally, ACS-CDE obtain the lowest value (2.3889) followed by CORCO at the Friedman's test III.5. These findings suggest that ACS-CDE has very competitive performance against the compared algorithm and demonstrates a strong capability on solving both 10 and 30 dimensions of IEEE CEC2010 test functions.

Algorithm	**Ranking**
ACS-DE	2.3889
CORCO	2.8056
FROFI	3.7500
DeCODE	3.7500
C2oDE	3.8611
ITLBO	4.4444

Table III.5: Ranking of ACS-CDE and Five Methods Using Friedman's Test on IEEE CEC2010 30D

Vs	R^+	R^-	p-value	$\alpha = 0.1$	$\alpha = 0.05$
C2oDE	132.0	21.0	6.652E-3	Yes	Yes
FROFI	133.0	38.0	3.85E-2	Yes	Yes
CORCO	87.5	65.5	≥ 0.2	No	No
DeCODE	126.5	26.5	1.62129E-2	Yes	Yes
ITLBO	123.5	29.5	2.495E-2	Yes	Yes

Table III.6: Wilcoxon Test Results for ACS-CDE vs. Five Methods on 18 IEEE CEC2010 30D Functions

Table III.7: Results of ACS-CDE and Comparison Methods on IEEE CEC2010 30D Test Suite.

CEC2010	C2oDE	FROFI	CORCO	DeCODE	ITLBO	ACS-CDE
D30	Mean ± Std	Mean ± Std	Mean ± Std	Mean ± Std	Mean ± Std	Mean ± Std
C01	-8.20E-01 ± 2.55E-03 (+)	-8.21E-01 ± 1.63E-03 (+)	-8.18E-01 ±4.28E-03 (+)	-8.21E-01 ± 1.61E-03 (+)	-8.20E-01 ± 1.03E-03 (+)	-8.22E-01 ± 1.08E-03
C02	-2.03E+00 ± 1.07E+00 (+)	-1.99E+00 ± 4.03E-02 (+)	-2.07E+00 ±1.70E-01 (+)	-2.24E+00 ± 2.60E-02 (−)	-2.05E+00 ± 6.22E-02 (+)	-2.19E+00 ± 5.40E-02
C03	3.75E+01 ± 2.16E+01 (+)	2.75E+01 ± 5.73E+00 (+)	0.00E+00 ±0.00E+00 (≈)	2.06E+01 ± 1.31E+01 (+)	5.63E+01 ± 6.22E+01 (+)	0.00E+00 ± 0.00E+00
C04	-1.29E-06 ± 2.78E-06 (+)	-3.33E-06 ± 9.55E-11 (−)	-3.23E-06 ±3.16E-07 (−)	-3.33E-06 ± 4.10E-12 (−)	2.13E-03 ± 1.82E-03 (+)	-3.15E-06 ± 4.93E-07
C05	-4.82E+02 ± 7.62E-01 (+)	-4.82E+02 ± 1.12E+00 (+)	-4.84E+02 ±7.64E-02 (≈)	-4.83E+02 ± 1.08E-01 (+)	-4.82E+02 ± 1.52E+00 (+)	-4.84E+02 ± 9.76E-06
C06	-5.31E+02 ± 6.89E-02 (≈)	-5.28E+02 ± 8.56E-01 (+)	-5.29E+02 ±1.20E+00 (+)	-5.28E+02 ± 1.13E+00 (+)	-5.31E+02 ± 3.73E-01 (≈)	-5.31E+02 ± 5.43E-02
C07	1.37E-27 ± 6.86E-27 (+)	1.59E-01 ± 7.97E-01 (+)	0.00E+00 ±0.00E+00 (≈)	1.59E-01 ± 7.97E-01 (+)	3.19E-01 ± 1.10E+00 (+)	0.00E+00 ± 0.00E+00
C08	3.67E+00 ± 1.84E+01 (+)	0.00E+00 ± 0.00E+00 (≈)	1.59E-01 ±7.97E-01 (+)	4.57E+00 ± 2.28E+01 (+)	7.99E+00 ± 2.52E+01 (+)	0.00E+00 ± 0.00E+00
C09	1.13E+01 ± 2.23E+01 (+)	4.25E+01 ± 3.90E+01 (+)	0.00E+00 ±0.00E+00 (≈)	8.92E+00 ± 2.37E+01 (+)	1.76E-01 ± 8.80E-01 (+)	0.00E+00 ± 0.00E+00
C10	3.13E+01 ± 4.82E-06 (+)	3.14E+01 ± 1.32E-01 (+)	5.30E-01 ±2.65E+00 (−)	3.13E+01 ± 1.49E-05 (+)	4.00E+01 ± 5.13E+01 (+)	1.62E+01 ± 8.20E+00
C11	-3.89E-04 ± 1.62E-05 (+)	-3.92E-04 ± 5.23E-07 (≈)	-3.92E-04 ±3.37E-11 (≈)	-3.92E-04 ± 4.08E-08 (≈)	-3.86E-04 ± 1.14E-05 (+)	-3.92E-04 ± 2.96E-08
C12	-1.99E-01 ± 1.41E-04 (≈)	-1.99E-01 ± 1.21E-06 (≈)	-1.99E-01 ±7.19E-08 (≈)	-1.99E-01 ± 2.94E-09 (≈)	-1.99E-01 ± 6.16E-05 (≈)	-1.99E-01 ± 3.71E-07
C13	-6.81E+01 ± 6.47E-01 (≈)	-6.83E+01 ± 3.50E-01 (−)	-6.80E+01 ±8.01E-01 (+)	-6.80E+01 ± 5.93E-01 (+)	-5.04E+01 ± 1.19E+00 (+)	-6.81E+01 ± 5.75E-01
C14	0.00E+00 ± 0.00E+00 (≈)	1.59E-01 ± 7.97E-01 (+)	1.59E-01 ±7.97E-01 (+)	1.59E-01 ± 7.97E-01 (+)	4.78E-01 ± 1.32E+00 (+)	0.00E+00 ± 0.00E+00
C15	2.16E+01 ± 2.80E-07 (+)	2.16E+01 ± 7.14E-05 (+)	0.00E+00 ±0.00E+00 (−)	2.18E+01 ± 1.18E+00 (+)	1.93E+01 ± 6.83E+00 (−)	2.07E+01 ± 2.65E-08
C16	0.00E+00 ± 0.00E+00 (≈)	0.00E+00 ± 0.00E+00 (≈)	0.00E+00 ±0.00E+00 (≈)	0.00E+00 ± 0.00E+00 (≈)	0.00E+00 ± 0.00E+00 (≈)	0.00E+00 ± 0.00E+00
C17	2.42E+02 ± 6.42E+02 (−)	3.98E-01 ± 5.32E-01 (−)	1.35E-01 ±2.36E-01 (−)	3.85E+01 ± 1.34E+02 (−)	5.61E-01 ± 1.54E+00 (−)	2.68E+02 ± 8.75E+02
C18	1.54E-07 ± 7.71E-07 (+)	2.14E-04 ± 1.07E-03 (+)	3.65E-07 ±1.82E-06 (+)	1.50E+00 ± 7.48E+00 (+)	4.56E-17 ± 1.31E-16 (+)	4.21E-22 ± 3.22E-20
(+)	12	11	7	12	13	/
(−)	1	3	4	3	2	/
(≈)	5	4	7	3	3	/

III.4.3 Experiment on IEEE CEC2017 benchmark

To further investigate the performance of the proposed ACS-CDE, the comparison on the high dimensional i.e., 50D and 100D 28 test functions of the IEEE CEC2017 are used. Comparative experiments results are conducted with ECO-HTC (Peng et al, 2023), CORCO (Wang et al, 2020), ITLBO Wang et al (2018), LSHADE44 (Polakova, 2017) and DeCODE (Wang et al, 2021). LSHADE44 obtain the top rankings in the IEEE CEC2017 competition and the rest of competitors are recent advanced CDE which are tested on these 28 functions and achieved excellent results. Since the optimal solutions are unknown for the CEC2017 benchmarks, Tables III.10 and III.13 show the comparison results with the 'Mean OFV and Std Dev over 25 independent runs for each test function on 50D and 100D, respectively. Note, that test functions where all algorithms failed to find feasible solutions on the benchmark are excluded from the comparison.

Algorithm	**Ranking**
ACS-CDE	**2.0455**
CORCO	3.1818
DeCODE	3.4318
LSHADE44	3.9091
ITLBO	4.0227
ECO-HTC	4.4091

Table III.8: Ranking of ACS-CDE and Five Methods Using Friedman's Test on IEEE CEC2017 50D

Vs	R^+	R^-	p-value	$\alpha = 0.1$	$\alpha = 0.05$
ECO-HTC	208.5	22.5	5.588E-4	Yes	Yes
CORCO	213.0	18.0	2.412E-4	Yes	Yes
ITLBO	220.5	10.5	4.673E-5	Yes	Yes
LSHADE44	214.5	38.5	3.046E-3	Yes	Yes
DeCODE	202.5	28.5	1.4872E-3	Yes	Yes

Table III.9: Wilcoxon Test Results for ACS-CDE vs. Five Methods on 22 IEEE CEC2017 50D Functions

Table III.10: The results obtained by ACS-CDE and comparison methods on IEEE CEC2017 test suite with 50D.

CEC2017 D50	ECO-HTC Mean ± STD	CORCO Mean ± STD	ITLBO Mean ± STD	LSHADE44 Mean ± STD	DeCODE Mean ± STD	ACS-CDE Mean ± STD
C01	1.59E-17 ± 3.02E-17 (+)	5.35E-24 ± 3.21E-23 (+)	4.68E-13 ± 9.32E-13 (+)	7.79E-29 ± 1.42E-29 (−)	9.03E-21 ± 8.43E-21 (+)	1.20E-25 ± 1.13E-24
C02	1.12E-18 ± 1.12E-18 (+)	1.50E-24 ± 2.56E-24 (+)	1.55E-13 ± 2.25E-13 (+)	9.79E-29 ± 4.60E-29 (−)	3.43E-20 ± 6.97E-20 (+)	5.63E-25 ± 1.06E-24
C03	1.34E-16 ± 2.98E-16 (+)	4.04E-23 ± 2.42E-23 (+)	2.28E-13 ± 3.88E-13 (+)	8.95E+05 ± 7.40E+05 (+)	2.59E-20 ± 2.54E-20 (+)	1.49E-25 ± 8.70E-25
C04	1.41E+02 ± 2.39E+01 (+)	1.46E+01 ± 1.19E+00 (+)	3.97E+02 ± 1.69E+01 (+)	1.36E+01 ± 5.44E-15 (−)	1.48E+01 ± 2.01E+00 (+)	1.41E+01 ± 2.91E+00
C05	1.06E+01 ± 3.40E+00 (+)	1.59E-01 ± 9.02E-01 (+)	3.19E-01 ± 1.10E+00 (+)	1.68E-28 ± 2.55E-27 (−)	3.99E-01 ± 1.26E+00 (+)	5.22E-27 ± 2.26E-27
C06	▽ (+)	3.70E+02 ± 9.41E+01 (+)	▽ (+)	▽ (+)	6.75E+02 ± 9.52E+01 (+)	2.98E+02 ± 1.63E+02
C07	▽ (+)	-5.74E+01 ± 2.82E+02 (+)	▽ (+)	-1.79E+02 ± 8.97E+01 (+)	-7.16E+01 ± 1.21E+02 (+)	-1.07E+03 ± 1.61E+02
C08	3.54E-03 ± 6.21E-04 (+)	-1.33E-04 ± 1.72E-06 (−)	-1.03E-04 ± 5.80E-06 (+)	-1.30E-04 ± 2.77E-20 (+)	-1.33E-04 ± 9.52E-07 (−)	-1.31E-04 ± 7.62E-06
C09	1.00E+00 ± 1.02E+00 (+)	-1.90E-03 ± 3.65E-04 (+)	1.09E-02 ± 5.56E-02 (+)	-2.04E-03 ± 1.33E-18 (≈)	2.12E+00 ± 1.30E+00 (+)	-2.04E-03 ± 1.25E-10
C10	1.03E-03 ± 1.90E-04 (+)	-4.81E-05 ± 2.79E-07 (−)	-3.03E-05 ± 3.64E-06 (+)	-4.83E-05 ± 0.00E+00 (−)	-4.82E-05 ± 3.95E-08 (−)	-4.76E-05 ± 1.44E-06
C11	▽ (≈)	▽ (≈)	▽ (≈)	-1.76E+00 ± 3.33E-01 (−)	▽ (≈)	▽
C12	1.43E+01 ± 5.30E+00 (+)	8.93E+00 ± 5.37E-01 (+)	1.57E+01 ± 8.54E+00 (+)	4.98E+01 ± 1.99E+01 (+)	1.30E+01 ± 6.33E+00 (+)	3.98E+00 ± 6.39E-05
C13	2.46E+03 ± 1.21E+03 (+)	4.78E-01 ± 1.49E+00 (+)	1.15E+01 ± 3.05E+01 (+)	2.67E+01 ± 1.36E+01 (+)	3.99E-01 ± 1.26E+00 (+)	1.94E-25 ± 8.85E-26
C14	1.18E+00 ± 4.77E-02 (+)	1.40E+00 ± 2.50E-01 (+)	1.13E+00 ± 9.77E-02 (+)	1.40E+00 ± 3.74E-02 (+)	1.10E+00 ± 2.34E-16 (≈)	1.10E+00 ± 2.34E-16
C15	2.36E+01 ± 3.78E+00 (+)	2.40E+00 ± 3.01E-05 (+)	1.23E+01 ± 3.10E+00 (+)	1.78E+01 ± 3.00E+00 (+)	5.81E+00 ± 9.93E-01 (+)	3.23E-09 ± 5.78E+00
C16	5.41E+01 ± 4.07E+01 (+)	0.00E+00 ± 0.00E+00 (≈)	0.00E+00 ± 0.00E+00 (≈)	2.53E+02 ± 1.62E+01 (+)	6.28E+00 ± 5.16E-05 (+)	0.00E+00 ± 0.00E+00
C20	1.53E+01 ± 4.27E-01 (+)	3.68E+00 ± 5.09E-01 (+)	1.42E+01 ± 4.23E-01 (+)	3.20E+00 ± 1.43E-01 (−)	4.07E+00 ± 5.59E-01 (+)	3.34E+00 ± 0.00E+00
C21	1.92E+01 ± 1.17E+01 (−)	▽ (+)	5.26E+01 ± 1.77E+01 (+)	6.29E+01 ± 1.59E+00 (+)	1.84E+01 ± 1.07E+01 (−)	2.98E+01 ± 0.00E+00
C22	9.49E+03 ± 5.22E+03 (+)	9.30E+00 ± 3.41E+00 (+)	▽ (+)	8.39E+03 ± 2.61E+04 (+)	2.48E+01 ± 2.63E+01 (+)	2.74E+00 ± 0.00E+00
C23	1.10E+00 ± 2.05E-03 (≈)	1.60E+00 ± 2.32E-01 (+)	1.14E+00 ± 2.14E-02 (+)	1.34E+00 ± 6.16E-02 (+)	1.44E+00 ± 2.49E-01 (+)	1.10E+00 ± 1.79E-01
C24	7.17E+00 ± 3.54E+00 (+)	2.40E+00 ± 1.88E+00 (+)	3.61E+00 ± 2.03E+00 (+)	1.43E+01 ± 1.28E+00 (+)	5.18E+00 ± 9.93E-01 (+)	1.37E-12 ± 2.14E+01
C25	5.94E+01 ± 3.40E+01 (+)	5.30E-07 ± 3.24E-01 (+)	2.31E+01 ± 1.72E+01 (+)	2.49E+02 ± 1.58E+01 (+)	6.28E+00 ± 4.83E-05 (+)	0.00E+00 ± 1.71E+00
(+)	19	18	20	14	17	/
(−)	1	2	0	7	3	/
(≈)	2	2	2	1	2	/

In the case of 50D, Table III.10 shows that ACS-CDE has the best performance in most cases. Particularly it surpasses the ECO-HTC, CORCO, ITLBO, LSHADE44 and DeCODE on 19, 18, 20, 14 and 17 test functions, respectively. On the other hand, ACS-CDE performs worse than them on 1, 2, 0, 7 and 3 respectively. The average ranking of the six approaches on the same experiment according to the Friedman's test in shown in Table III.8, where ACS-CDE is ranked far less than other competitors (2.0455) making it in the top rank. The multiple-problem Wilcoxon test results for 22 functions with 50D are displayed in Table III.9. It is evident that the R+ values are higher than the R- values in every case. Additionally, it is shown that significant differences occur in every scenario where $\alpha = 0.1$ and 0.05.

As shown in Table III.13, in the case of 100D, ACS-CDE performs better than ECO-HTC, CORCO, ITLBO, LSHADE44 and DeCODE on 20, 17, 20, 14, 19 test functions while these competitors provide better results on one, two, zero, seven and two test functions. Therefore, this comparison verifies that ACS-CDE demonstrates superior overall performance on high-dimensional test functions compared to all the aforementioned algorithms. Table III.11 presents the average ranking of the six approaches according to the Friedman test on 100D. ACS-CDE achieves a significantly lower (1.6667) rank compared to the other five algorithms, achieving the top rank. Additionally, Table III.12 displays the multiple-problem Wilcoxon test results on 21 functions. In all instances, the R+ values are larger than the R- values. Furthermore, significant differences are observed in all cases when $\alpha = 0.1$ and $\alpha = 0.05$.

Algorithm	**Ranking**
ACS-CDE	**1.6667**
CORCO	2.8571
DeCODE	3.3333
LSHADE44	3.5238
ECO-HTC	4.5238
ITLBO	5.0476

Table III.11: Friedman's test ranking on IEEE CEC2017 100D

Vs	R^+	R^-	p-value	$\alpha = 0.1$	$\alpha = 0.05$
ECO-HTC	218.0	13.0	8.392E-5	Yes	Yes
CORCO	197.5	33.5	3.068E-3	Yes	Yes
ITLBO	210.0	0.0	1.9074E-6	Yes	Yes
LSHADE44	186.0	45.0	1.2692E-2	Yes	Yes
DeCODE	196.0	35.0	3.752E-3	Yes	Yes

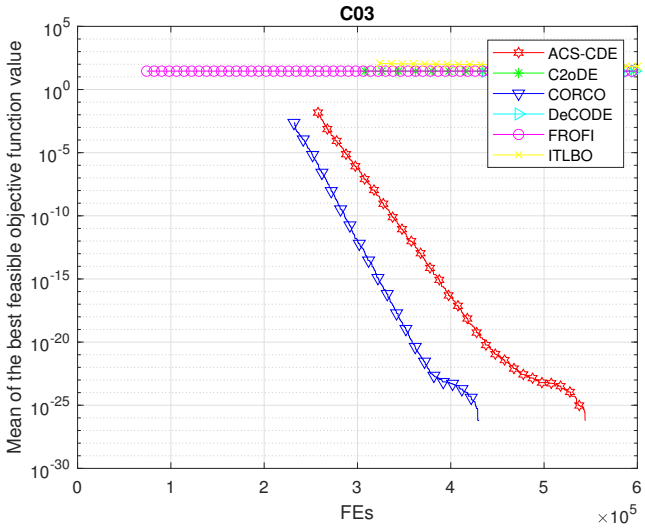
Table III.12: Wilcoxon test on 21 IEEE CEC2017 100D Functions

Table III.13: The results obtained by ACS-CDE and comparison methods on IEEE CEC2017 test suite with 100D.

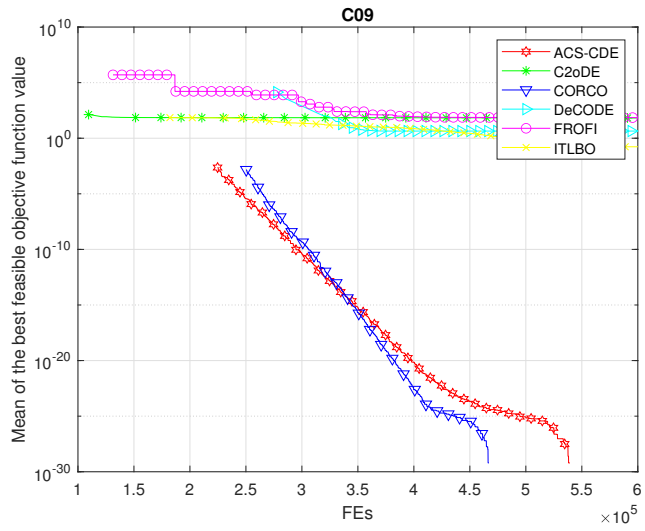
CEC2017 D100	ECO-HTC Mean ± STD	CORCO Mean ± STD	ITLBO Mean ± STD	LSHADE44 Mean ± STD	DeCODE Mean ± STD	ACS-CDE Mean ± STD
C01	1.05E-05 ± 1.08E-05 (+)	5.76E-12 ± 2.74E-12 (+)	1.34E-04 ± 1.40E-04 (+)	1.03E-25 ± 1.68E-25 (−)	3.35E-09 ± 3.62E-09 (+)	7.89E-16 ± 1.47E-15
C02	4.58E-06 ± 5.25E-06 (+)	6.09E-12 ± 4.78E-12 (+)	1.08E-04 ± 9.30E-05 (+)	8.47E-26 ± 1.25E-25 (−)	1.01E-09 ± 7.75E-10 (+)	6.24E-16 ± 7.89E-16
C03	2.18E-05 ± 1.71E-05 (+)	1.50E-11 ± 1.45E-11 (+)	1.56E-04 ± 1.22E-04 (+)	2.73E+06 ± 6.39E+04 (+)	3.05E-09 ± 1.96E-09 (+)	2.90E-16 ± 2.61E-16
C04	3.23E+02 ± 3.96E+01 (+)	6.97E+01 ± 9.15E+00 (+)	9.44E+02 ± 4.98E+01 (+)	1.40E+01 ± 7.88E-01 (−)	8.87E+01 ± 2.90E+01 (+)	6.39E+01 ± 3.37E+01
C05	8.84E+01 ± 2.50E+01 (+)	2.55E+00 ± 3.45E+00 (+)	2.14E+01 ± 2.32E+01 (+)	3.28E-05 ± 9.25E-05 (+)	4.39E-01 ± 1.07E+00 (+)	8.33E-15 ± 1.44E-14
C06	▽ (+)	7.67E+02 ± 8.05E+01 (+)	▽ (+)	▽ (+)	1.53E+03 ± 3.43E+02 (+)	5.93E+02 ± 4.09E+01
C07	2.81E+02 ± 0.00E+00 (+)	-2.33E+02 ± 2.24E+02 (+)	▽ (+)	-3.02E+02 ± 1.35E+02 (+)	-1.15E+02 ± 3.14E+02 (+)	-1.66E+03 ± 1.32E+02
C08	4.09E-03 ± 4.90E-04 (+)	1.78E-03 ± 2.24E-04 (+)	4.10E-03 ± 1.72E-03 (+)	-4.81E-05 ± 1.33E-07 (−)	1.70E-03 ± 4.90E-04 (+)	8.36E-04 ± 1.42E-04
C09	2.17E+00 ± 2.55E+00 (+)	1.31E-07 ± 2.20E-07 (+)	4.58E+00 ± 1.13E+00 (+)	-1.43E-03 ± 2.20E-20 (−)	2.27E-08 ± 8.80E-08 (+)	0.00E+00 ± 0.00E+00
C10	1.32E-03 ± 2.43E-04 (+)	4.51E-04 ± 6.02E-05 (+)	1.14E-03 ± 3.82E-04 (+)	-1.72E-05 ± 1.29E-08 (−)	4.02E-04 ± 2.15E-04 (+)	1.63E-04 ± 2.36E-05
C12	1.59E+01 ± 6.83E+00 (+)	6.42E+00 ± 6.78E+00 (+)	6.79E+01 ± 1.28E+01 (+)	3.25E+01 ± 8.19E-01 (+)	2.10E+01 ± 8.82E+00 (+)	5.03E+00 ± 0.00E+00
C13	2.12E+05 ± 2.10E+05 (+)	3.21E+01 ± 4.54E+02 (−)	▽ (+)	8.07E+01 ± 7.37E+00 (+)	3.85E+01 ± 6.04E-01 (+)	3.44E+01 ± 9.79E-06
C14	8.42E-01 ± 3.69E-02 (+)	1.12E+00 ± 9.44E-03 (+)	1.02E+00 ± 1.24E-01 (+)	9.72E-01 ± 1.94E-02 (+)	9.93E-01 ± 1.66E-01 (+)	7.84E-01 ± 1.81E+00
C15	2.87E+01 ± 3.90E+00 (+)	1.62E+01 ± 1.72E+00 (+)	1.35E+01 ± 3.86E+00 (+)	1.81E+01 ± 1.28E+00 (+)	1.20E+01 ± 8.11E-01 (+)	5.12E+00 ± 0.00E+00
C16	1.21E+02 ± 6.94E+01 (+)	0.00E+00 ± 0.00E+00 (≈)	0.00E+00 ± 0.00E+00 (≈)	5.35E+02 ± 3.08E+01 (+)	6.28E+00 ± 5.16E-05 (+)	0.00E+00 ± 7.82E+00
C20	3.50E+01 ± 2.66E+00 (+)	8.04E+00 ± 7.85E-01 (+)	3.35E+01 ± 9.44E-01 (+)	9.36E+00 ± 3.78E-01 (+)	8.51E+00 ± 1.16E+00 (+)	7.31E+00 ± 0.00E+00
C21	5.84E+00 ± 3.28E+00 (−)	▽ (+)	5.39E+01 ± 1.79E+01 (+)	3.16E+01 ± 2.99E-03 (−)	8.69E+00 ± 9.57E+00 (−)	5.24E+01 ± 0.00E+00
C22	▽ (+)	9.82E+01 ± 4.02E+01 (+)	▽ (+)	▽ (+)	2.50E+02 ± 1.72E+02 (+)	8.63E+01 ± 0.00E+00
C23	7.85E-01 ± 7.69E-04 (+)	7.90E-01 ± 1.19E-02 (+)	8.10E-01 ± 5.28E-03 (+)	9.69E-01 ± 4.26E-02 (+)	1.03E+00 ± 1.46E-01 (+)	7.84E-01 ± 0.00E+00
C24	1.13E+01 ± 1.96E+00 (+)	2.36E+00 ± 0.00E+00 (≈)	1.34E+01 ± 2.24E+00 (+)	1.71E+01 ± 1.43E+00 (+)	5.92E+00 ± 1.11E+00 (+)	2.36E+00 ± 7.35E-01
C25	2.37E+02 ± 6.06E+01 (+)	1.45E+01 ± 1.07E+01 (−)	2.02E+02 ± 3.58E+01 (+)	5.44E+02 ± 2.86E+01 (+)	3.10E+01 ± 1.53E+01 (−)	1.89E+02 ± 1.74E+01
(+)	20	17	20	14	19	/
(−)	1	2	0	7	2	/
(≈)	0	2	1	0	0	/

III.4.4 Convergence analysis

To illustrate the convergence behavior of the proposed ACS-CDE algorithm compared to other algorithms, we employed several benchmark tests functions from the CEC2010 and CEC2017 suites. Only the convergence curves of the comparison algorithms which we could obtain their source codes are plotted.



(a)



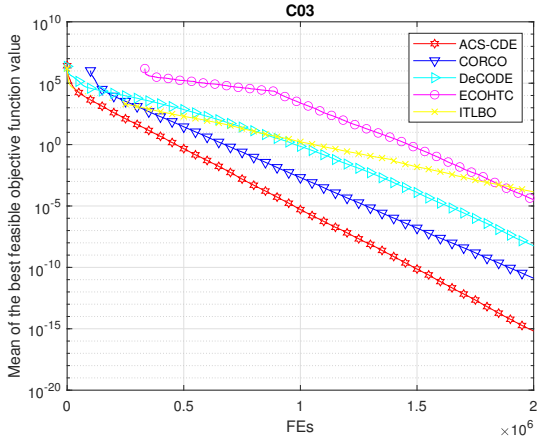
(b)

Figure III.6: Convergence graphs of ACS-CDE and comparison algorithms on two representative CEC2010 functions with 30D.

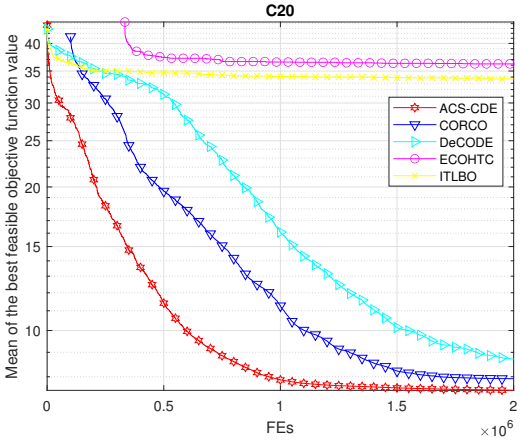
Figure. III.6 presents the convergence curves for two representative 30D problems from the CEC2010 benchmark: C03 and C09. The ACS-CDE algorithm exhibits a convergence rate comparable to that of the CORCO algorithm on both test functions. Notably, CORCO performs slightly better than ACS-CDE and significantly outperforms other al-

gorithms, which tend to become trapped in local optima. This prove the ACS-CDE's robustness in escaping local minima and achieving superior optimization performance.

In Figure. III.7, we show the convergence curves for two representative 50D problems from the CEC2017 benchmark: C03 and C25. The results clearly indicate that ACS-CDE maintains a substantial advantage over competing algorithms throughout the entire optimization process. Figure. III.8 illustrates the convergence behavior on high-dimensional (100D) problems C03 and C09 from the CEC2017 benchmark. Here, ACS-CDE outperforms other algorithms significantly from the early stages of optimization through to the later stages. This indicates that the proposed ACS-CDE scales effectively with dimensionality, maintaining its optimization efficiency and effectiveness even as the problem complexity increases.



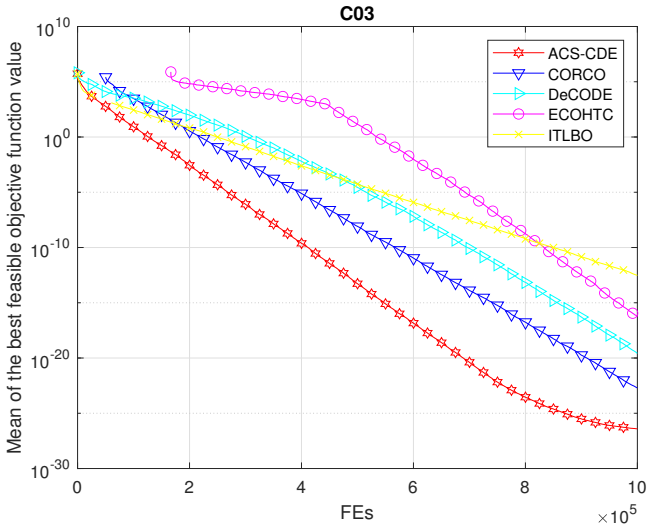
(a)



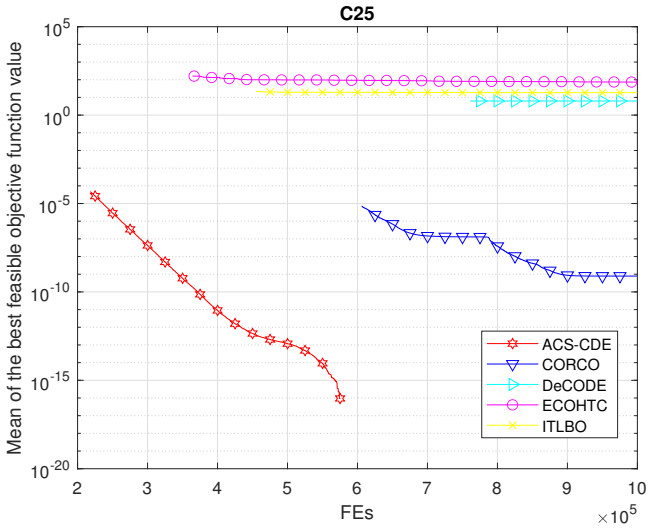
(b)

Figure III.8: Convergence graphs of ACS-CDE and comparison algorithms on two representative CEC2017 functions with 100D.

Overall, the convergence results highlight the competitive performance of ACS-CDE



(a)



(b)

Figure III.7: Convergence graphs of ACS-CDE and comparison algorithms on two representative CEC2017 functions with 50D.

against other advanced optimization methods on both CEC2010 and CEC2017 benchmark problems. Particularly, ACS-CDE shows a marked improvement in high-dimensional scenarios, where it consistently achieves better convergence than its counterparts. This demonstrates the capability of our proposed ACS-CDE to handle complex, high-dimensional optimization problems effectively.

III.4.5 Effectiveness of Eigen coordinate system

To demonstrate the effectiveness of incorporating the Eigen coordinate system based on the information of the constraint violation and the objective function, we implemented two variants of our proposed approach i.e. ACSCDE-objective and ACSCDE-constraint. In the ACSCDE-objective, we only use the objective Eigen coordinate system and for the ACSCDE-constraint only the constrained Eigen coordinate system is incorporated, while the original system is removed from both variants. Additionally, to verify the effectiveness of the adaptive selection of multiple coordinate system, a third variant noted as ACSCDE-Random was implemented, where the proposed adaptive system selection scheme is replaced with a random selection method of the three coordinate system. The rest of other components for each variant are kept the same as in the original ACSCDE.

The three algorithm variants were assessed using the 18 (30D) test functions from the IEEE CEC2010 suite. Their performance compared with that of ACS-CDE in table III.16. As shown in the Table III.16 , the variants ACSCDE-objective, ACSCDE-constraint, ACSCDE-random obtain inferior results compared with ACSCDE in most test functions where they only performed better in 5, 3, 3 out of 18 cases respectively. Also, it is observed that ACSCDE-objective cannot find a feasible solution consistently in two test functions, those are, C11 and C12 with 30D. The performance of ACSCDE -Constraint and ACSCDE -Objective on CEC 2010 with 30D reveals their respective strengths and weaknesses of each system.

Algorithm	**Ranking**
ACS-CDE	2.0000
ACSCDE-Random	2.2778
ACSCDE-Objective	2.6389
ACSCDE-Constraint	3.0833

Table III.14: Friedman's test of the ACS-CDE and its variants on IEEE CEC2010 with 30D.

VS	R^+	R^-	p-value
ACSCDE-Objective	96.0	57.0	≥ 0.2
ACSCDE-Constraint	137.5	15.5	2.334E-3
ACSCDE-Random	105.0	48.0	1.9E-1

Table III.15: Results obtained by the Wilcoxon test for algorithm ACSCDE and its variants on IEEE CEC2010 with 30D.

IEEE CEC2010 D30	ACSCDE-Objective Mean ± STD	ACSCDE-Constraint Mean ± STD	ACSCDE-Random Mean ± STD	ACSCDE Mean ± STD
C01	-8.20E-01 ± 1.99E-03 (+)	-8.20E-01 ± 2.50E-03 (+)	-8.22E-01 ± 1.12E-03 (≈)	-8.22E-01 ± 1.08E-03
C02	-2.27E+00 ± 7.98E-03 (−)	-2.23E+00 ± 4.07E-02 (−)	-2.21E+00 ± 5.57E-02 (−)	-2.19E+00 ± 5.40E-02
C03	0.00E+00 ± 0.00E+00 (≈)	2.52E+01 ± 1.45E+01 (+)	0.00E+00 ± 0.00E+00 (≈)	0.00E+00 ± 0.00E+00
C04	-3.33E-06 ± 2.45E-09 (−)	-3.33E-06 ± 4.36E-09 (−)	-3.31E-06 ± 2.06E-08 (−)	-3.15E-06 ± 4.93E-07
C05	-4.84E+02 ± 6.84E-12 (≈)	-7.91E+01 ± 2.84E+02 (+)	-4.76E+02 ± 1.11E-07 (+)	-4.84E+02 ± 9.76E-06
C06	-5.31E+02 ± 4.66E-01 (≈)	-3.96E+02 ± 1.17E+02 (+)	-5.31E+02 ± 1.05E-01 (≈)	-5.31E+02 ± 5.43E-02
C07	0.00E+00 ± 0.00E+00 (≈)	6.30E-01 ± 5.80E+00 (+)	0.00E+00 ± 0.00E+00 (≈)	0.00E+00 ± 0.00E+00
C08	1.37E+01 ± 3.37E+01 (+)	5.38E+01 ± 1.02E+02 (+)	3.27E+00 ± 1.63E+01 (+)	0.00E+00 ± 0.00E+00
C09	5.28E-01 ± 1.46E+00 (+)	3.52E-01 ± 1.22E+00 (+)	0.00E+00 ± 0.00E+00 (≈)	0.00E+00 ± 0.00E+00
C10	1.58E+00 ± 2.17E+00 (−)	3.49E+01 ± 1.77E+01 (+)	2.26E+01 ± 8.20E+00 (+)	1.62E+01 ± 8.20E+00
C11	▽ (+)	-1.48E-04 ± 3.79E-03 (+)	-3.92E-04 ± 4.91E-02 (≈)	-3.92E-04 ± 2.96E-08
C12	▽ (+)	-1.99E-01 ± 2.29E-08 (≈)	-1.99E-01 ± 4.42E-08 (≈)	-1.99E-01 ± 3.71E-07
C13	-6.71E+01 ± 1.44E+00 (+)	-6.77E+01 ± 1.16E+00 (+)	-6.80E+01 ± 4.38E-01 (+)	-6.81E+01 ± 5.75E-01
C14	1.98E+01 ± 9.94E+01 (+)	0.00E+00 ± 0.00E+00 (≈)	0.00E+00 ± 0.00E+00 (≈)	0.00E+00 ± 0.00E+00
C15	2.54E+01 ± 7.85E+02 (+)	2.18E+01 ± 1.14E+00 (+)	2.18E+01 ± 2.33E-08 (+)	2.07E+01 ± 2.65E-08
C16	0.00E+00 ± 0.00E+00 (≈)	0.00E+00 ± 0.00E+00 (≈)	0.00E+00 ± 0.00E+00 (≈)	0.00E+00 ± 0.00E+00
C17	1.56E+01 ± 2.58E+01 (−)	8.11E+02 ± 1.06E+03 (+)	4.15E+02 ± 6.91E+02 (+)	2.68E+02 ± 8.75E+02
C18	2.89E-29 ± 9.63E+01 (−)	1.10E-25 ± 5.52E-25 (−)	4.67E-25 ± 3.39E-23 (−)	4.21E-22 ± 3.22E-20
(+)	8	12	6	/
(−)	5	3	3	/
(≈)	5	3	9	/

Table III.16: Experimental results of ACSCDE-Objective, ACSCDE-Constraint, ACSCDE-Random and original ACSCDE on CEC2010 with 30D.

Furthermore, Table III.14-III.15 present a statistical significant results of the Friedman's test and the Wilcoxon's signed rank test, respectively, it can be seen the ACSCDE can get the best results compared with its variants. this imply that utilizing two coordinate systems by leveraging from the information of the objective and constraint function is very important to improve the performance and achieving the trade-off between constraint and objective. Also, by comparing with ACSCDE-random we find that properly utilizing the Eigen coordinate systems is very crucial to better adapt to the complexity of various test functions.

The convergence performance of ACS-CDE and its variants across four test functions (C03, C07, C13, C16) is depicted in Fig.III.9. The figure demonstrates that the convergence rate of ACSCDE can sometimes lag behind the other variants due to its consideration of both objective function and constraint violation information during the optimization process. However, ACSCDE consistently manages to identify the optimal solution in the later stages.

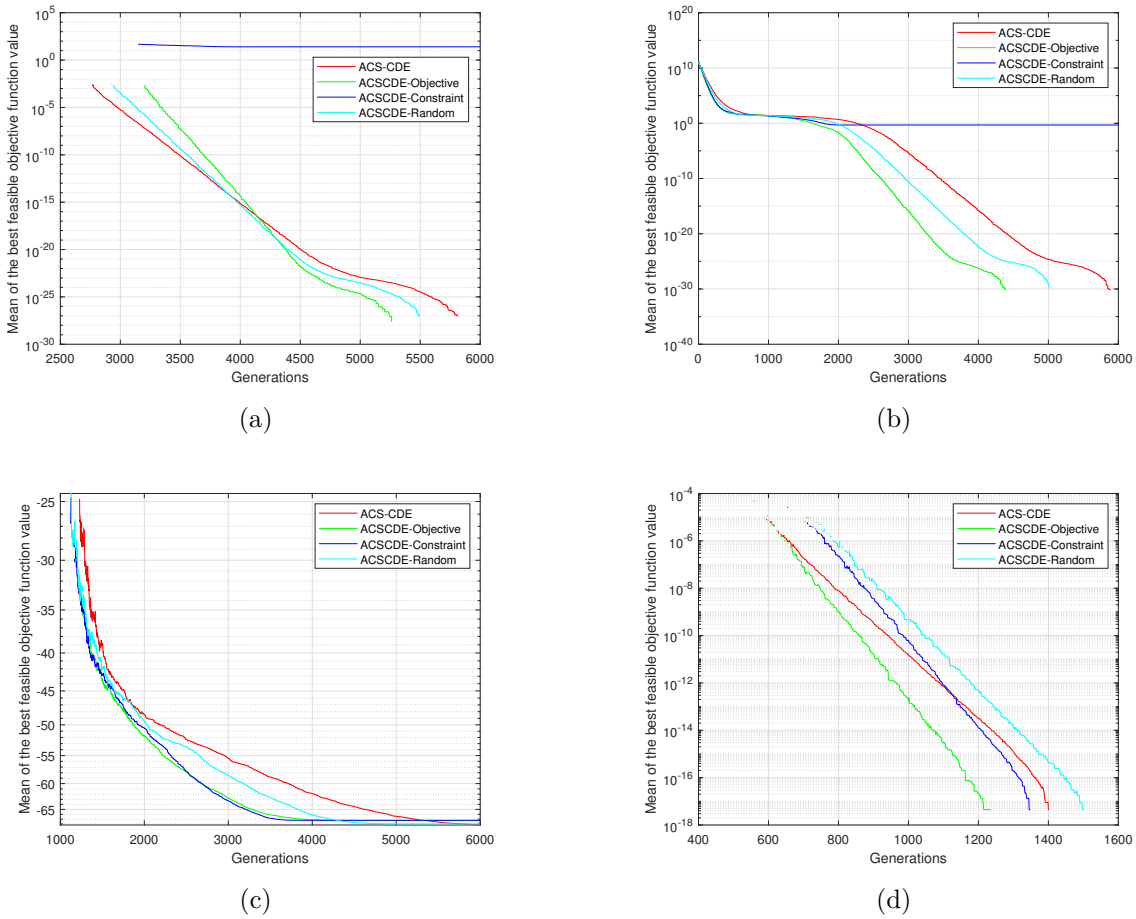


Figure III.9: The convergence performance of ACS-CDE and its variants on CEC2010 30D: (a) C03, (b) C07, (c) C13, (d) C16.

In summary, the aforementioned experimental findings reveal that designing an adap-

tive dynamic coordinate system based on the characteristics of the COPs in term of objective and constraint function information is effective in dealing with COPs with complex constraints and COPs with complex objective.

III.4.6 Diversity analysis

In this section, we present the population diversity curve for the proposed ACS-CDE algorithm, as illustrated in Figure III.10. This figure displays the evolution of diversity throughout the optimization process, comparing the proposed algorithm with its variants that has been discussed in the previous section III.4.5.

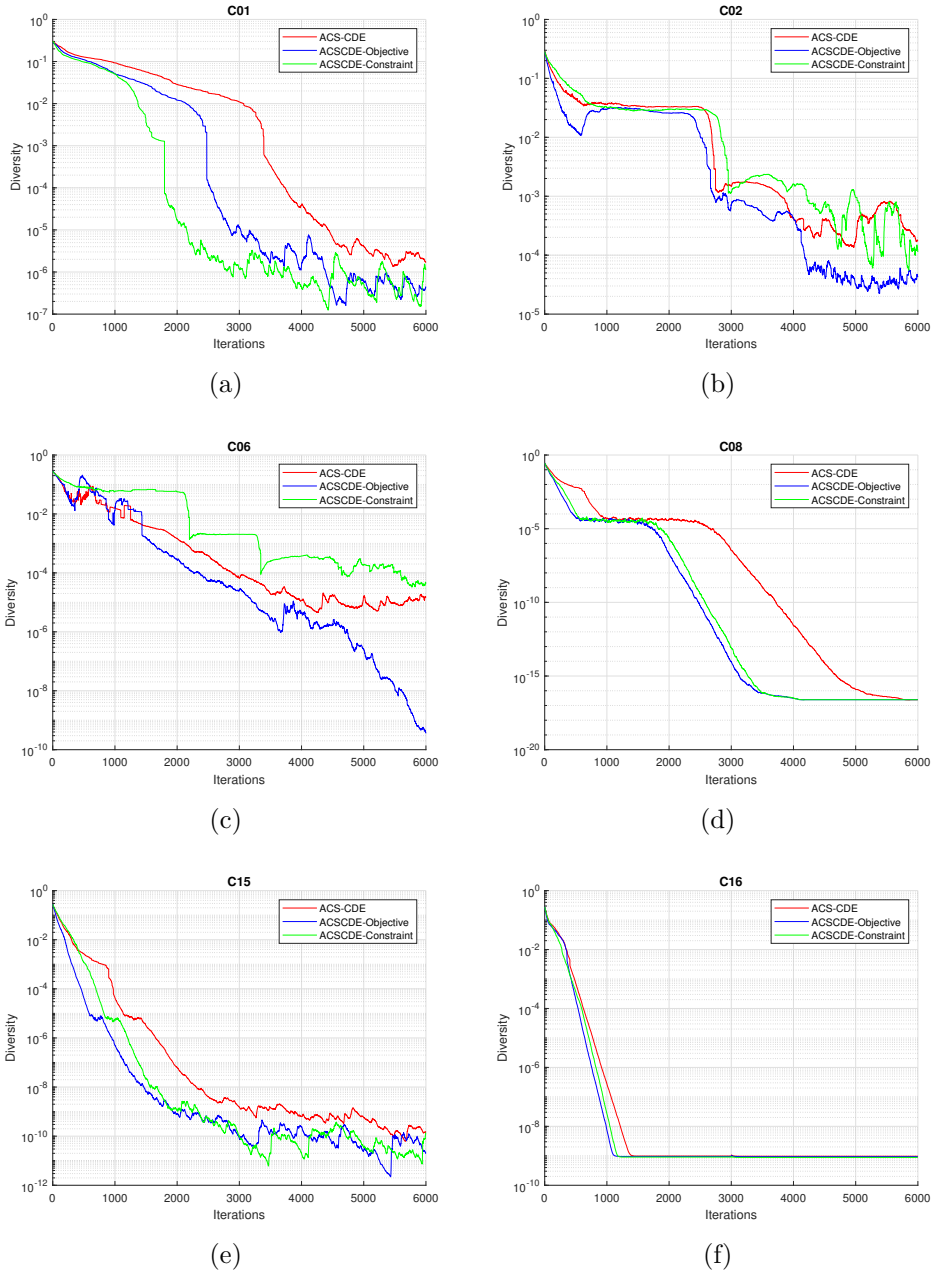


Figure III.10: Population diversity of ACS-CDE and its variants on CEC2010-30D functions.

Diversity is calculated using the distribution state D_s of the population, defined as follows:

$$D_s = \sum_{i=1}^{D} \text{std}(x_1(i), \ldots, x_{NP}(i)), \tag{III.17}$$

where D represents the number of dimensions and NP is the population size. This metric is commonly used as an indicator of population diversity during evolution Wang et al (2020).

The performance of the diversity in our proposed algorithm, which utilizes the original coordinate system combined with a mutation strategy that emphasizes diversity, is particularly notable when compared to variants that do not incorporate the original system Fig III.10. This integration enhances the algorithm's ability to maintain diversity, thereby improving its performance during the optimization process.

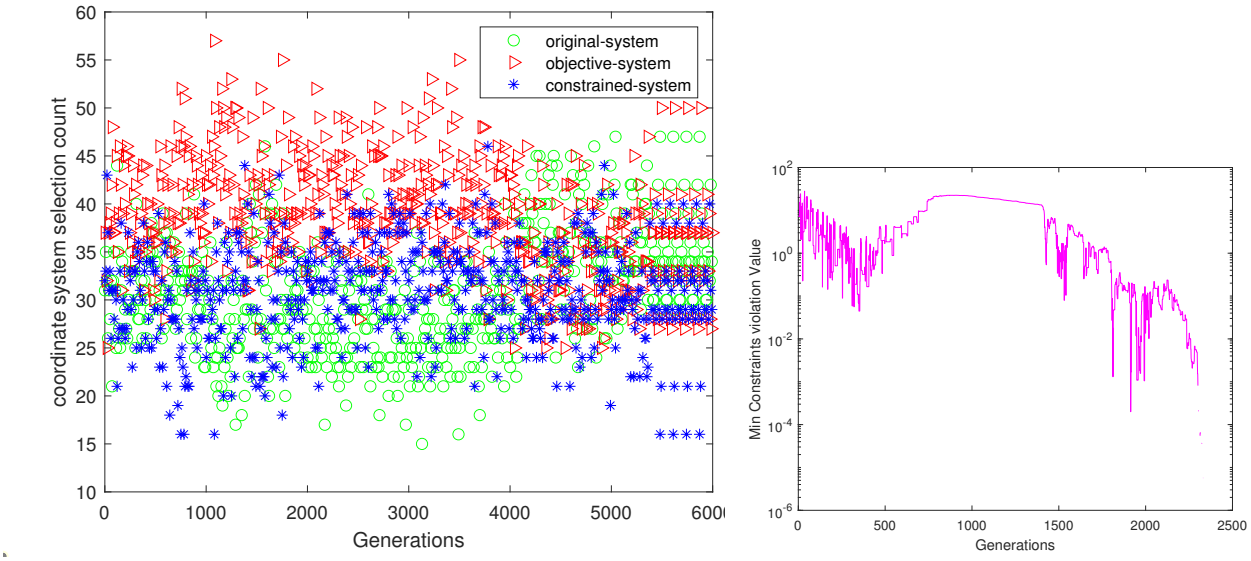
It is crucial to recognize that the nature of the optimization problem significantly influences diversity outcomes. For simpler problems, the algorithm may converge quickly in the initial stages, which could lead to a decrease in diversity. This rapid convergence does not inherently signify poor performance; rather, it may indicate efficient optimization.

III.4.7 Effectiveness of the adaptive selection mechanism

To gain a deeper understanding of the behavior of adaptive selection among the three-coordinate systems, we recorded the selection count of each system across the entire population during the search process. We used two representative test functions, C09 and C12, each with 10 D from the CEC2010, to investigate the effectiveness of the adaptive selection mechanism.

For the C09 function Fig. III.11, the data indicates that the objective-oriented system dominates the selection process throughout most of the search, particularly in the early stages. This suggests that the objective-oriented system is more effective for the C09 test function, delivering superior performance during the initial exploration.

In contrast in Fig. III.12, for the C12 function, the system selection process observed three distinct phases. During the initial phase (first 2000 generations), none of the coordinate system consistently dominates and systems are selected almost equally. In the second phase (second 2000 generations), the constrained-oriented system clearly prevails, indicating its superior performance at this stage by guiding the search towards feasible regions. In the final phase (last 2000 generations), as the search process enters the feasible region, the selection probabilities for the original and constrained systems are converged, while the dominance of the objective system increases.
Additionally, it is observed that throughout the entire process, even less effective systems are continuously explored, demonstrating a balance between exploration and exploitation.



(a) (b)

Figure III.11: Evolution of ACS-CDE on C09 with 10D: (a) coordinate system selection count, (b) min degree of constraint violation

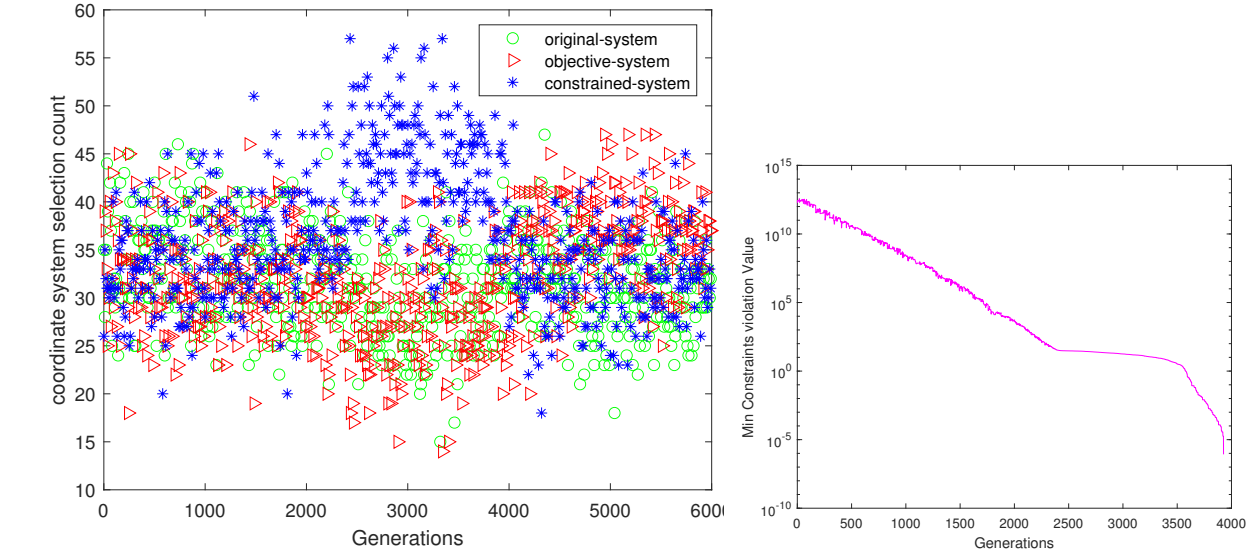


Figure III.12: Evolution of ACS-CDE on C12 with 10D: (a) coordinate system selection count, (b) min degree of constraint violation

These findings imply that the proposed adaptive system effectively selects the most suitable coordinate system in a manner that is contextually appropriate for different stages of the search. The observed selection dynamics are influenced primarily by the complexity of the objective and constraint functions, which dictate the selection rates of each system. It is essential to explore the search space extensively in the initial stages to understand the landscape of both the objective and constraints and to focus on exploitation in later stages. Notably, the epsilon-constrained method employed in this research allow using more information from the objective function at the initial stage.

III.5 Conclusion

This chapter introduced ACS-CDE, an adaptive coordinate system for constrained differential evolution, to address the balance between objective optimization and constraint satisfaction in constrained optimization problems (COPs). By utilizing two dynamic coordinate systems—one guiding the population toward optimal solutions and the other toward feasible regions. The coordinate system constructed using an archive-based covariance matrix. Moreover, an adaptive selection process for the coordinate systems based on the multi-armed bandit is developed. The algorithm also incorporates an epsilon constraint-handling technique and a restart mechanism to address complex constraints.

Our experimental results on the CEC2010 and CEC2017 benchmark test suites demonstrate that ACS-CDE is effective in solving complex COPs, outperforming several advanced state-of-the-art techniques. The dynamic coordinate system framework enables ACS-CDE to adaptively navigate the fitness landscape, striking a balance between exploring feasible regions and optimizing the objective function. However, ACS-CDE has some limitations. The algorithm may experience slower convergence on certain COPs, especially those with tightly constrained or highly complex landscapes. Additionally, the covariance matrix updates can lead to increased computational overhead, which could limit the algorithm's scalability to larger, real-world problems with high-dimensional search spaces.

To address these limitations, future work could focus on improving ACS-CDE's computational efficiency, such as reducing the frequency of covariance matrix updates and enhancing convergence speed in highly constrained problems. We also plan to extend ACS-CDE to handle multi-objective optimization, balancing multiple conflicting objectives and constraints, and apply it to expensive optimization problems where evaluations are costly. Incorporating surrogate models or reducing the number of evaluations could make ACS-CDE more applicable to real-world engineering and scientific challenges.

Chapter IV
Grey Wolf Optimizer for TSP

Chapter IV
Grey Wolf Optimizer for TSP

IV.1 Introduction

In this chapter, we explore the practical application of bio-inspired algorithms, focusing specifically on their adaptation for discrete optimization problems. Building on the theoretical foundations of constrained optimization and bio-inspired methodologies established in previous chapters, we now apply these concepts to address the Traveling Salesman Problem (TSP), a classic NP-hard challenge. The TSP serves as a crucial benchmark for assessing optimization algorithms' effectiveness in tackling complex combinatorial problems.

Given the proven success of bio-inspired algorithms in continuous optimization, we propose an enhanced variant of the Grey Wolf Optimizer (GWO) adapted for discrete optimization tasks. Through the integration of 2-opt and swap mechanisms, in addition to a proposed greedy approach for solution construction, we illustrate how these adaptations enable effective solutions to the TSP. This chapter demonstrates the versatility and robustness of bio-inspired algorithms in discrete problem domains, supporting the broader objectives of this thesis: advancing optimization techniques and showcasing their applicability across diverse problem contexts.

IV.2 Literature Review

The Grey Wolf Optimizer (GWO) is an effective meta-heuristic algorithm known for its well-balanced convergence between exploration and exploitation during the search process. Due to this capability, GWO has attracted significant research interest and has been successfully applied across various domains, including engineering optimization (Nadimi-Shahraki et al, 2021), image processing (Li et al, 2017), clustering (Tripathi

et al, 2018), path planning (Qu et al, 2020), power dispatch (Sulaiman et al, 2015), and scheduling (Komaki and Kayvanfar, 2015). Given the complexity inherent to these applications, numerous GWO variants have emerged to address specific challenges.

For instance, an improved GWO (I-GWO) tailored for engineering design optimization is proposed in (Nadimi-Shahraki et al, 2021), where a dimension-learning hunting (DLH) strategy is implemented to enhance the balance between exploration and exploitation. In another study, chaotic maps and Opposition-Based Learning (OBL) are applied to GWO for improved initial population diversity (Ibrahim et al, 2018). In multi-objective optimization, Mirjalili et al (2016b) developed a multi-objective version of GWO (MOGWO) with an external archive to support solution diversity.

For discrete optimization challenges, researchers have developed specialized GWO adaptations. For example, Lu et al (2016) introduced a multi-objective discrete GWO for real-world scheduling in welding processes, using permutation vectors and a machine assignment matrix for encoding. In image processing, a modified discrete GWO using rounding and weighted updating was employed for multiple image thresholding (Li et al, 2017). Additionally, Komaki and Kayvanfar (2015) utilized a largest-position-value approach in GWO for a two-stage assembly flow shop scheduling problem. Binary GWO adaptations, such as the one by Emary et al (2016) for feature selection, use binary steps with crossover and sigmoidal functions for position updates, while Zhang et al (2016) applied GWO to a 2D path planning problem for unmanned aerial vehicles.

To further enhance GWO's performance, researchers have hybridized it with various techniques. For instance, Tripathi et al (2018) combined GWO with Map-Reduce and enhanced it with binomial crossover and Lévy flights for clustering tasks, while Jayabarathi et al (2016) used crossover and mutation operators to solve economic dispatch problems. Another approach, GWO-DE, hybridizes GWO with Differential Evolution to address nonlinear systems effectively (Tawhid and Ibrahim, 2020).

This work focuses on developing a novel GWO updating strategy. Various updating strategies have been proposed to improve GWO, such as a weighted average method in (Malik et al, 2015) and an exploration-enhanced GWO (EEGWO) that bases updates on the three best individuals plus a randomly selected individual from the population (Long et al, 2018a). Additionally, Long et al (2018b) applied GWO to large-scale optimization by modifying position updates based on personal historical and global best positions.

Despite the extensive research on GWO adaptations, limited attention has been given to routing problems, particularly the Traveling Salesman Problem (TSP). Panwar and Deep (2021) introduced a discrete GWO for TSP that employs Hamming distance and 2-opt algorithms for the update mechanism, taking the best of three new solutions as the final position rather than the standard average. While effective, this approach does not fully exploit all GWO parameters, which we address in this study through an extended GWO framework with additional algorithm parameters.

The TSP remains a popular benchmark for assessing algorithm performance in discrete optimization. Various approaches have been applied to TSP, including classical methods like Tabu Search (TS) (Knox, 1994), Genetic Algorithms (GA) (Hussain et al, 2017; Moon et al, 2002), Simulated Annealing (SA) (Geng et al, 2011), Ant Colony Optimization (Dorigo and Gambardella, 1997), Particle Swarm Optimization (Wang et al, 2003), and Artificial Bee Colony (Karaboga and Gorkemli, 2011). In recent years, novel meta-heuristic algorithms have also emerged, such as Cuckoo Search (Ouaarab et al, 2014), Pigeon-Inspired Optimization (Zhong et al, 2019), Firefly Algorithm (Jati et al, 2011), Harmony Search (Boryczka and Szwarc, 2019), Spider Monkey Optimization (Akhand et al, 2020), Crow Search Algorithm (Al-Gaphari et al, 2021), and Bat Algorithm (Osaba et al, 2016), sparrow search algorithm (Zhang and Han, 2022).

This review highlights key contributions in GWO adaptations and TSP solutions. However, numerous related studies remain beyond the scope of this work. For a broader overview of GWO applications, readers are referred to Faris et al (2018), and for further insights into TSP, see Osaba et al (2020).

IV.3 Problem Statement

The Traveling Salesman Problem (TSP) is one of the most well-known and extensively studied problems in combinatorial optimization literature. As an NP-hard problem, it poses significant computational challenges, making it an attractive case for developing and testing optimization algorithms. This complexity has motivated numerous researchers to devise diverse approaches to solve the TSP efficiently (Lawler et al, 1986).

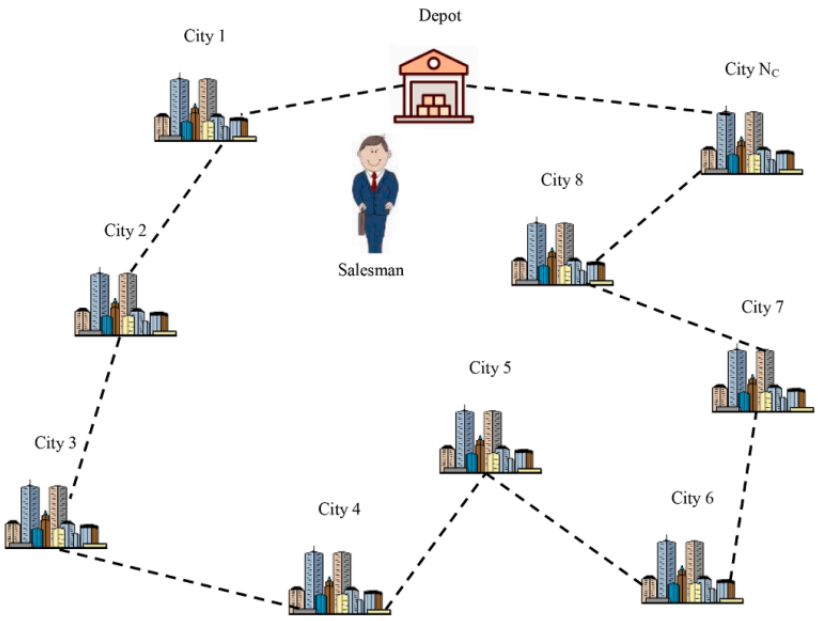


Figure IV.1: Traveling salesman problem schema

The objective of the TSP is to determine the shortest possible path that allows a salesman to visit a set of cities (nodes) exactly once, starting and ending at the same city. Mathematically, the TSP can be represented by a graph $G = (V, E)$, where $V = \{V_1, V_2, \ldots, V_n\}$ represents the set of cities (vertices), and E is the set of arcs (edges) representing the paths between the cities. Each edge in the graph has an associated distance or cost.

Formally, let x_{ij} represent the distance between city i and city j. The goal is to find a Hamiltonian cycle—a closed path that visits each vertex exactly once and minimizes the total travel cost, which can be written as:

$$\min \; Z = \sum_{i=1}^{n} \sum_{j=1}^{n} d_{ij} x_{ij} \tag{IV.1}$$

$$s.t \quad \begin{cases} \sum_{j=1}^{n} x_{ij} = 1, & i \neq j, \quad i \in V \\ \sum_{i=1}^{n} x_{ij} = 1, & i \neq j, \quad j \in V \\ x_{ij} \in \{0, 1\} \end{cases} \tag{IV.2}$$

where x_{ij} is a binary variable that takes the value 1 if the tour travels from city i to city j, and 0 otherwise.

This classical formulation of TSP provides the foundation for more advanced algorithms and solution techniques explored later in this work. The NP-hard nature of the problem necessitates the use of approximation, heuristics, or meta-heuristic methods, especially for larger instances, making TSP a rich field of study in both theoretical and applied optimization research.

IV.4 Proposed greedy discrete GWO algorithm

This section presents our adaptation of the Grey Wolf Optimizer (GWO) for solving the Traveling Salesman Problem (TSP). In addition, we introduce a novel solution construction approach tailored to the discrete nature of the TSP.

As discussed in Chapter 2, the original GWO algorithm is primarily designed for continuous optimization problems. However, both TSP and other similar routing problems are combinatorial in nature, requiring discrete optimization techniques. Therefore, several modifications to the traditional GWO framework are necessary to adapt it effectively for TSP.

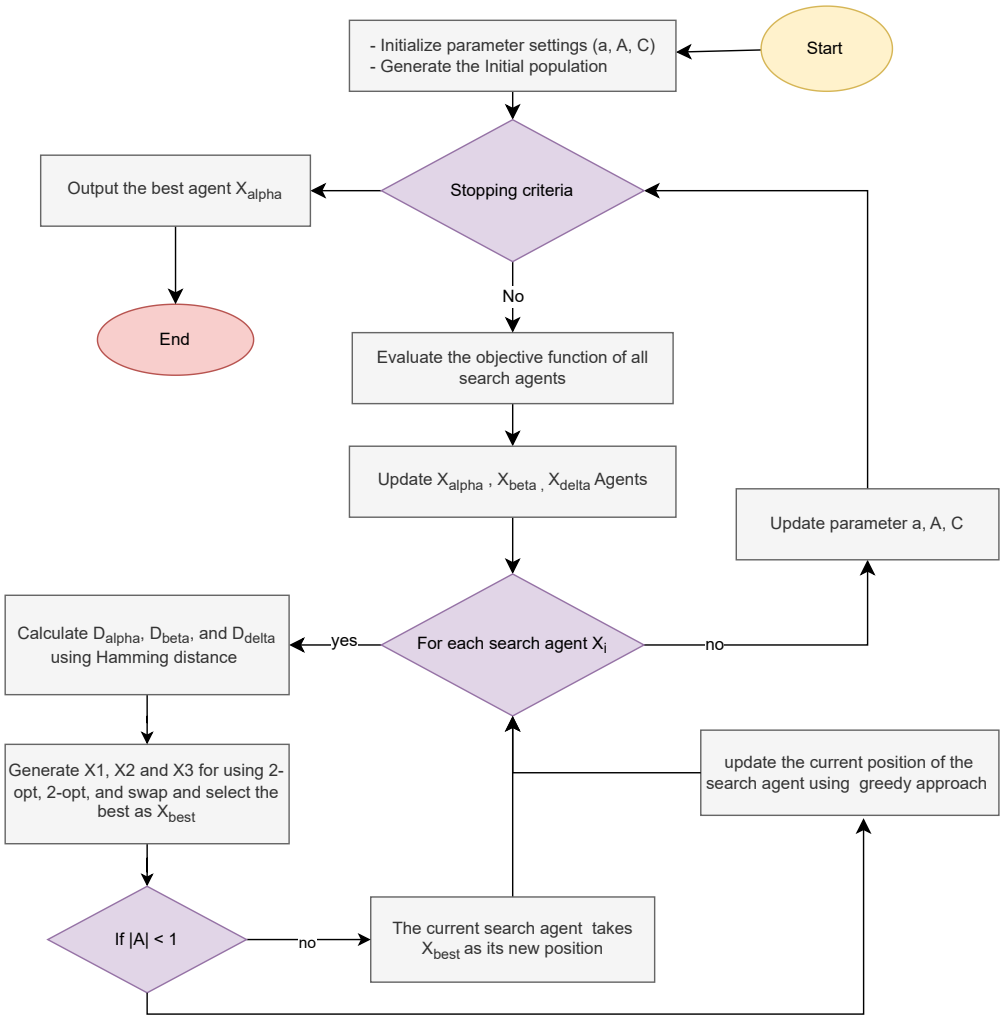


Figure IV.2: Flowchart of GD-GWO

To address these challenges, we propose an algorithm with the following procedure:

IV.4.1 Representation and Initialization

In the proposed algorithm, each wolf in the population represents a feasible solution to the TSP, with the total travel cost of the route serving as the objective function. Each solution is encoded as a permutation of city indices, which dictates the order in which the cities are visited.

During the initialization phase, the population is generated by randomly permuting city indices for each solution. This random initialization helps ensure a diverse starting population, which is essential for thoroughly exploring the search space and reducing the risk of premature convergence.

IV.4.2 Position Update

In the proposed Greedy Discrete Grey Wolf Optimizer (GD-GWO) algorithm, we adapt the original Grey Wolf Optimizer (GWO) for the Traveling Salesman Problem (TSP) by incorporating the 2-opt local search algorithm, swapping move and the concept of Hamming distance. Furthermore we developed a greedy mechanism from the constructing the final solution. The following procedure describe each of this concepts:

1. Hamming distance-based approach

In this modified GWO approach, we measure the difference between two solutions by calculating their Hamming distance, which indicates the number of differing elements between two permutation vectors. For each comparison, only a randomly selected portion of the sequence is analyzed, controlled by a coefficient C representing a fraction of the total number of cities. Specifically, the distance-based difference vectors D_α, D_β, and D_δ are calculated as follows:

$$D_\alpha = \sum_{k \in \text{subseq}} \delta(X_i(k), X_\alpha(k)) \tag{IV.3}$$

$$D_\beta = \sum_{k \in \text{subseq}} \delta(X_i(k), X_\beta(k)) \tag{IV.4}$$

$$D_\delta = \sum_{k \in \text{subseq}} \delta(X_i(k), X_\delta(k)) \tag{IV.5}$$

where:

- subseq is a randomly selected subset of indices within the range $[1, n]$, with size $|\text{subseq}| = C \times n$, where n is the total number of cities.
- $\delta(X_i(k), X_j(k)) = \begin{cases} 1 & \text{if } X_i(k) \neq X_j(k) \\ 0 & \text{if } X_i(k) = X_j(k) \end{cases}$.

For instance, consider two positions in a TSP instance with 10 nodes, as shown in Figure IV.3. Assuming the parameters $C = 5$, $i = 3$, and $j = 8$, the difference vector $\vec{D}$ between $\vec{X_p}$ and $\vec{X}$ would yield a value of 3.

0	1	5	3	6	2	4	8	9	7

0	1	6	3	8	5	4	9	2	7

Figure IV.3: Illustration of two solution vectors difference

2. Updating approach

After calculating the approximate distances D, each candidate solution in the basic GWO updates its position according to alpha, beta, and delta using Eq. II.12. However, this formula cannot be applied directly to the TSP. Therefore this study provides two leading operators for the movement update:

- **2-Opt move**

  2-opt algorithm, introduced by Croes Croes (1958), is a local search technique widely used to improve route quality in TSP and other routing problems. It optimizes the route by iteratively eliminating two randomly chosen edges from the path and reconnecting them to reduce the overall travel cost. This helps accelerate convergence and achieve more accurate solutions. In Figure.IV.4, we can see how the 2-opt move shortens the route by reorganizing connections. An Example of route transformation using 2-opt:

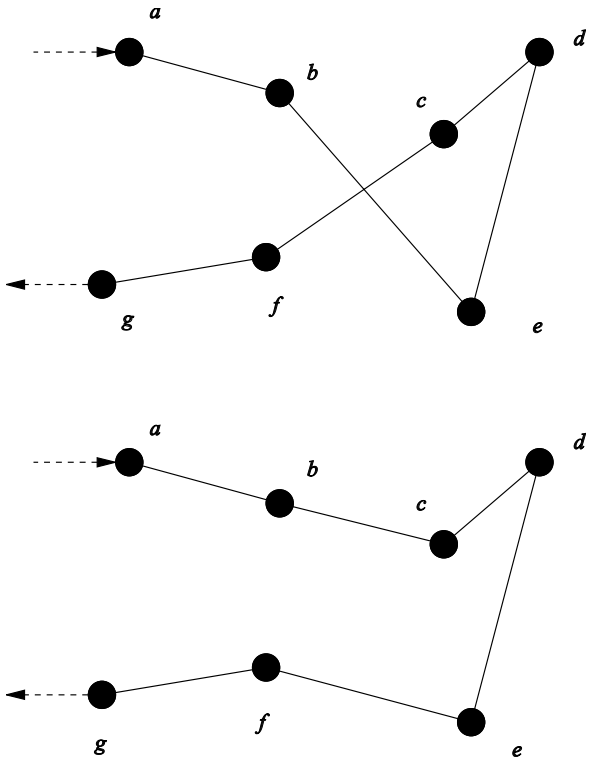


Figure IV.4: Example of 2-opt movement

  - Initial route: $A \rightarrow B \rightarrow \mathbf{E} \rightarrow \mathbf{D} \rightarrow \mathbf{C} \rightarrow F \rightarrow G$
  - After reconnecting, the new route might look like: $A \rightarrow B \rightarrow C \rightarrow D \rightarrow E \rightarrow F \rightarrow G$

  The algorithm continues to refine the route with repeated 2-opt moves, progressively building on improved paths.

- **Swap move**

  The swap move is another local search technique in which two random cities in the current route are swapped to adjust the sequence, potentially leading to a better solution. Example route transformation using swap move between city 2 and 5 in a sample route:

  - Initial route: $A \rightarrow B \rightarrow \mathbf{E} \rightarrow D \rightarrow C \rightarrow \mathbf{F} \rightarrow G$
  - After swapping cities, the new route is: $A \rightarrow B \rightarrow F \rightarrow D \rightarrow C \rightarrow E \rightarrow G$

By integrating these two local search techniques, the GD-GWO algorithm enhances exploration and refines solutions, making it well-suited for the TSP and similar combinatorial optimization problems.

Similar to the original Grey Wolf Optimizer (GWO), the proposed algorithm updates each wolf's position using the difference vectors D_α, D_β, and D_δ. In each time step t, each wolf i generates new candidate positions X_1, X_2, and X_3 based on these vectors:

1. **Generating $\mathbf{X_1}$:** For each generation, wolf i performs a series of D_α 2-opt moves on its current solution. Each time an improved solution is found, it replaces the previous one. This approach ensures that the 2-opt moves incrementally enhance the solution quality Fig.IV.5.

2. **Generating $\mathbf{X_2}$:** Each wolf examines D_β neighboring solutions by applying 2-opt move on and selecting the best one as its new position Fig.IV.6.

3. **Generating $\mathbf{X_3}$:** Each wolf examines D_δ neighbors using the swap move, comparing the results and selecting the best one for its next position Fig.IV.7.

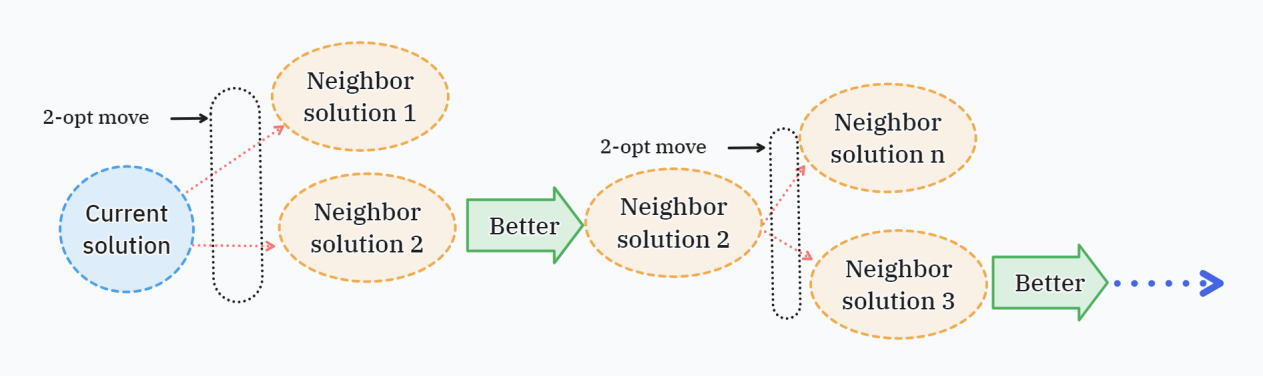


Figure IV.5: successive 2-opt moves update approach

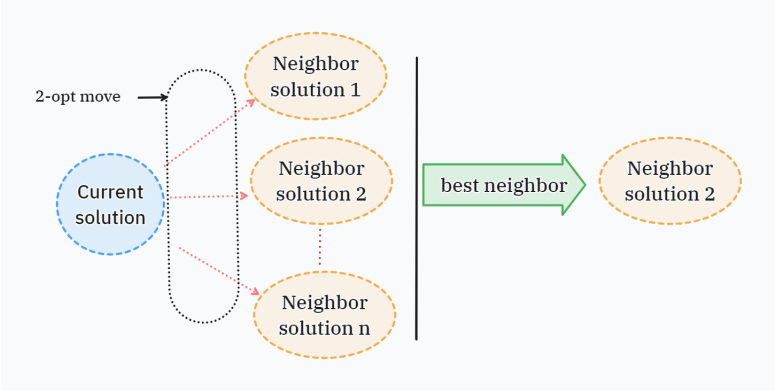


Figure IV.6: 2-opt move update approach

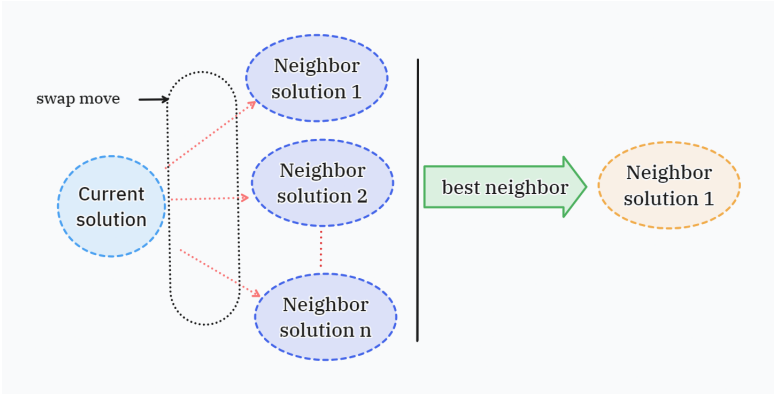


Figure IV.7: swap move update approach

Additionally, to generate a new solution, a modification of the original equation is used, with two methods available for selecting the wolf's new position:

1. In the first method, the best candidate among X_1, X_2, and X_3, denoted as X_{best}, is chosen as the wolf's new position.

2. In the second method, a greedy approach is used to create a new position by comparing the current wolf position X_i and X_{best}. This approach will be discussed further in a later section.

The choice of which method to apply is controlled by the parameter $|A|$ from the original GWO:

$$\vec{X}_{\text{new}} = \begin{cases} \vec{X}_{\text{best}}, & \text{if } |A| \geq 1, \\ \text{greedy}(\vec{X}_i, \vec{X}_{\text{best}}), & \text{if } |A| < 1. \end{cases} \tag{IV.6}$$

This dynamic alternation between exploration (using swap mutation) and exploitation (via 2-opt moves) enhances the search process, balancing global exploration and local refinement to improve convergence quality.

IV.4.3 Proposed Greedy Position Approach

In this section, we describe the Greedy Position Approach used to construct solutions for the Traveling Salesman Problem (TSP) within the Grey Wolf Optimizer (GWO) framework. This approach incrementally builds a solution by selecting cities based on their proximity to the current city, effectively minimizing the travel distance at each step. Also it balances exploration and exploitation by utilizing both the current agent's position and a guide vector generated during the GWO update step. The procedure is described by the following:

Constructs a new solution by selecting the next city to visit based on the positions of the corresponding cities in $X_{current}$ and X_{best}. The city closest to the last visited city is chosen, minimizing the immediate travel distance.

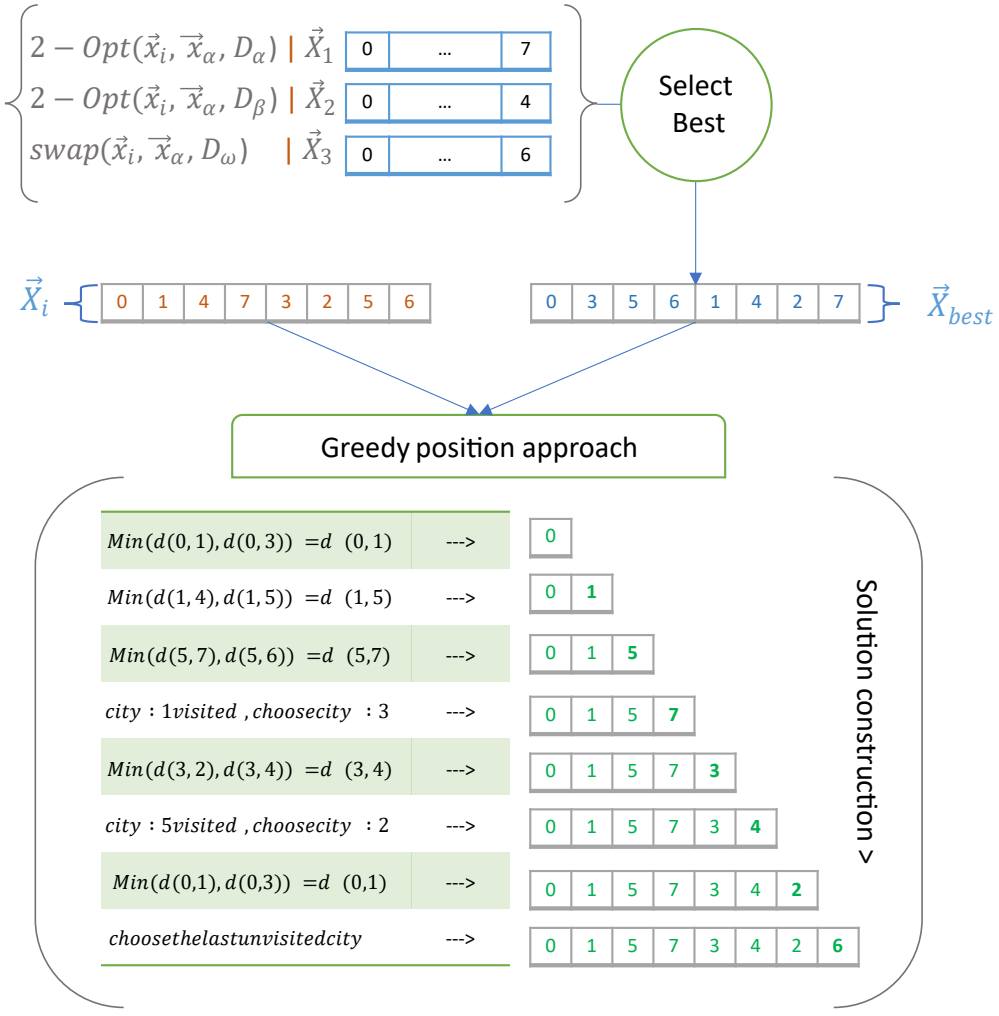


Figure IV.8: Demonstration of greedy position approach.

The Greedy Position Approach starts by initializing an empty solution vector and an unvisited cities set. Each iteration involves selecting the next city based on a greedy criterion that minimizes the immediate travel distance. The algorithm details are as follows:

- **Initialization**: Begin with an empty solution vector and a set to track visited cities. The process starts from a predefined starting city.
- **Iterative Construction**: For each step in the tour:

  (*Note only cities that align with the same position in the sequence of each vector are considered see figure IV.8*)

  - Select two potential next cities: `next_city_current` from the current wolf position vector and `next_city_X` from the $\vec{X}_{best}$, vector.
  - Evaluate both cities for the shortest unvisited option and select the one with the minimal distance from the current city.
  - Update the solution vector with the chosen city and mark it as visited.
- **Termination**: The process repeats until all cities are visited, resulting in a feasible TSP route.

The key to this approach lies in the greedy selection mechanism, which is used to choose the next city to add to the tour based on proximity. Given two candidate cities (`next_city_current` and `next_city_X`), the algorithm computes the travel distance from the current city to each candidate, selecting the one with the shorter distance if both are unvisited. If only one candidate remains unvisited, it is automatically chosen for the next step.

The greedy selection mechanism ensures that each addition to the tour is locally optimized, contributing to a shorter overall path. This greedy heuristic is computationally efficient and aids in rapid convergence of the algorithm.

Figure IV.8 demonstrates the process of constructing a tour using the Greedy Position Approach. In this example, we visualize the selection of cities based on distance, with each step showing the comparison of distances between the candidates and the current city, resulting in an incrementally constructed optimized tour. For this example, we consider the $\vec{X}_{\text{best}}$ and $\vec{X}_i$ to construct a new solution $\vec{X}_{i,new}$ using our strategy. first, we add the closest city to the depot. then we continue to concatenate the last city of the new solution by the closest city either from $\vec{X}_{\text{best}}$ or $\vec{X}_i$ based on the strategy chosen. During the construction process of the new solution, it is necessary to ensure that the cities chosen from $\vec{X}_{\text{best}}$ and $\vec{X}_i$ are not already in the new solution. The pseudo-code and the flowchart of the proposed GD-GWO is depicted in Algorithm 4 and Fig.IV.2.

Algorithm 4 Pseudo Code of GD-GWO

Define the cost function $f(x)$
Initialize population of wolves $X_i (i = 1, 2, \ldots, n)$
Initialize the parameters a, A, and C
while $t \leq MaxIter$ **do**
    Calculate the fitness of each wolf
    Update value of A
    X_α = the best search agent
    X_β = the second best search agent
    X_δ = the third best search agent
    for wolf X_i in the population **do**
        $D_\alpha = \sum_{k \in \text{subseq}} \delta(X_i(k), X_\alpha(k))$
        $D_\beta = \sum_{k \in \text{subseq}} \delta(X_i(k), X_\beta(k))$
        $D_\delta = \sum_{k \in \text{subseq}} \delta(X_i(k), X_\delta(k))$
        $X_1 = \text{2-opt}(X_i, D_\alpha)$
        $X_2 = \text{2-opt}(X_i, D_\beta)$
        $X_3 = \text{swap}(X_i, D_\delta)$
        $X_{best} = best(X_1, X_2, X_3)$
        if $|A| < 0$ **then**
            $X_i = X_{best}$
        else
            $X_i = greedy(X_i, X_{best})$
        end if
        $t = t + 1$
    end for
end while

IV.5 Experimental Evaluation

To assess the effectiveness of the Greedy Discrete Grey Wolf Optimizer (GD-GWO) in solving the Traveling Salesman Problem (TSP), experiments were conducted on 19 benchmark instances from TSPLIB (Reinelt, 1991). These instances varied in size and complexity, from smaller problems like Oliver30 to larger, more challenging instances such as Pr264 and Pr299.

The experiments for GD-GWO are conducted on a 2.30 GHz CPU Desktop with 4 GB RAM, while implemented on software MATLAB R2016a. It is important to mention that results obtained by all the comparison techniques DICA, GA, ESA, and IDGA have been taken from recently published work (Osaba et al, 2016) Also, it is worth mentioning that results depend on random numbers so they can vary slightly. The primary metric of comparison was the mean solution quality (Mean) across multiple runs, while additional metrics, including standard deviation, provided further insight into algorithmic

performance.

The performance of the proposed GD-GWO algorithm has been compared with four other algorithms: ESA, DICA, IDGA, and GA summarized in Table IV.2. The results in Table IV.2 highlight GD-GWO's ability to achieve superior or comparable results across multiple instances, demonstrating its robustness and effectiveness in solving optimization problems.

Comparison with ESA: GD-GWO outperformed ESA in Oliver30, Eil51, Berlin52, St70, Eil76, KroA100, KroB100, and KroC100. It performed equally in KroD100 and KroE100, and was outperformed in Eil101, Pr107, Pr124, Pr136, Pr144, Pr152, Pr264, and Pr299.

Comparison with DICA: GD-GWO performed better than DICA in Oliver30, Eil51, Berlin52, St70, Eil76, KroA100, KroB100, KroC100, and KroD100. It performed equally in KroE100 and Eil101, and was outperformed in Pr107, Pr124, Pr136, Pr144, Pr152, Pr264, and Pr299.

Comparison with IDGA: GD-GWO outperformed IDGA in Oliver30, Eil51, Berlin52, St70, Eil76, KroA100, KroB100, KroC100, KroD100, KroE100, Eil101, Pr107, Pr124, Pr136, and Pr144. It performed equally in Pr152 and was outperformed in Pr264 and Pr299.

Comparison with GA: GD-GWO showed the strongest performance against GA, outperforming it in Oliver30, Eil51, Berlin52, St70, Eil76, KroA100, KroB100, KroC100, KroD100, KroE100, Eil101, Pr107, Pr124, Pr136, Pr144, and Pr152. It performed equally in Pr264 and was outperformed in Pr299.

To ensure a fair and rigorous comparison, Friedman's non-parametric test was conducted on the results using KEEL software (Alcalá-Fdez et al, 2009), to determine any significant differences between the algorithms considered in the experimentation. The test results, presented as mean rankings in Table IV.1, indicate that lower mean ranks correspond to better performance. As seen in the table, the proposed DG-GWO algorithm achieves the lowest mean rank, confirming its superior performance compared to the other approaches.

Algorithm	**Ranking**
GDGWO	2.11
DICA	2.13
ESA	2.47
IDGA	3.61
GA	4.66

Table IV.1: Average Rankings of the Algorithms

Table IV.2: Results of GD-GWO, ESA, DICA, IDGA, and GA on TSP

Instance	Optima	GD-GWO		ESA		DICA		IDGA		GA	
		Mean	Std	Mean	Std	Mean	Std	Mean	Std	Mean	Std
Oliver30	420	420	0.0	420.0	0.0	420.0	0.0	421.5	2.1	422.8	3.4
Eil51	426	429.1	3.6	431.6	2.9	432.3	3.1	434.4	4.5	440.8	7.3
Berlin52	7542	7542	0.0	7542.0	0.0	7542.0	0.0	7542.0	0.0	7542.0	0.0
St70	675	680.4	4.8	682.1	3.9	684.7	3.7	690.2	9.8	709.8	5.7
Eil76	538	553.6	5.2	553.7	4.2	557.6	5.8	557.7	6.8	565.4	9.8
KroA100	21282	21329.2	127.4	21481.7	150.1	21500.3	183.4	21731.8	340.7	21812.4	420.8
KroB100	22140	22599.7	245.3	22602.2	210.2	22599.7	244.9	22712.6	312.8	22687.4	407.7
KroC100	20749	21147.1	273.6	21170.4	188.7	21103.9	161.1	21298.7	290.7	21510.4	390.2
KroD100	21294	21615.4	161.2	21726.5	156.9	21666.8	174.0	21696.9	408.9	22184.6	405.0
KroE100	22068	22882.6	197.6	22499.7	171.4	22453.3	196.9	22721.9	368.0	22741.3	306.0
Eil101	629	663.8	11.9	658.4	4.4	663.8	9.6	660.7	7.5	673.8	12.5
Pr107	44303	44663	382.4	44821.5	179.3	44803.3	302.7	44902.5	660.3	45619.6	1395.4
Pr124	59030	59204.6	405.2	59593.6	367.8	59436.9	299.4	59912.8	532.1	59901.0	562.6
Pr136	96772	99858.3	1102.2	99858.3	655.7	99583.7	848.9	99932.7	1301.2	100472.4	1225.6
Pr144	58537	58893.1	495.3	58807.3	220.9	59070.9	323.0	58893.0	1012.4	60591.4	2342.8
Pr152	73682	75406.1	626.2	74969.5	498.9	74886.7	513.9	75126.7	1005.7	75658.3	910.8
Pr264	49135	52198.3	907.9	52198.5	426.1	51943.6	863.7	52290.0	782.7	52499.8	932.4
Pr299	48191	52179.7	1341.1	50532.3	915.8	49880.3	1413.7	50513.3	1257.9	50817.1	1585.7

114

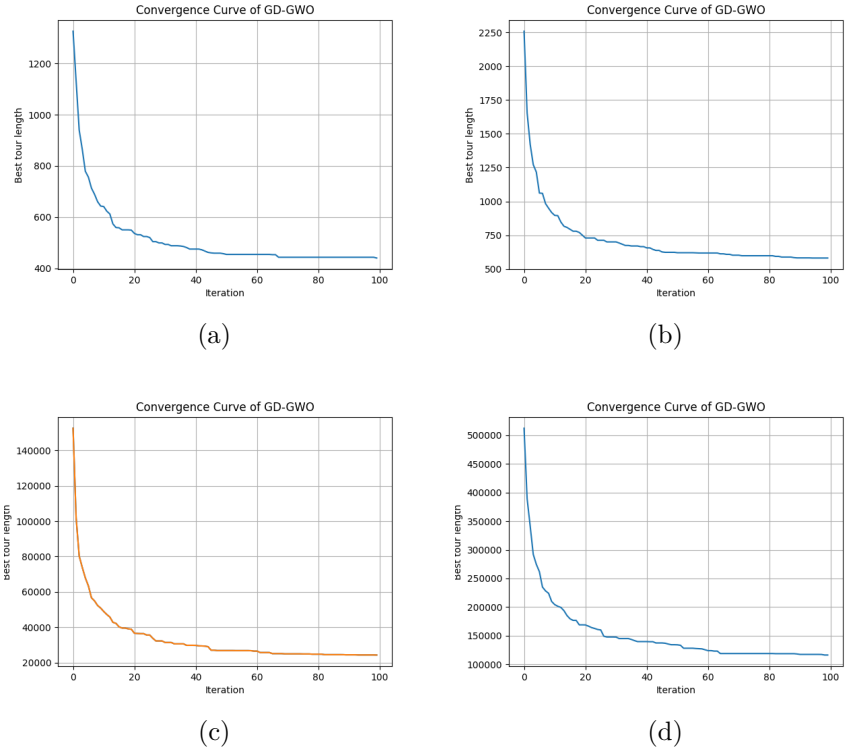
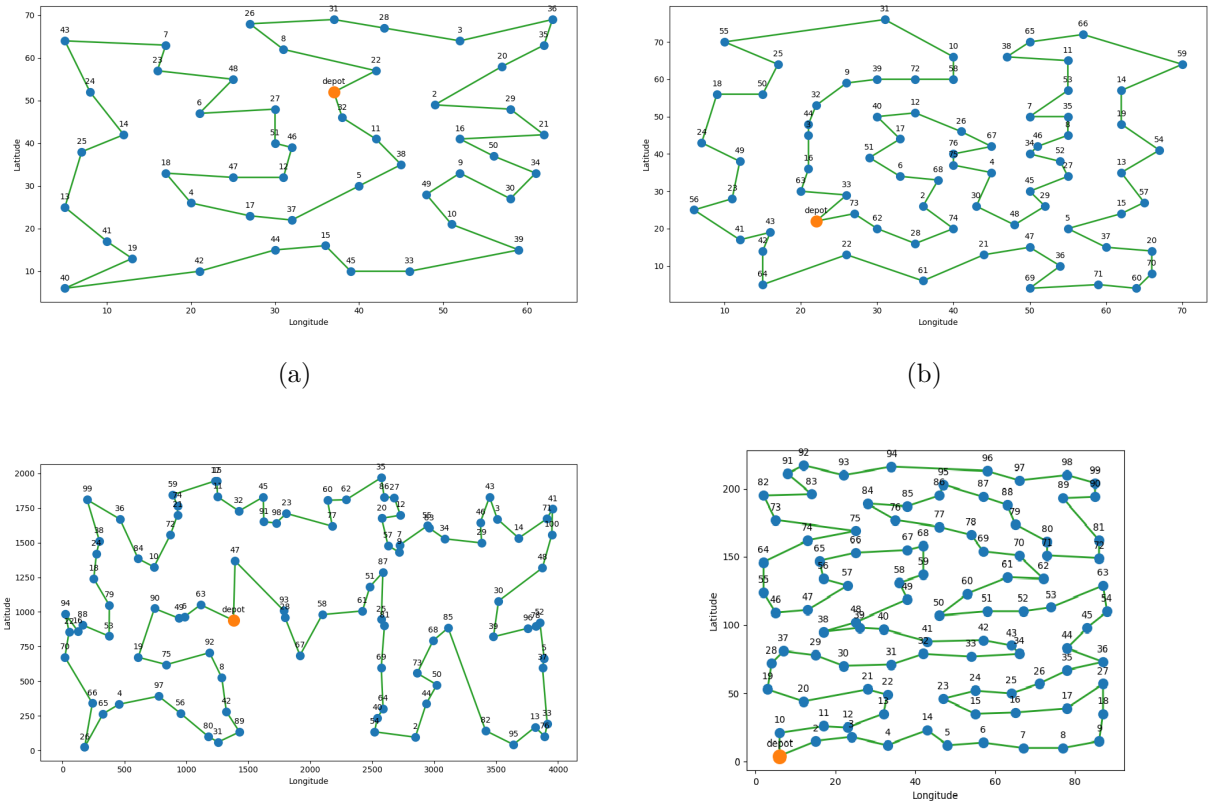


Figure IV.9: Convergence curves of GD-GWO: (a) Eil51, (b) Eil76, (c) kroA100, (d) pr76



(a) (b)

(c) (d)

Figure IV.10: Best tours obtained by GD-GWO: (a) Eil51, (b) Eil76, (c) kroA100, (d) ratt99

Fig.IV.9 shows the convergence tests of a number of TSP problem instances of different scales which were benchmarked to test the effectiveness of the DG-GWO algorithm where only the first 100 iterations are shown. It is important to note that convergence graphs are only presented for the proposed algorithm since the data for the other algorithms, used for comparison, were sourced from the literature, where this information was unavailable. Furthermore, Fig.IV.10 illustrate some of the best tours achieved by the proposed approach in some TSP instances.

IV.6 Conclusion

In this chapter, we proposed a Greedy Discrete Grey Wolf Optimizer (GD-GWO) as a novel approach for solving the Traveling Salesman Problem (TSP), demonstrating the potential of bio-inspired algorithms in constrained optimization. The original Grey Wolf Optimizer, designed for continuous optimization, was adapted to handle discrete problems by incorporating mechanisms like 2-opt and swap mutation, as well as a greedy selection process. This modification allowed us to effectively address the combinatorial constraints inherent in the TSP.

Our primary goal was to showcase the success of bio-inspired algorithms, such as GD-GWO, in solving constrained optimization problems. To validate this, we compared the performance of GD-GWO with four state-of-the-art algorithms across 10 benchmark TSP instances. The results clearly demonstrated that GD-GWO provided competitive solutions, reinforcing the efficacy of bio-inspired methods in navigating the complex search space of constrained problems like the TSP. Despite the increased computational time as problem complexity grew, GD-GWO remained a promising alternative to traditional metaheuristics, showing that bio-inspired strategies can successfully manage constraints while finding high-quality solutions.

Moreover, the flexibility of GD-GWO suggests that it can be extended to other constrained routing problems. Future work may involve enhancing the algorithm by incorporating additional features from the original GWO and exploring its effectiveness in solving real-world constrained optimization problems, such as the Capacitated Vehicle Routing Problem (CVRP) and Rich Vehicle Routing Problem (RVRP). Additionally, more extensive comparisons with other bio-inspired algorithms and exact methods will offer deeper insights into its performance, particularly in terms of computational efficiency and constraint handling, further demonstrating the applicability of bio-inspired approaches to constrained optimization challenges.

General Conclusion

In this thesis, we focused on addressing one of the most significant challenges in optimization which is solving constrained optimization problems using bio-inspired algorithms. These problems are prevalent across various fields and impact critical performance criteria, such as cost, accuracy, and computational efficiency. Given their complexity, exact methods are often inadequate due to the extensive computational time required to reach optimal solutions. Bio-inspired algorithms have emerged as powerful alternatives, gaining popularity due to their ability to efficiently explore large and complex search spaces.

Our research made several contributions to the field of constrained optimization by proposing innovative solutions leveraging bio-inspired techniques. The first major contribution is the development of an adaptive constrained differential evolution. By introducing the Adaptive Coordinate System for constrained optimization, we enhanced the balance between optimizing the objective function and constraint satisfaction.

The proposed technique, abbreviated ACSCDE, introduces two novel dynamic coordinate systems to the original Differential Evolution (DE) framework, enhancing the balance between optimizing the objective function and satisfying constraints. We developed a new ranking-based improvement metric for evaluating individuals, which allows for a more nuanced assessment of solution quality in the context of constraint handling. Additionally, we incorporated innovative mutation strategies to increase the diversity of the search, promoting a broader exploration of the solution space. we further refine the ACSCDE approach by implementing an adaptive mechanism to dynamically adjust the search direction. These improvements aim to enhance the exploration and exploitation balance of DE, enabling it to handle a wide range of constraint types dynamically. The experimental results demonstrated that our method offers superior performance in terms of convergence and accuracy, outperforming several state-of-the-art algorithms.

In the second contribution, we aimed to demonstrate the applicability of bio-inspired algorithms in real-world constrained optimization scenarios by focusing on the Traveling Salesman Problem (TSP). To this end, we developed the Greedy Discrete Grey Wolf Optimizer (GD-GWO), a hybrid approach that enhances the original Grey Wolf Optimizer

(GWO) to effectively tackle the complexities associated with discrete optimization. Comparative studies showed that our algorithm produced competitive results, positioning it as a promising solution for complex routing problems. This contribution highlights reinforces the relevance of bio-inspired techniques in solving practical constrained real-world optimization challenges

While the proposed approaches demonstrated their efficiency and adaptability, there remain several avenues for future work. Enhancements could include:

- Extending our algorithms to multi-objective optimization problems, where multiple conflicting objectives such as cost, accuracy, and reliability must be considered.
- Addressing real-world complexities such as dynamic constraints, stochastic environments, and uncertainty in decision variables.
- Implementing our algorithms in real-world scenarios to assess their practical applicability and robustness in solving large-scale optimization problems.
- Investigating hybridization with other advanced bio-inspired techniques to further improve performance and scalability.

In conclusion, this thesis has shown the potential of bio-inspired algorithms in solving constrained optimization problems and paves the way for future innovations in the field. These contributions not only offer solutions to current challenges but also provide a foundation for addressing more complex optimization problems in a variety of real-world applications.

LIST OF PUBLICATIONS

International scientific journals

Boualem, S.A.E.M., Meftah, B. & Debbat, F. *An adaptive coordinate systems for constrained differential evolution.* Cluster Comput 28, 37 (2025). https://doi.org/10.1007/s10586-024-04698-8

International conferences

Boualem, S. M., Meftah, B., & Debbat, F. (2022). *Solving Travelling Salesman Problem Using a Modified Grey Wolf Optimizer.* In M. Hatti (Ed.), *Artificial Intelligence and Heuristics for Smart Energy Efficiency in Smart Cities* (pp. 708–716). Cham: Springer International Publishing. https://doi.org/10.1007/978-3-030-92038-8_71

Boualem, S. A. M., Boudjelal, M., (2022).*sustainable waste management optimization: A review of objectives, constraints, modelling approches and their challenges toward achieving sustainable development goals.*the 1st international conferance on sustainable economy,energy and environmental-prospective modelling.(ICS3EPM'22)

Bibliography

Ahmadianfar I, Bozorg-Haddad O, Chu X (2020) Gradient-based optimizer: A new metaheuristic optimization algorithm. Information Sciences 540:131–159

Akhand M, Ayon SI, Shahriyar S, et al (2020) Discrete Spider Monkey Optimization for Travelling Salesman Problem. Applied Soft Computing 86:105887. https://doi.org/10.1016/j.asoc.2019.105887

Al-Betar MA, Alyasseri ZAA, Awadallah MA, et al (2021) Coronavirus herd immunity optimizer (chio). Neural Computing and Applications 33:5011–5042

Al-Gaphari GH, Al-Amry R, Al-Nuzaili AS (2021) Discrete crow-inspired algorithms for traveling salesman problem. Engineering Applications of Artificial Intelligence 97:104006

Alcalá-Fdez J, Sanchez L, Garcia S, et al (2009) Keel: a software tool to assess evolutionary algorithms for data mining problems. Soft Computing 13:307–318

Andréasson N, Evgrafov A, Patriksson M (2020) An introduction to continuous optimization: foundations and fundamental algorithms. Courier Dover Publications

Andrei N (2008) An unconstrained optimization test functions collection. Adv Model Optim 10(1):147–161

Askarzadeh A (2014) Bird mating optimizer: an optimization algorithm inspired by bird mating strategies. Communications in Nonlinear Science and Numerical Simulation 19(4):1213–1228

Askarzadeh A (2016) A novel metaheuristic method for solving constrained engineering optimization problems: crow search algorithm. Computers & structures 169:1–12

Asorey-Cacheda R, Garcia-Sanchez AJ, García-Sánchez F, et al (2017) A survey on nonlinear optimization problems in wireless sensor networks. Journal of Network and Computer Applications 82:1–20

Atashpaz E, Lucas C (2007) Imperialist competitive algorithm: an algorithm for optimization inspired by imperialistic competition. In: 2007 IEEE congress on evolutionary computation, Ieee, pp 4661–4667

Auer P, Cesa-Bianchi N, Fischer P (2002) Finite-time Analysis of the Multiarmed Bandit Problem. Machine Learning 47(2):235–256. https://doi.org/10.1023/A:1013689704352

Barbosa HJ, Lemonge AC, Bernardino HS (2015) A critical review of adaptive penalty techniques in evolutionary computation. Evolutionary constrained optimization pp 1–27

Bazgan C, Ruzika S, Thielen C, et al (2022) The power of the weighted sum scalarization for approximating multiobjective optimization problems. Theory of Computing Systems pp 1–21

Bertsekas DP, Tsitsiklis JN (1999) Nonlinear Programming. Athena Scientific

Birgin EG, Martínez JM (2014) Practical augmented Lagrangian methods for constrained optimization. SIAM

Boryczka U, Szwarc K (2019) The harmony search algorithm with additional improvement of harmony memory for asymmetric traveling salesman problem. Expert Systems with Applications 122:43–53

Boyd S, Vandenberghe L (2004) Convex optimization. Cambridge university press

Chou JS, Truong DN (2021) A novel metaheuristic optimizer inspired by behavior of jellyfish in ocean. Applied Mathematics and Computation 389:125535

Chu SC, Tsai PW, Pan JS (2006) Cat swarm optimization. In: PRICAI 2006: Trends in Artificial Intelligence: 9th Pacific Rim International Conference on Artificial Intelligence Guilin, China, August 7-11, 2006 Proceedings 9, Springer, pp 854–858

Coello CAC (2002) Theoretical and numerical constraint-handling techniques used with evolutionary algorithms: a survey of the state of the art. Computer methods in applied mechanics and engineering 191(11-12):1245–1287

Črepinšek M, Liu SH, Mernik M (2013) Exploration and exploitation in evolutionary algorithms: A survey. ACM computing surveys (CSUR) 45(3):1–33

Croes GA (1958) A method for solving traveling-salesman problems. Operations research 6(6):791–812

Deb K (2000) An efficient constraint handling method for genetic algorithms. Computer methods in applied mechanics and engineering 186(2-4):311–338

Deb K, Sindhya K, Hakanen J (2016) Multi-objective optimization. In: Decision sciences. CRC Press, p 161–200

Dehghani M, Trojovskỳ P, Malik OP (2023) Green anaconda optimization: a new bio-inspired metaheuristic algorithm for solving optimization problems. Biomimetics 8(1):121

Diwekar UM (2020) Introduction to applied optimization, vol 22. Springer Nature

Dorigo M, Gambardella LM (1997) Ant colonies for the travelling salesman problem. biosystems 43(2):73–81

Dorigo M, Birattari M, Stutzle T (2006) Ant colony optimization. IEEE computational intelligence magazine 1(4):28–39

Duan H, Qiao P (2014) Pigeon-inspired optimization: a new swarm intelligence optimizer for air robot path planning. International journal of intelligent computing and cybernetics 7(1):24–37

Dulac-Arnold G, Levine N, Mankowitz DJ, et al (2021) Challenges of real-world reinforcement learning: definitions, benchmarks and analysis. Machine Learning 110(9):2419–2468

Emary E, Zawbaa HM, Hassanien AE (2016) Binary grey wolf optimization approaches for feature selection. Neurocomputing 172:371–381. Publisher: Elsevier

Eskandar H, Sadollah A, Bahreininejad A, et al (2012) Water cycle algorithm–a novel metaheuristic optimization method for solving constrained engineering optimization problems. Computers & Structures 110:151–166

Fan X, Sayers W, Zhang S, et al (2020) Review and classification of bio-inspired algorithms and their applications. Journal of Bionic Engineering 17:611–631

Fan Z, Li H, Wei C, et al (2016) An improved epsilon constraint handling method embedded in moea/d for constrained multi-objective optimization problems. In: 2016 IEEE symposium series on computational intelligence (SSCI), IEEE, pp 1–8

Faris H, Aljarah I, Al-Betar MA, et al (2018) Grey wolf optimizer: a review of recent variants and applications. Neural Computing and Applications 30(2):413–435. https://doi.org/10.1007/s00521-017-3272-5

Fioretto F, Pontelli E, Yeoh W (2018) Distributed constraint optimization problems and applications: A survey. Journal of Artificial Intelligence Research 61:623–698

Fletcher R (2013) Practical methods of optimization. John Wiley & Sons

Gandomi AH, Alavi AH (2012) Krill herd: a new bio-inspired optimization algorithm. Communications in nonlinear science and numerical simulation 17(12):4831–4845

Gao W, Dang Q, Gong M (2022) An adaptive framework to select the coordinate systems for evolutionary algorithms. Applied Soft Computing 129:109585. https://doi.org/10.1016/j.asoc.2022.109585

Geng X, Chen Z, Yang W, et al (2011) Solving the traveling salesman problem based on an adaptive simulated annealing algorithm with greedy search. Applied Soft Computing 11(4):3680–3689

Giagkiozis I, Fleming PJ (2015) Methods for multi-objective optimization: An analysis. Information Sciences 293:338–350

Glover F, Laguna M (1998) Tabu search. Springer

Guo SM, Yang CC (2015) Enhancing Differential Evolution Utilizing Eigenvector-Based Crossover Operator. IEEE Transactions on Evolutionary Computation 19(1):31–49. https://doi.org/10.1109/TEVC.2013.2297160

Haeser G, Liu H, Ye Y (2019) Optimality condition and complexity analysis for linearly-constrained optimization without differentiability on the boundary. Mathematical Programming 178(1):263–299

Hansen N, Ostermeier A (2001) Completely Derandomized Self-Adaptation in Evolution Strategies. Evolutionary Computation 9(2):159–195. https://doi.org/10.1162/106365601750190398

Hansen N, Ros R (2010) Benchmarking a weighted negative covariance matrix update on the bbob-2010 noiseless testbed. In: Proceedings of the 12th annual conference companion on Genetic and evolutionary computation, pp 1673–1680

Hansen N, Arnold DV, Auger A (2015) Evolution strategies. Springer handbook of computational intelligence pp 871–898

Hansen P, Mladenović N (2001) Variable neighborhood search: Principles and applications. European journal of operational research 130(3):449–467

Heidari AA, Mirjalili S, Faris H, et al (2019) Harris hawks optimization: Algorithm and applications. Future generation computer systems 97:849–872

Houssein EH, Saad MR, Hashim FA, et al (2020) Lévy flight distribution: A new metaheuristic algorithm for solving engineering optimization problems. Engineering Applications of Artificial Intelligence 94:103731

Hu Z, Gong W (2022) Constrained evolutionary optimization based on reinforcement learning using the objective function and constraints. Knowledge-Based Systems 237:107731

Hussain A, Muhammad YS, Nauman Sajid M, et al (2017) Genetic algorithm for traveling salesman problem with modified cycle crossover operator. Computational intelligence and neuroscience 2017

Ibrahim RA, Elaziz MA, Lu S (2018) Chaotic opposition-based grey-wolf optimization algorithm based on differential evolution and disruption operator for global optimization. Expert Systems with Applications 108:1–27. https://doi.org/10.1016/j.eswa.2018.04.028

Jati GK, et al (2011) Evolutionary discrete firefly algorithm for travelling salesman problem. In: International conference on adaptive and intelligent systems, Springer, pp 393–403

Jayabarathi T, Raghunathan T, Adarsh BR, et al (2016) Economic dispatch using hybrid grey wolf optimizer. Energy 111:630–641. https://doi.org/10.1016/j.energy.2016.05.105

Kar AK (2016) Bio inspired computing–a review of algorithms and scope of applications. Expert Systems with Applications 59:20–32

Karaboga D, Gorkemli B (2011) A combinatorial artificial bee colony algorithm for traveling salesman problem. In: 2011 International Symposium on Innovations in Intelligent Systems and Applications, IEEE, pp 50–53

Kaveh A, Mohsen K (2019) Artificial coronary circulation system: A new bio-inspired metaheuristic algorithm. Scientia Iranica 26(5):2731–2747

Kennedy J (2006) Swarm intelligence. In: Handbook of nature-inspired and innovative computing: integrating classical models with emerging technologies. Springer, p 187–219

Kennedy J, Eberhart R (1995) Particle swarm optimization. In: Proceedings of ICNN'95-international conference on neural networks, IEEE, pp 1942–1948

Kiran MS (2015) Tsa: Tree-seed algorithm for continuous optimization. Expert Systems with Applications 42(19):6686–6698

Kirkpatrick S, Gelatt Jr CD, Vecchi MP (1983) Optimization by simulated annealing. science 220(4598):671–680

Knox J (1994) Tabu search performance on the symmetric traveling salesman problem. Computers & Operations Research 21(8):867–876

Komaki GM, Kayvanfar V (2015) Grey Wolf Optimizer algorithm for the two-stage assembly flow shop scheduling problem with release time. Journal of Computational Science 8:109–120. https://doi.org/10.1016/j.jocs.2015.03.011

Kramer O, Kramer O (2017) Genetic algorithms. Springer

Kulkarni O, Kulkarni N, Kulkarni AJ, et al (2018) Constrained cohort intelligence using static and dynamic penalty function approach for mechanical components design. International journal of parallel, emergent and distributed systems 33(6):570–588

Lawler EL, Lenstra JK, Rinnooy Kan AH, et al (1986) Erratum: The traveling salesman problem: A guided tour of combinatorial optimization. Journal of the Operational Research Society 37(6):655–655

Lee KS, Geem ZW (2005) A new meta-heuristic algorithm for continuous engineering optimization: harmony search theory and practice. Computer methods in applied mechanics and engineering 194(36-38):3902–3933

Li J, Li G, Wang Z, et al (2023a) Differential evolution with an adaptive penalty coefficient mechanism and a search history exploitation mechanism. Expert Systems with Applications 230:120530. https://doi.org/10.1016/j.eswa.2023.120530

Li K, Fialho A, Kwong S, et al (2014a) Adaptive Operator Selection With Bandits for a Multiobjective Evolutionary Algorithm Based on Decomposition. IEEE Transactions on Evolutionary Computation 18(1):114–130. https://doi.org/10.1109/TEVC.2013.2239648

Li L, Sun L, Guo J, et al (2017) Modified discrete grey wolf optimizer algorithm for multilevel image thresholding. Computational intelligence and neuroscience 2017

Li X, Zhang J, Yin M (2014b) Animal migration optimization: an optimization algorithm inspired by animal migration behavior. Neural computing and applications 24:1867–1877

Li Y, Han T, Tang S, et al (2023b) An improved differential evolution by hybridizing with estimation-of-distribution algorithm. Information Sciences 619:439–456. https://doi.org/10.1016/j.ins.2022.11.029

Liang J, Ban X, Yu K, et al (2021) Differential evolution with rankings-based fitness function for constrained optimization problems. Applied Soft Computing 113:108016. https://doi.org/10.1016/j.asoc.2021.108016

Liang Y, Ren Z, Yao X, et al (2020) Enhancing Gaussian Estimation of Distribution Algorithm by Exploiting Evolution Direction With Archive. IEEE Transactions on Cybernetics 50(1):140–152. https://doi.org/10.1109/TCYB.2018.2869567

Liberti L, Kucherenko S (2005) Comparison of deterministic and stochastic approaches to global optimization. International Transactions in Operational Research 12(3):263–285

Liu R, Liu Y, Zeng S, et al (2021) Towards gradient-based bilevel optimization with non-convex followers and beyond. Advances in Neural Information Processing Systems 34:8662–8675

Liu ZZ, Wang Y, Yang S, et al (2019) An Adaptive Framework to Tune the Coordinate Systems in Nature-Inspired Optimization Algorithms. IEEE Transactions on Cybernetics 49(4):1403–1416. https://doi.org/10.1109/TCYB.2018.2802912

Long W, Jiao J, Liang X, et al (2018a) An exploration-enhanced grey wolf optimizer to solve high-dimensional numerical optimization. Engineering Applications of Artificial Intelligence 68:63–80. https://doi.org/10.1016/j.engappai.2017.10.024

Long W, Jiao J, Liang X, et al (2018b) Inspired grey wolf optimizer for solving large-scale function optimization problems. Applied Mathematical Modelling 60:112–126. https://doi.org/10.1016/j.apm.2018.03.005

Lourenço HR, Martin OC, Stützle T (2019) Iterated local search: Framework and applications. Handbook of metaheuristics pp 129–168

Lu C, Xiao S, Li X, et al (2016) An effective multi-objective discrete grey wolf optimizer for a real-world scheduling problem in welding production. Advances in Engineering Software 99:161–176. https://doi.org/10.1016/j.advengsoft.2016.06.004

Luenberger DG, Ye Y, Luenberger DG, et al (2016) Penalty and barrier methods. Linear and Nonlinear Programming pp 397–428

Makhadmeh SN, Al-Betar MA, Doush IA, et al (2023) Recent advances in grey wolf optimizer, its versions and applications. IEEE Access

Malik MRS, Mohideen ER, Ali L (2015) Weighted distance grey wolf optimizer for global optimization problems. In: 2015 IEEE International Conference on Computational Intelligence and Computing Research (ICCIC). IEEE, pp 1–6

Mallipeddi R, Suganthan PN (2010) Problem definitions and evaluation criteria for the cec 2010 competition on constrained real-parameter optimization. Nanyang Technological University, Singapore 24:910

Marti K, et al (2008) Stochastic optimization methods, vol 2. Springer

Melin P, Astudillo L, Castillo O, et al (2013) Optimal design of type-2 and type-1 fuzzy tracking controllers for autonomous mobile robots under perturbed torques using a new chemical optimization paradigm. Expert Systems with Applications 40(8):3185–3195

Meng X, Liu Y, Gao X, et al (2014) A new bio-inspired algorithm: chicken swarm optimization. In: Advances in Swarm Intelligence: 5th International Conference, ICSI 2014, Hefei, China, October 17-20, 2014, Proceedings, Part I 5, Springer, pp 86–94

Mezura-Montes E, Coello CAC (2011) Constraint-handling in nature-inspired numerical optimization: past, present and future. Swarm and Evolutionary Computation 1(4):173–194

Mezura-Montes E, Coello Coello CA, Tun-Morales EI (2004) Simple feasibility rules and differential evolution for constrained optimization. In: Mexican International Conference on Artificial Intelligence, Springer, pp 707–716

Mirjalili S (2015) The ant lion optimizer. Advances in engineering software 83:80–98

Mirjalili S, Mirjalili S (2019) Introduction to evolutionary single-objective optimisation. Evolutionary Algorithms and Neural Networks: Theory and Applications pp 3–14

Mirjalili S, Mirjalili SM, Lewis A (2014) Grey wolf optimizer. Advances in engineering software 69:46–61

Mirjalili S, Mirjalili SM, Hatamlou A (2016a) Multi-verse optimizer: a nature-inspired algorithm for global optimization. Neural Computing and Applications 27:495–513

Mirjalili S, Saremi S, Mirjalili SM, et al (2016b) Multi-objective grey wolf optimizer: A novel algorithm for multi-criterion optimization. Expert Systems with Applications 47:106–119. https://doi.org/10.1016/j.eswa.2015.10.039

Mirjalili S, Gandomi AH, Mirjalili SZ, et al (2017) Salp swarm algorithm: A bio-inspired optimizer for engineering design problems. Advances in engineering software 114:163–191

Mohamed AW, Sabry HZ (2012) Constrained optimization based on modified differential evolution algorithm. Information Sciences 194:171–208

Moon C, Kim J, Choi G, et al (2002) An efficient genetic algorithm for the traveling salesman problem with precedence constraints. European Journal of Operational Research 140(3):606–617

Morrison DR, Jacobson SH, Sauppe JJ, et al (2016) Branch-and-bound algorithms: A survey of recent advances in searching, branching, and pruning. Discrete Optimization 19:79–102

Naderi B, Khalili M, Khamseh AA (2014) Mathematical models and a hunting search algorithm for the no-wait flowshop scheduling with parallel machines. International Journal of Production Research 52(9):2667–2681

Nadimi-Shahraki MH, Taghian S, Mirjalili S (2021) An improved grey wolf optimizer for solving engineering problems. Expert Systems with Applications 166:113917. Publisher: Elsevier

Nemhauser GL, Wolsey LA (1988) Integer and Combinatorial Optimization. Wiley-Interscience

Osaba E, Diaz F, Onieva E (2014) Golden ball: a novel meta-heuristic to solve combinatorial optimization problems based on soccer concepts. Applied intelligence 41:145–166

Osaba E, Yang XS, Diaz F, et al (2016) An improved discrete bat algorithm for symmetric and asymmetric Traveling Salesman Problems. Engineering Applications of Artificial Intelligence 48:59–71. https://doi.org/10.1016/j.engappai.2015.10.006

Osaba E, Yang XS, Del Ser J (2020) Traveling salesman problem: a perspective review of recent research and new results with bio-inspired metaheuristics. In: Nature-Inspired Computation and Swarm Intelligence. Elsevier, p 135–164, https://doi.org/10.1016/B978-0-12-819714-1.00020-8

Ouaarab A, Ahiod B, Yang XS (2014) Discrete cuckoo search algorithm for the travelling salesman problem. Neural Computing and Applications 24(7-8):1659–1669. https://doi.org/10.1007/s00521-013-1402-2

Palakonda V, Mallipeddi R (2020) An evolutionary algorithm for multi and many-objective optimization with adaptive mating and environmental selection. IEEE Access 8:82781–82796

Panwar K, Deep K (2021) Discrete Grey Wolf Optimizer for symmetric travelling salesman problem. Applied Soft Computing 105:107298. https://doi.org/10.1016/j.asoc.2021.107298

Papadimitriou CH, Yannakakis M (1991) Optimization, approximation, and complexity classes. Journal of Computer and System Sciences 43(3):425–440

Peng H, Xu Z, Qian J, et al (2023) Evolutionary constrained optimization with hybrid constraint-handling technique. Expert Systems with Applications 211:118660. https://doi.org/10.1016/j.eswa.2022.118660

Petsagkourakis P, Sandoval IO, Bradford E, et al (2020) Constrained reinforcement learning for dynamic optimization under uncertainty. IFAC-PapersOnLine 53(2):11264–11270

Pham DT, Ghanbarzadeh A, Koç E, et al (2006) The bees algorithm—a novel tool for complex optimisation problems. In: Intelligent production machines and systems. Elsevier, p 454–459

Polakova R (2017) L-shade with competing strategies applied to constrained optimization. In: 2017 IEEE congress on evolutionary computation (CEC), IEEE, pp 1683–1689

Qais MH, Hasanien HM, Turky RA, et al (2022) Circle search algorithm: A geometry-based metaheuristic optimization algorithm. Mathematics 10(10):1626

Qi X, Zhu Y, Zhang H (2017) A new meta-heuristic butterfly-inspired algorithm. Journal of computational science 23:226–239

Qiao K, Liang J, Yu K, et al (2022) Self-adaptive resources allocation-based differential evolution for constrained evolutionary optimization. Knowledge-Based Systems 235:107653. https://doi.org/10.1016/j.knosys.2021.107653

Qu C, Gai W, Zhang J, et al (2020) A novel hybrid grey wolf optimizer algorithm for unmanned aerial vehicle (uav) path planning. Knowledge-Based Systems 194:105530

Rajeev S, Krishnamoorthy C (1992) Discrete optimization of structures using genetic algorithms. Journal of structural engineering 118(5):1233–1250

Rao R (2016) Jaya: A simple and new optimization algorithm for solving constrained and unconstrained optimization problems. International Journal of Industrial Engineering Computations 7(1):19–34

Rao SS (2019) Engineering optimization: theory and practice. John Wiley & Sons

Rechenberg I (1973) Evolutionsstrategie. Optimierung technischer Systeme nach Prinzipien derbiologischen Evolution

Reddy K S, Panwar L, Panigrahi B, et al (2019) Binary whale optimization algorithm: a new metaheuristic approach for profit-based unit commitment problems in competitive electricity markets. Engineering Optimization 51(3):369–389

Reinelt G (1991) Tsplib—a traveling salesman problem library. ORSA journal on computing 3(4):376–384

Runarsson TP, Yao X (2000) Stochastic ranking for constrained evolutionary optimization. IEEE Transactions on evolutionary computation 4(3):284–294

Sallam KM, Elsayed SM, Sarker RA, et al (2020) Landscape-assisted multi-operator differential evolution for solving constrained optimization problems. Expert Systems with Applications 162:113033

Schulte J, Nissen V (2020) Sensitivity analysis in constrained evolutionary optimization. In: Proceedings of the 2020 genetic and evolutionary computation conference, pp 894–902

Schwefel HP (1965) Kybernetische evolution als strategie der experimentellen forschung in der stromungstechnik. Diploma thesis, Technical Univ of Berlin

Selman B, Gomes CP (2006) Hill-climbing search. Encyclopedia of cognitive science 81(333-335):10

Shi Y (2011) Brain storm optimization algorithm. In: Advances in Swarm Intelligence: Second International Conference, ICSI 2011, Chongqing, China, June 12-15, 2011, Proceedings, Part I 2, Springer, pp 303–309

Singh MK (2012) A new optimization method based on adaptive social behavior: Asbo. In: Proceedings of International Conference on Advances in Computing, Springer, pp 823–831

Sioshansi R, Conejo AJ, et al (2017) Optimization in engineering. Cham: Springer International Publishing 120

Smith AE, Coit DW, Baeck T, et al (1997) Penalty functions. Handbook of evolutionary computation 97(1):C5

Sowmya R, Premkumar M, Jangir P (2024) Newton-raphson-based optimizer: A new population-based metaheuristic algorithm for continuous optimization problems. Engineering Applications of Artificial Intelligence 128:107532

Storn R, Price K (1997) Differential evolution–a simple and efficient heuristic for global optimization over continuous spaces. Journal of global optimization 11:341–359

Subramanian C, Sekar A, Subramanian K (2013) A new engineering optimization method: African wild dog algorithm. Int J Soft Comput 8(3):163–170

Sulaiman MH, Mustaffa Z, Mohamed MR, et al (2015) Using the gray wolf optimizer for solving optimal reactive power dispatch problem. Applied Soft Computing 32:286–292

Takahama T, Sakai S (2010) Constrained optimization by the ε constrained differential evolution with an archive and gradient-based mutation. In: IEEE congress on evolutionary computation, IEEE, pp 1–9

Talbi EG (2009) Metaheuristics: From design to implementation. Wiley Series on Parallel and Distributed Computing

Tanino T, Tanaka T, Inuiguchi M (2013) Multi-objective programming and goal programming: theory and applications, vol 21. Springer Science & Business Media

Tawhid MA, Ibrahim AM (2020) A hybridization of grey wolf optimizer and differential evolution for solving nonlinear systems. Evolving Systems 11(1):65–87. https://doi.org/10.1007/s12530-019-09291-8

Tripathi AK, Sharma K, Bala M (2018) A Novel Clustering Method Using Enhanced Grey Wolf Optimizer and MapReduce. Big Data Research 14:93–100. https://doi.org/10.1016/j.bdr.2018.05.002

Uryasev S (2013) Probabilistic constrained optimization: methodology and applications, vol 49. Springer Science & Business Media

Van Laarhoven PJ, Aarts EH, van Laarhoven PJ, et al (1987) Simulated annealing. Springer

Vince A (2002) A framework for the greedy algorithm. Discrete Applied Mathematics 121(1-3):247–260

Wang BC, Li HX, Feng Y (2018) An improved teaching-learning-based optimization for constrained evolutionary optimization. Information Sciences 456:131–144. https://doi.org/10.1016/j.ins.2018.04.083

Wang BC, Li HX, Li JP, et al (2019) Composite Differential Evolution for Constrained Evolutionary Optimization. IEEE Transactions on Systems, Man, and Cybernetics: Systems 49(7):1482–1495. https://doi.org/10.1109/TSMC.2018.2807785

Wang BC, Li HX, Zhang Q, et al (2021) Decomposition-Based Multiobjective Optimization for Constrained Evolutionary Optimization. IEEE Transactions on Systems, Man, and Cybernetics: Systems 51(1):574–587. https://doi.org/10.1109/TSMC.2018.2876335, conference Name: IEEE Transactions on Systems, Man, and Cybernetics: Systems

Wang KP, Huang L, Zhou CG, et al (2003) Particle swarm optimization for traveling salesman problem. In: Proceedings of the 2003 international conference on machine learning and cybernetics (IEEE cat. no. 03ex693), IEEE, pp 1583–1585

Wang Y, Li HX, Huang T, et al (2014) Differential evolution based on covariance matrix learning and bimodal distribution parameter setting. Applied Soft Computing 18:232–247. https://doi.org/10.1016/j.asoc.2014.01.038

Wang Y, Liu ZZ, Li J, et al (2016a) Utilizing cumulative population distribution information in differential evolution. Applied Soft Computing 48:329–346. https://doi.org/10.1016/j.asoc.2016.07.012

Wang Y, Wang BC, Li HX, et al (2016b) Incorporating Objective Function Information Into the Feasibility Rule for Constrained Evolutionary Optimization. IEEE Transactions on Cybernetics 46(12):2938–2952. https://doi.org/10.1109/TCYB.2015.2493239

Wang Y, Li JP, Xue X, et al (2020) Utilizing the Correlation Between Constraints and Objective Function for Constrained Evolutionary Optimization. IEEE Transactions on Evolutionary Computation 24(1):29–43. https://doi.org/10.1109/TEVC.2019.2904900

Wright MH (1992) Interior methods for constrained optimization. Acta numerica 1:341–407

Wright SJ, Nocedal J (1999) Numerical optimization methods for constrained optimization. SIAM Journal on Optimization 8(3):1000–1023

Wu G, Mallipeddi R, Suganthan PN (2017) Problem definitions and evaluation criteria for the cec 2017 competition on constrained real-parameter optimization. National University of Defense Technology, Changsha, Hunan, PR China and Kyungpook National University, Daegu, South Korea and Nanyang Technological University, Singapore, Technical Report

Xue F, Sanderson AC, Graves RJ (2003) Pareto-based multi-objective differential evolution. In: The 2003 Congress on Evolutionary Computation, 2003. CEC'03., IEEE, pp 862–869

Yadav A, et al (2019) Aefa: Artificial electric field algorithm for global optimization. Swarm and Evolutionary Computation 48:93–108

Yang XS (2009) Firefly algorithms for multimodal optimization. In: International symposium on stochastic algorithms, Springer, pp 169–178

Yang XS (2012) Flower pollination algorithm for global optimization. In: International conference on unconventional computing and natural computation, Springer, pp 240–249

Yang XS (2020) Nature-inspired optimization algorithms: Challenges and open problems. Journal of Computational Science 46:101104

Yoo H, Zavala VM, Lee JH (2021) A dynamic penalty function approach for constraint-handling in reinforcement learning. IFAC-PapersOnLine 54(3):487–491

Yu X, Gen M (2010) Introduction to evolutionary algorithms. Springer Science & Business Media

Yuan Y, Shen Q, Wang S, et al (2023) Coronavirus mask protection algorithm: A new bio-inspired optimization algorithm and its applications. Journal of Bionic Engineering 20(4):1747–1765

Zhang H, Li H, Tam C (2006) Particle swarm optimization for resource-constrained project scheduling. International journal of project management 24(1):83–92

Zhang Q, Wang R, Yang J, et al (2019) Biology migration algorithm: a new nature-inspired heuristic methodology for global optimization. Soft Computing 23:7333–7358

Zhang S, Zhou Y, Li Z, et al (2016) Grey wolf optimizer for unmanned combat aerial vehicle path planning. Advances in Engineering Software 99:121–136. https://doi.org/10.1016/j.advengsoft.2016.05.015

Zhang W, Xu F (2023) A comprehensive review of feasible region exploration in constrained optimization. IEEE Transactions on Neural Networks and Learning Systems 34(6):2502–2515. https://doi.org/10.1109/TNNLS.2022.3145783

Zhang Z, Han Y (2022) Discrete sparrow search algorithm for symmetric traveling salesman problem. Applied Soft Computing 118:108469

Zhao W, Wang L, Zhang Z, et al (2024) Electric eel foraging optimization: A new bio-inspired optimizer for engineering applications. Expert Systems with Applications 238:122200

Zhong Y, Wang L, Lin M, et al (2019) Discrete pigeon-inspired optimization algorithm with Metropolis acceptance criterion for large-scale traveling salesman problem. Swarm and Evolutionary Computation 48:134–144. https://doi.org/10.1016/j.swevo.2019.04.002

Zitouni F, Harous S, Belkeram A, et al (2022) The archerfish hunting optimizer: A novel metaheuristic algorithm for global optimization. Arabian Journal for Science and Engineering 47(2):2513–2553

Zou F, Chen D, Liu H, et al (2022) A survey of fitness landscape analysis for optimization. Neurocomputing 503:129–139