PEOPLE'S DEMOCRATIC REPUBLIC OF ALGERIA

MINISTRY OF HIGHER EDUCATION AND SCIENTIFIC RESEARCH

University of BLIDA-1

Faculty of Technology

Department of Renewable Energies

Thesis submitted for obtaining the Academic Master's degree

OPTION: Photovoltaic Conversion

Entitled:

Numerical Analysis of The Differential Evolution Optimization Algorithm and Its Control Parameters

By: **Idris BABAUOSMAIL**

Hadj Daoud ABDERRAHMAN

Defended in front of the jury composed of:

Dr. B. AMROUCHE	M.C.A	BLIDA-1	Chairman
Dr. F. KHOUJA	M.C.B	BLIDA-1	Examiner
Dr. O.AIT SAHED	M.C.B	BLIDA-1	Supervisor
Dr. R. BOUKENOUI	M.C.B	BLIDA-1	Co-Supervisor

Academic year: 2023 /2024

Acknowledgments

First and foremost, we thank God for granting us life, health, and the will to complete this modest work.

We would like to thank our department head, Mr. M. BOUZAKI, and our option head, Mr. T. DOUMAZ, for their assistance.

We express our gratitude to our supervisor, Mr. O. AIT SAHED, PhD in Electrical Engineering at the University of Blida1, for accepting to lead our work and for his numerous pieces of advice and support throughout this thesis. We also extend our heartfelt thanks to Dr. BOUKENOUI Rachid, our co-supervisor, for his invaluable guidance and support.

With great pride and honor, we extend our thanks to the entire administrative family of the Department of Renewable Energies, and to all the teachers who have imparted knowledge to us.

Our sincerest thanks go to the jury president and jury members, Mrs. B. AMROUCHE and Mr. F.KHOUJA , for the honor of accepting to evaluate this work.

Finally, we would like to express our recognition and gratitude to all the people who have contributed, directly or indirectly, to the realization of this modest work, and to everyone who takes the time to read this document.

Dedication:

I dedicate this modest work to the two people dearest to me in the world: my father, Yahia, and my mother, Fatiha. May God keep them safe.

To my brothers and sisters, whom I love dearly, and especially to my big sister, Iman, and her husband, Rostom, for their tremendous help and support.

To my entire family and friend.

To all my friends in the Department of Renewable Energies.

And to everyone I know, near and far.

Hadj Daoud ABDERRAHMANE

Dedication:

First and foremost, I dedicate this work as an expression of my eternal gratitude to the people I owe the most in this world, my dear parents.

To my sisters and my wife, who have supported and encouraged me.

To all the members of the Al-Haqqa Charitable Foundation who contributed from near or far to the success of my work, especially Sheikh Khazmati Mohammed.

To my precious family and friend.

To all my friends in the Department of Renewable Energies.

To all my loved ones.

Idris BABAUOSMAIL

CONTENT

ملخص:

تهدف هذه الأطروحة إلى دراسة تأثير معاملات التحكم وإصدارات خوارزمية التطور التفاضلي (DE) على أدائها. من خلال تحليل معاملات مثل حجم التعداد ومعدلات الطفرة، إلى جانب استكشاف إصدارات مختلفة من خوارزمية DE، تهدف الدراسة إلى فهم تأثيرها المشترك على كفاءة الخوارزمية. علاوة على ذلك، يحتمل أن يتم تطبيق هذا البحث على مجالات مختلفة تتطلب التحسين، بما في ذلك تتبع نقطة الطاقة القصوى (MPPT).

الكلمات المفتاحية: التحسين العام، خوارزميات ميتا-هيورستيك (GA، ABC، PSO)، التطور التفاضلي، معاملات التحكم، الأصناف التفاضلية، وظائف الاختبار، جودة التقارب، سرعة التقارب، وقت التنفيذ.

Abstract:

This thesis aims to investigate the impact of control parameters and variants of the DE algorithm on its performance. By analyzing parameters like population size and mutation rates, alongside exploring various DE algorithm variants, the study aims to understand their combined influence on algorithm efficiency. Moreover, this research has the potential to be applied to various fields requiring optimization, including Maximum Power Point Tracking (MPPT).

Keywords: Global Optimization, Meta-heuristic Algorithms (GA, ABC, PSO), Differential Evolution (DE), Control Parameters, DE Variants, Benchmark Functions, Convergence Quality, Convergence Speed, Execution Time.

Résumé :

Cette thèse vise à étudier l'impact des paramètres de contrôle et des variantes de l'algorithme DE sur ses performances. En analysant des paramètres tels que la taille de la population et les taux de mutation, ainsi qu'en explorant différentes variantes de l'algorithme DE, l'étude vise à comprendre leur influence combinée sur l'efficacité de l'algorithme. De plus, cette recherche a le potentiel d'être appliquée à divers domaines nécessitant une optimisation, y compris le suivi du point de puissance maximale (MPPT).

Mots clés : Optimisation globale, Algorithmes méta-heuristiques (AG, ABC, PSO), Évolution différentielle (ED), Paramètres de contrôle, Variantes d'évolution différentielle (ED), Fonctions de benchmark, Qualité de convergence, Vitesse de convergence, Temps d'exécution

List of abbreviations:

ABC	Artificial Bee Colony
ACO	Ant Colony Optimization
AIS	Artificial Immune Systems
CA	Cultural Algorithms
CR	Mutation Rate
D	Number of Dimensions.
DE	Differential Evolution
EA	Evolutionary Algorithms
ES	Evolution Strategies
f	Objective Function
F	Scaling Factor
GA	Genetic Algorithms
GEN	Nombre of Iterations.
LP	Linear programming.
MPPT	Maximum Power Point Tracking
NLP	Non-Linear Programming.
NP	Non-deterministic Polynomial
NP	Population Size.
OP	Optimization Problem
P	Polynomial
PSO	Particle swarm optimization.
SA	Simulated Annealing.
TS	Tabu Seach
TSP	Traveling Salesman Problem

List of Figures

List of Tables

List of Algorithms

General introduction

General introduction

Photovoltaic (PV) systems are a vital renewable energy technology that converts sunlight into clean electricity. Mathematical optimization techniques play a crucial role in enhancing the efficiency of these systems by improving their design, operation, and management. Key aspects include determining the optimal sizing of components, adjusting panel orientation and tilt, implementing solar tracking systems, and using Maximum Power Point Tracking (MPPT). Additionally, optimization aids in energy management and storage, grid integration, maintenance planning, and microgrid design. These optimizations help maximize energy production, minimize costs, and promote environmental sustainability.

In the Information Age, characterized by an explosion of data and relentless efficiency demands, optimization techniques have become indispensable assets. From streamlining organizational processes to maximizing individual decision-making, optimization empowers us to do more with less. These techniques, encompassing finding the minimum or maximum value (e.g., cost, time, output) within a set of constraints, find applicability across diverse domains like agriculture, finance, engineering, and science. Real-world examples include optimizing construction works, financial portfolios, marketing campaigns, and agricultural water management. [1]

However, traditional optimization methods often struggle with complex problems marked by discontinuities, dynamic changes, multiple objectives, or stringent constraints. As a result, modern optimization relies on metaheuristics, a class of general-purpose algorithms employing computational methods. These algorithms, inspired by natural phenomena like evolution or social behaviors, utilize a guided random search approach to iteratively refine initial solutions and achieve optimal outcomes. This makes them particularly valuable for tackling intricate challenges that classical methods find difficult. One such powerful metaheuristic algorithm is Differential Evolution (DE). DE's strength lies in its ability to effectively navigate complex search spaces. Unlike classical methods, DE leverages a population of potential solutions and a set of control parameters (e.g., population size, mutation rates) to iteratively explore the solution space and converge on the optimal outcome. These control parameters, along with the specific DE variant chosen, significantly impact the algorithm's effectiveness. However, the intricate relationship between these elements and their combined influence on DE's overall performance remains an area for further exploration.

This thesis investigates the impact of control parameters and variants on DE's performance. By meticulously analyzing these factors, the study aims to unlock new avenues for optimizing the

algorithm. This optimization has the potential to improve DE's efficiency in solving problems across numerous fields. One such promising application lies in the realm of renewable energy, specifically in optimizing Maximum Power Point Tracking (MPPT) for solar panels. Solar panels exhibit varying levels of efficiency depending on environmental conditions. MPPT algorithms ensure these panels operate at their maximum power point, maximizing energy output. By optimizing DE for MPPT applications, we can potentially contribute to cleaner and more efficient energy production.

Chapter 1: discussed Optimization problems.

chapter 2: we delved deeper into metaheuristics and Genetic Algorithms.

chapter 3: we extensively covered Differential Evolution

chapter 4: Simulations and Results

Chapter I

Optimization problems

Chapitre I. **Optimization problems**

I.1. Introduction

Optimization is a fundamental discipline with widespread applications across various domains. This chapter serves as a foundational introduction to the comprehensive exploration of optimization methods. We will delve into essential concepts, principles, and diverse approaches utilized in addressing optimization challenges. Our journey begins by elucidating the core principles of optimization. This encompasses understanding the nature of optimization problems, categorizing them into various types such as linear, non-linear, single-objective, multi-objective, with or without constraints, and establishing criteria for assessing solution efficacy.

I.2. Historical Review

Although rigorous mathematical analysis of optimization problems was conducted throughout the 20th century, the roots of this field can be traced back to around 300 B.C. when the Greek mathematician Euclid evaluated the minimum distance between a point and a line. In 200 B.C., another Greek mathematician, Zenodorus, demonstrated that a semicircle has the maximum area for a given perimeter when bounded by a line.

In the 17th century, French mathematician Pierre de Fermat laid the foundation for calculus by showing that the gradient of a function vanishes at its maximum or minimum points. Advancing further, Newton and Leibniz developed the calculus of variations, a method dealing with the maxima and minima of functionals. In the 18th century, Euler and Lagrange provided rigorous mathematical details on the calculus of variations. Subsequently, Gauss and Legendre developed the least squares method, still extensively used today, and Cauchy introduced the steepest descent method for solving unconstrained optimization problems.

The first textbook on optimization, authored by Harris Hancock, was published in 1917. In 1939, Leonid Kantorovich presented the linear programming (LP) model and an algorithm for solving it. A few years later, in 1947, George Dantzig introduced the simplex method for solving LP problems. Kantorovich and Dantzig are considered pioneers who made significant breakthroughs in optimization techniques. The conditions for constrained optimization were unified by Harold Kuhn and Albert Tucker in 1951, and earlier by William Karush in 1939.

Richard Bellman established the principles of dynamic programming, which involves breaking down complex problems into smaller subproblems. Ralph Gomory made significant contributions to integer programming, where design variables can take integer values such as 0 and 1. The advent of computers in the 1980s allowed for the solving of many large-scale problems. Present-day optimization problems are multidisciplinary and multiobjective, and solution techniques now

include not only gradient-based algorithms but also nontraditional methods like genetic algorithms, ant colony optimization, and particle swarm optimization, which mimic natural processes.

Today, optimization methods are essential for solving problems across all disciplines, including economics, science, and engineering. Due to stiff competition in virtually all fields, the role of optimization has become even more critical as it aims to minimize costs and allocate resources efficiently [2].

I.3. Definition of optimization

Optimization is the process of systematically selecting the best possible solution from a range of feasible options, considering various constraints and objectives. It is often an iterative process that involves researching, evaluating, and adjusting different candidate solutions until a satisfactory solution is found. Feasible solutions are those that meet all the specified constraints of the optimization problem. This process often involves minimizing costs, maximizing efficiency, or achieving other desired outcomes. The core element we aim to improve in optimization problems is called the objective function, also known as the performance index. This function represents a specific value we want to either minimize (like cost) or maximize (like profit or efficiency). It can encompass various quantifiable aspects like size, shape, weight, or output, depending on the problem. It's no surprise that minimizing costs or maximizing profits are frequent goals for many organizations.[2]

According to certain dictionaries, the French verb "optimiser" originated from England around 1844, where "to optimiste" meant "to act optimistically." Hence, an optimizer is akin to an optimist who continually seeks improvement: "minimize a cost," "maximize a profit," "optimize a process," "gain by optimizing," etc. All these phrases pertain to the relatively young mathematical field of optimization and its applications. [9]

Optimization is surprisingly ubiquitous. We strive to optimize personnel schedules, teaching styles, economic systems, game strategies, even biological and healthcare systems. Its appeal lies not only in its algorithmic and theoretical depth but also in its broad applicability across various domains. it can be applied in many fields, such as solar energy, engineering, economics, finance, logistics, planning, system design, resource management, operations research, machine learning, and many more. It is used to solve complex problems that require effective decision-making and optimal use of available resources. To find the optimal solution, various optimization methods are used, ranging from mathematical and analytical approaches to heuristic and evolutionary methods. [4]

I.3.1. General formulation of a function to be optimized:

Mathematical Optimization involves creating and solving a constrained optimization problem, typically expressed in the general mathematical form:

$$\underset{\text{w.r.t. } \mathbf{x}}{\text{minimize}} f(\mathbf{x}),\ \mathbf{x} = [x_1, x_2, \dots, x_n]^T \in \mathbb{R}^n \qquad \text{Equation I–1}$$

subject to the constraints:

$$\begin{aligned} g_j(\mathbf{x}) &\leq 0, \quad j = 1,2,\dots,m \\ h_j(\mathbf{x}) &= 0, \quad j = 1,2,\dots,r \end{aligned} \qquad \text{Equation I–2}$$

In this context, $f(\mathbf{x})$, $g_j(\mathbf{x})$, and $h_j(\mathbf{x})$ are scalar functions dependent on the real column vector x.

The continuous components x_i of $\mathbf{x} = [x_1, x_2, \dots, x_n]^T$, referred to as design variables, form the basis of the optimization. The function $f(\mathbf{x})$ is known as the objective function, $g_j(\mathbf{x})$ represents the inequality constraints, and $h_j(\mathbf{x})$ signifies the equality constraints.

The optimal solution to problem (1.1) is referred to as $\mathbf{x}^*$ and it is the vector $\mathbf{x}$ that minimizes or maximizes $(\mathbf{x})$, yielding the optimal function value $f(\mathbf{x}^*)$. If no constraints are specified, this is termed an unconstrained minimization problem.[5]

I.4. Optimization Problem Handling:

In various domains, including science, engineering, and management, decision-making is a ubiquitous task. As the world grows increasingly complex and competitive, the necessity for rational and optimal decision-making becomes paramount. The decision-making process typically involves the following steps: (see Fig.1).

I.4.1. Problem Formulation:

Initially, the decision problem is identified, and an initial statement of the problem is articulated. This stage may involve multiple decision-makers outlining internal and external factors along with the objectives of the problem. However, the formulation may be imprecise at this stage.

I.4.2. Problem Modeling:

Subsequently, an abstract mathematical model is constructed for the problem. The modeler may draw inspiration from existing models in the literature, aiming to simplify the problem into well-studied optimization models. Often, the models used for solving are simplified representations of reality, incorporating approximations and omitting complex processes that are challenging to represent mathematically.

I.4.3. Problem Optimization:

Once the problem is modeled, optimization procedures are employed to generate a "good" solution. This solution may either be optimal or suboptimal. It's important to note that the solution is derived for an abstract model of the problem rather than the original real-life scenario. Therefore, the performance of the obtained solution serves as an indication, particularly when the model accurately reflects the problem. Algorithm designers may utilize existing algorithms designed for similar problems or tailor algorithms based on the specific application's knowledge.

I.4.4. Solution Implementation:

The obtained solution is practically tested by the decision-maker and implemented if deemed "acceptable." Practical knowledge may be integrated into the solution for implementation purposes. If the solution is deemed unacceptable, the model and/or optimization algorithm may require refinement, and the decision-making process is reiterated[6]

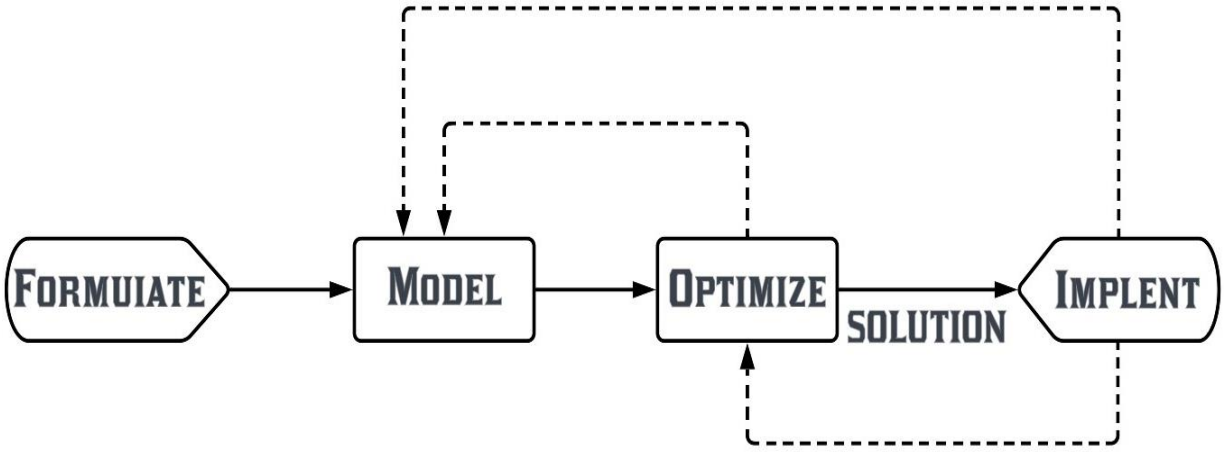

Figure 1. The classical process in decision making: formulate, model, solve, and implement. In practice, this process may be iterated to improve the optimization model or algorithm until an acceptable solution is found. Like life cycles in software engineering, [6]

I.5. Types of optimization problems:

As shown in Fig.2, optimization problems come in many forms, distinguished by various factors such as the number of objective functions, the types of objective functions and constraints, the types of variables involved, and whether we seek global or local optimization.

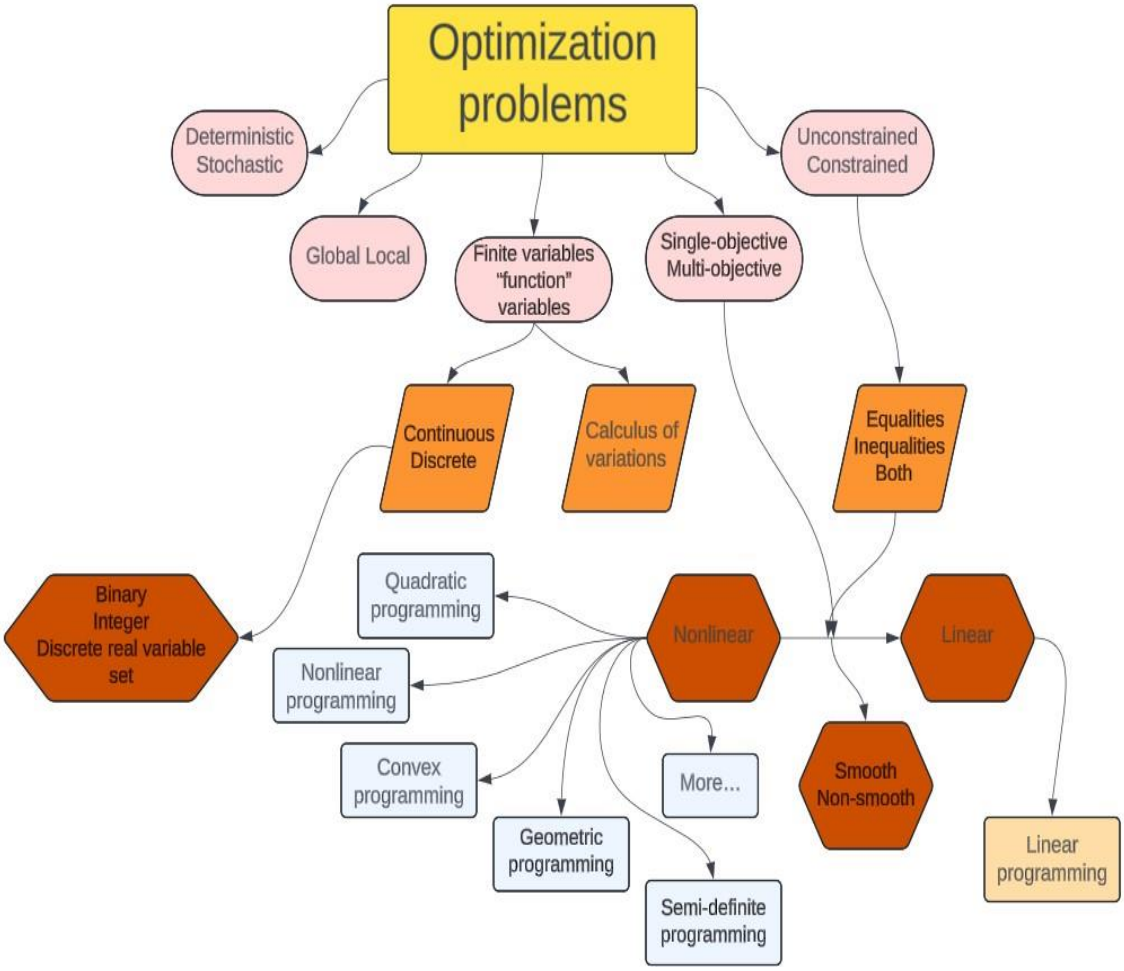

Figure 2.Calculus of Optimization problems[7]

I.5.1. Classification based on the objective function:

The formulation of optimization problems can often seem ambiguous due to the diversity of terminology used and the potential confusions this can cause. To clarify these concepts, we have adopted the following terminology:

A single-objective optimization problem is defined by a set of variables, an objective function, and a set of constraints.

A multi-objective optimization problem is defined by a set of variables, a set of objective functions, and a set of constraints.

I.5.1.1. Single-objective optimization problem :

A single-objective optimization problem is a type of optimization problem where the goal is to find the best solution among a set of possibilities by maximizing or minimizing a single objective

function. In this type of problem, there is only one criterion or performance measure to optimize. The aim is to determine the values of the decision variables that lead to the best possible value of the objective function while adhering to the specified constraints. Optimizing this problem can guarantee the optimality of the solution found, but it identifies only one optimal solution. In real-world situations, decision-makers usually need multiple alternatives. [8]

I.5.1.2. Multi-objective optimization problem:

A multi-criteria optimization problem, also known as a multi-objective optimization problem, involves optimizing multiple objective functions simultaneously. Unlike a single-objective optimization problem, which focuses on optimizing one performance measure, a multi-objective optimization problem seeks to find a set of solutions that balance trade-offs among various objectives.[9]

The multi-objective optimization problem aims to find a set of solutions called the PARETO boundary; a solution is said to be Pareto-optimal if it cannot be improved in a goal without deteriorating at least one other goal. Solving these problems is complex because there is no single optimal solution, but rather a set of alternative solutions. Specific methods are used to explore and represent the PARETO boundary. [6]

I.5.2. Classification based on constraints:

I.5.2.1. Optimization problem with constraints:

A Constraint Optimization Problem is a type of optimization problem where potential solutions must adhere to specific constraints or conditions. The goal is to find the best possible solution that satisfies all the problem's constraints. These constraints can include limits on decision variables (mandatory or structural), relationships between variables, and equations or inequalities that must be met (indicative or soft).

There are two main types of constrained optimization problems: Linear Programming (LP) problems and Non-Linear Programming (NLP) problems. In linear programming problems, the objective function and constraints are all linear, whereas in non-linear programming problems, at least one of the functions is non-linear.

A Constraint Optimization Problem is defined by the quadruplet (X D C f) with:

- X= $\{x_2, x_1, \cdots, x_n\}$ It is the set of variables of the problem
- D= {D (x_1), ⋯, D (x_n)} is the set of domains
- C= $\{C_1, \cdots, C_k\}$ represents the set of constraints
- f is an objective function defined on a subset of X (to be minimized or maximized) [10].

I.5.2.2. Unconstrained optimization problem:

This is a type of optimization problem where there are no constraints to respect. The absence of constraints means that all variable values are considered feasible and acceptable. Thus, the search for the optimal solution focuses solely on maximizing or minimizing the objective function. Unconstrained optimization problems can be formulated as mathematical programming problems, these problems can be linear, non-linear, convex, or non-convex depending on the nature of the cost function [10].

I.5.3. Classification based on Both the objective and constraints:

I.5.3.1. Linear optimization problems:

This type of problem involves a linear function to be minimized or maximized, under linear constraints. The constraints and variables are generally represented by linear equations.

A linear optimization problem is a type of optimization problem where the objective function is a linear function that we seek to minimize or maximize, while the constraints are linear inequalities or equalities (i.e., first-degree) that must be satisfied.[11]

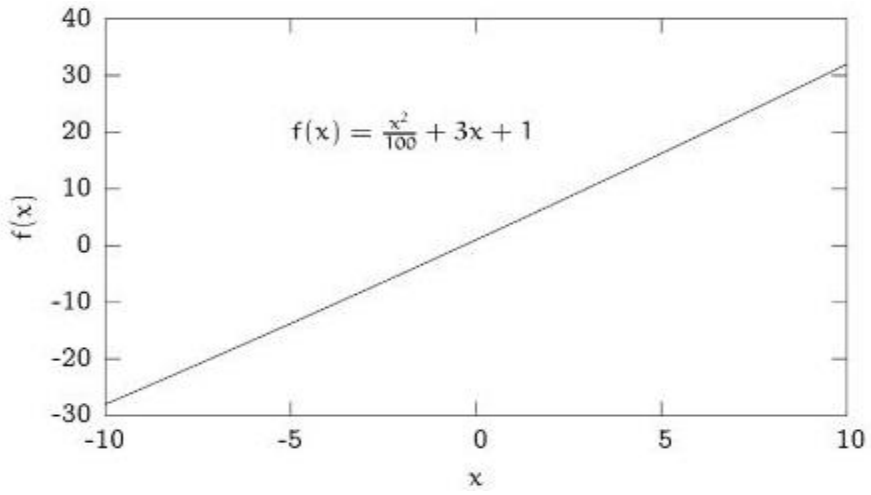

Figure 3.Example of an almost linear function[12]

I.5.3.2. Quadratic programming problems:

Quadratic programming (QP) problems are a class of mathematical optimization problems that involve quadratic objective functions and linear constraints. Quadratic programming problems arise in a variety of fields, including finance, engineering, operations research, and machine learning. They are used when the objective function or constraints have quadratic forms, which often occurs in situations where there are quadratic relationships between variables.

Solving quadratic programming problems can be more computationally intensive compared to linear programming due to the presence of quadratic terms. Various algorithms exist for solving quadratic programming problems efficiently, including interior point methods, active-set methods, and gradient-based methods. The choice of algorithm depends on the problem size, structure, and desired accuracy. [6]

I.5.3.3. Nonlinear optimization problems:

A nonlinear optimization problem is characterized by having a nonlinear objective function or constraints, or both. This implies that the objective function and/or constraints may include nonlinear terms like products, powers, trigonometric functions, exponential functions, etc.

Nonlinear optimization problems are generally more complex to solve than linear optimization problems, as they can have more complicated mathematical properties and can have multiple local optima. Specific optimization techniques, such as gradient methods, derivative-free optimization methods, local search methods, or genetic algorithms, are used to solve these problems. [13]

I.5.3.4. Geometric programming problems:

Geometric programming (GP) is a specific class of mathematical optimization problems where the objective function and the constraints are defined using posynomials and monomials. Geometric programming problems are particularly useful in fields such as engineering, finance, and biology where many real-world problems can be naturally expressed in terms of posynomials. These problems often arise in situations where there are multiplicative relationships between variables Solving geometric programming problems typically involves specialized algorithms due to their unique structure. These algorithms often involve transformations to convert the problem into a standard form, followed by techniques such as interior point methods or sequential quadratic programming to find the optimal solution. [2]

I.5.3.5. Convex optimization problem:

Convex optimization is a subclass of optimization problems where the objective function is convex and the feasible region, defined by the constraints, forms a convex set. This means that any local minimum is also a global minimum, which significantly simplifies the problem-solving process. A convex optimization problem can be generally formulated as follows:

$$\text{minimize} \quad f_0(x) \qquad \text{Equation I–3}$$

$$\text{subject to} \quad f_i(x) \leq 0, i = 1, \dots, m \qquad \text{Equation I–4}$$

$$h_i(x) = 0, i = 1, \dots, p$$

In this formulation, $f_0(x)$ is the convex objective function, $f_i(x)$ are the convex inequality constraint functions, and $h_i(x)$ are the affine equality constraint functions. The decision variable x typically belongs to a convex set defined by these constraints.

Convex optimization is widely used in various fields such as economics, engineering, machine learning, and finance because it ensures efficient and reliable solutions even for large-scale problems.[14]

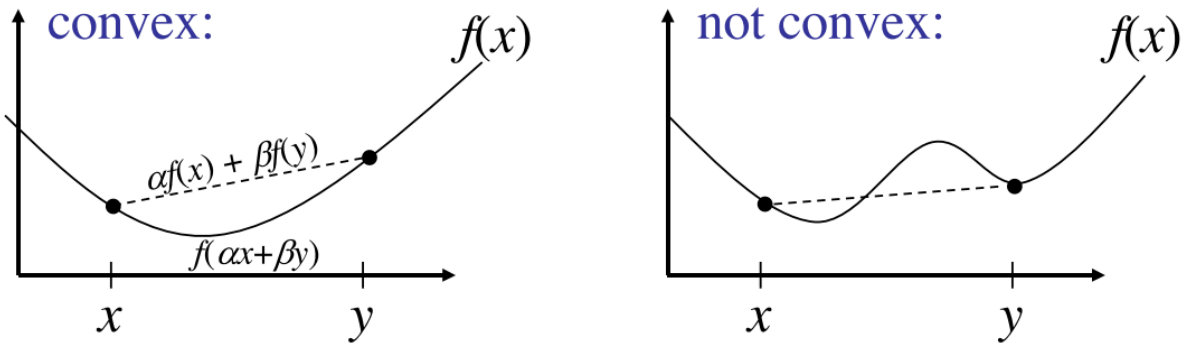

Figure 4 .not convex or Convex optimization [15]

I.5.3.6. Differentiable optimization problem:

This is a type of optimization problem where the objective function and the constraints are all differentiable. This means that the partial derivatives of these functions with respect to the decision variables can be calculated. The differentiability of the functions makes it possible to use these derivatives to provide information on the directions to be followed to improve the solution. In mathematics:

Let $f: \mathrm{IR}^n \rightarrow$ IRa continuous function. If, for all d $\in \mathrm{IR}^n$, the directional derivative of f in the direction d exists, then the function f is said to be differentiable. The most commonly used differentiable optimization methods include the gradient method, Newton's method, quasi-Newton's method, and Lagrange multiplier's method. These methods use the derivatives of the functions to iterate and gradually find the optimal solution.[8]

I.5.3.7. Undifferentiable optimization problem:

This is the term under which optimization problems are listed, where certain data-functions are not (everywhere) differentiable. This happens more frequently than one imagines; And even if the area or the data are not differentiable concern only "very few" points, it is often these points that interest the optimizer (they are in fact unavoidable). As an example, consider an objective-function of the form:f

$$f(x) = max\{f_1(x), \ldots, f_m(x)\}$$ Equation I–5

Where the functions are differentiable (this is a form of criteria involved in approximation or mathematical economics f_i [11]

I.5.4. Classification based on the nature of optimization:

I.5.4.1. Global optimization problem:

These problems aim to find the optimal solution from all possible options, rather than being confined to a specific subset. Global optimization becomes crucial when the objective is to find the smallest value across the entire feasible domain. However, locating the global minimum presents a significant challenge, primarily due to the intricate nature of the search space, which is often large and filled with many local optima. This task is further complicated by the limitations of conventional optimization methods, which frequently struggle to ensure the discovery of the global minimum. Moreover, the feasibility of employing global optimization strategies is restricted to specific problem types, adding another layer of complexity to the task [16] **[7]** .

I.5.4.2. Local optimization problems:

Local optimization problems focus on finding satisfactory solutions nearby a specific point, often called a local minimum. This point marks the smallest value within a limited neighborhood **[7]**.

The core idea behind local search is to continuously adjust solutions within their nearby area. This method defines changes through neighborhoods, which include all solutions reachable with just one step from the current solution. In essence, neighborhood search means moving step by step from one solution to another by doing certain actions [17].

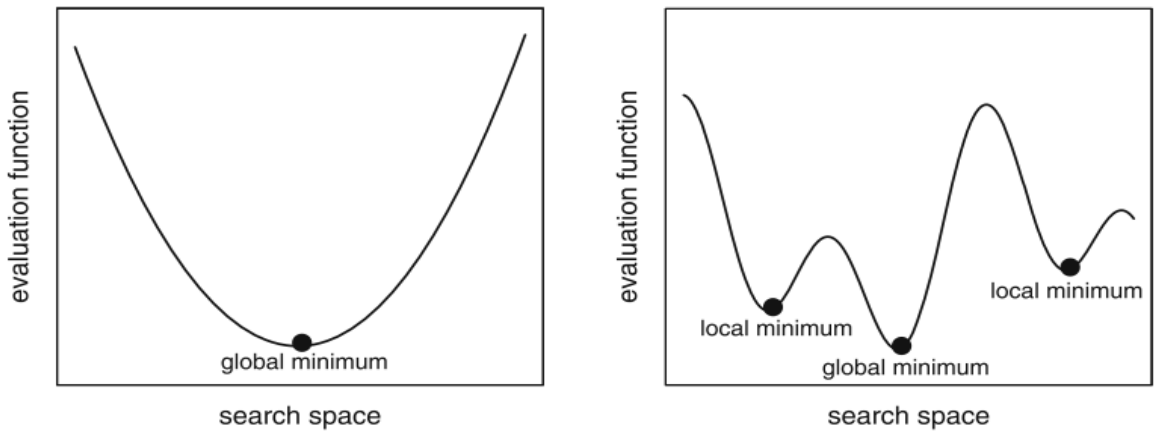

Figure 5. Example of a convex (left) and non-convex (right) function landscapes including the global and local minimums [1]

I.5.4.3. Deterministic optimization problems:

Deterministic optimization problem: Deterministic optimization problems involve fixed variables and parameters, providing a predictable framework for finding optimal solutions without considering randomness or uncertainty. Traditional optimization algorithms, such as linear programming and dynamic programming, are commonly used to address deterministic optimization challenges by systematically exploring the solution space based on predetermined conditions and constraints.

I.5.4.4. Stochastic optimization problems:

These problems involve random variables and objective probabilistic functions. Stochastic optimization techniques take in to account uncertainty and variability in the data, variables, or the process itself. Unlike deterministic optimization problems, where variables and parameters are fixed. [6]

An optimization problem can be stochastic if the variables involved are random. However, non-deterministic methods can be employed to solve deterministic problems. Techniques like genetic algorithms, simulated annealing methods, and Monte Carlo search are examples of stochastic approaches used to tackle these optimization challenges.[7]

I.5.4.5. Dynamic optimization problem :

This is a type of optimization problem where the decisions to be made evolve over time in a sequential manner. Unlike static optimization problems, where all decisions are made simultaneously, they take in to account temporal interdependencies and allow adaptive decisions to be made at each stage based on the state of the system. The goal is to find a sequence of decisions that maximizes or minimizes an objective function over a given period of time. The decisions made at each stage can influence the state of the system and future decisions.[6]

I.5.5. Classification based on types of variables:

I.5.5.1. Continuous or discrete variables:

In discrete optimization problems, variables are limited to specific values, unlike in continuous optimization problems where they can have any value within a range. This difference highlights the common occurrence of discrete optimization challenges across various fields like logistics, telecommunications, and computer science. For example, in network routing problems, decisions often involve choosing specific paths between nodes. Similarly, in telecommunications network design, decisions about component placement or resource allocation must be made within budget constraints. These decisions require precise variable values, such as binary variables representing

yes/no choices or integer variables for whole numbers. These constraints are visible in applications like determining bearing sizes or screw threads, where only certain integer values are allowed, avoiding fractions or arbitrary choices. Discrete optimization often relies on specialized methods and algorithms to address the unique challenges it presents.[7][18]

I.5.5.2. Finite variables

In numerous real-world optimization scenarios, the variables that dictate decision-making are confined to a finite range of potential values. For instance, in scheduling dilemmas, these variables may signify the allocation of tasks to machinery, where each task is exclusive to a particular machine from a limited selection. Likewise, within inventory management, decision variables might denote the quantities of various items for stocking, subject to constraints such as available storage capacity and projected market demands. The process of finite variables optimization revolves around enhancing outcomes within such discrete decision frameworks, often necessitating the application of specialized algorithms and solution methodologies.[18]

I.5.5.3. Combinatorial optimization problem:

These are complex optimization problems in which the search for the best solution involves finding the best combination among a finite set of possibilities. These problems are characterized by a combinatorial explosion, which means that the total number of possible solutions increases rapidly with the size of the problem. These problems are usually difficult to solve exactly, as it is often not possible to explore all the solutions comprehensively due to the computation time required. Therefore, heuristic algorithms and approximation methods are commonly used to find good quality solutions in a reasonable time [6].

I.6. Complexity of problems:

The complexity of a problem corresponds to the complexity of the most efficient algorithm that solves it. A problem is considered tractable (or easy) if it can be solved by an algorithm that runs in polynomial time. Conversely, a problem is deemed intractable (or difficult) if no polynomial-time algorithm can solve it.

A polynomial-time algorithm is an algorithm whose execution time is limited by a polynomial function based on the size of the problem's input. The temporal complexity of the algorithm increases reasonably as the size of the problem increases. This means that the algorithm can solve large problems efficiently. [11]

Complexity theory primarily addresses decision problems, which can be answered with a simple yes or no. For example, in the traveling salesman problem, the optimization problem is "find the optimal Hamiltonian tour that minimizes the total distance," while the corresponding decision problem is "given an integer D, is there a Hamiltonian tour with a total distance less than or equal to D?"

A key aspect of computational theory is classifying problems into complexity classes. A complexity class encompasses all problems that can be solved with a specific amount of computational resources. Two important complexity classes are P and NP.

I.6.1. P Class:

The complexity class P encompasses all decision problems solvable by a deterministic machine in polynomial time. An algorithm is polynomial for a decision problem if its worst-case complexity is bounded by a polynomial function of the input size. Therefore, class P includes problems that have known polynomial-time algorithms, making them relatively "easy" to solve. Examples of such problems include the minimum spanning tree, shortest path problems, maximum flow network, maximum bipartite matching, and continuous models of linear programming. [6]

I.6.2. NP Class:

The complexity class NP includes all decision problems that can be solved by a nondeterministic algorithm in polynomial time, (ex: knapsack decision problem). A nondeterministic algorithm features one or more points where multiple possible paths can be chosen without specifying which one to follow. This type of algorithm uses primitives such as choice, which proposes a potential solution (oracle); check, which verifies in polynomial time if the proposed solution (certificate) is correct; success, indicating a "yes" answer after verification; and fail, indicating the absence of a "yes" answer. If the choice primitive suggests a solution that yields a "yes" and the oracle can achieve this, then the computational complexity is polynomial. [6]

I.6.3. NP-complete:

A decision problem A belonging to NP is classified as NP-complete if all other problems within the NP class can be reduced to problem A in polynomial time. Figure 5 illustrates the interconnection between P, NP, and NP-complete problems. If there exists a polynomial deterministic algorithm to solve an NP-complete problem, then all problems within the NP class can be solved in polynomial time. [6]

I.6.4. NP-Hard:

NP-Hard problems are decision problems for which there is no known algorithm that can solve them in deterministic polynomial time and their corresponding decision problems are NP-complete. They are considered to be among the most difficult problems in theoretical computer science and they present the majority of real-world optimization, demanding exponential time for optimal solutions unless P equals NP. Metaheuristics emerge as a crucial alternative for tackling this class of problems. [6]

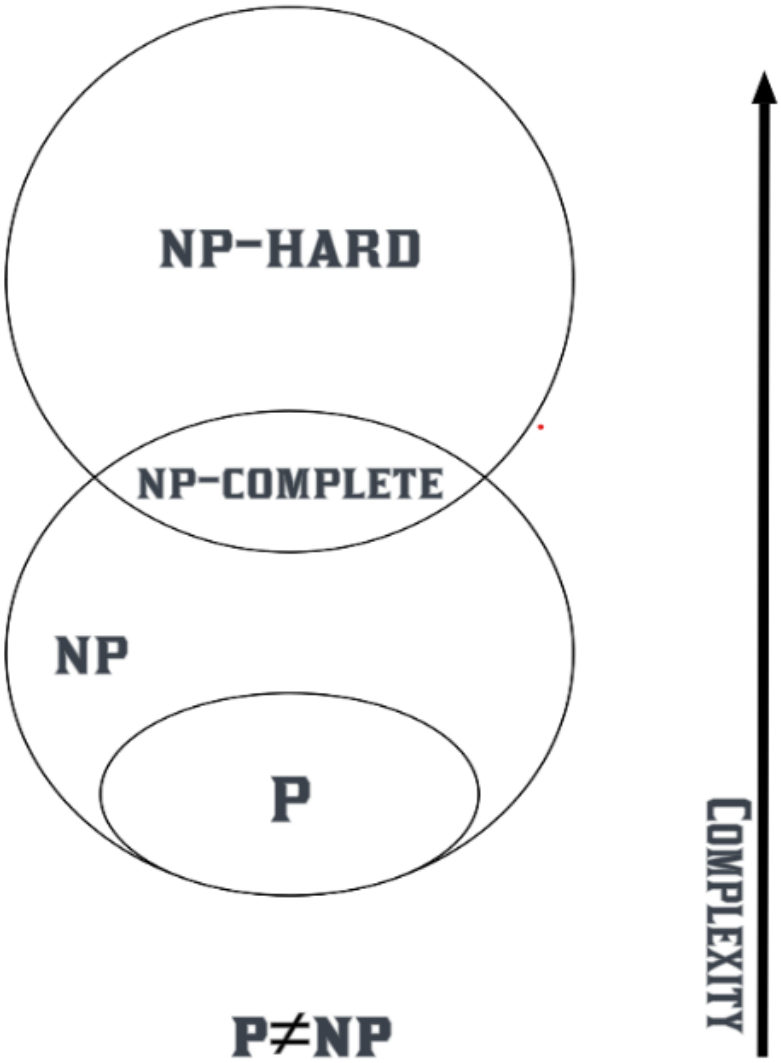

Figure 6. Complexity classes of decision problems. [6]

I.7. Optimization Methods:

The methods of solving optimization problems can be grouped into two main families, which are exact methods and approximate methods (as shown in Fig.7):

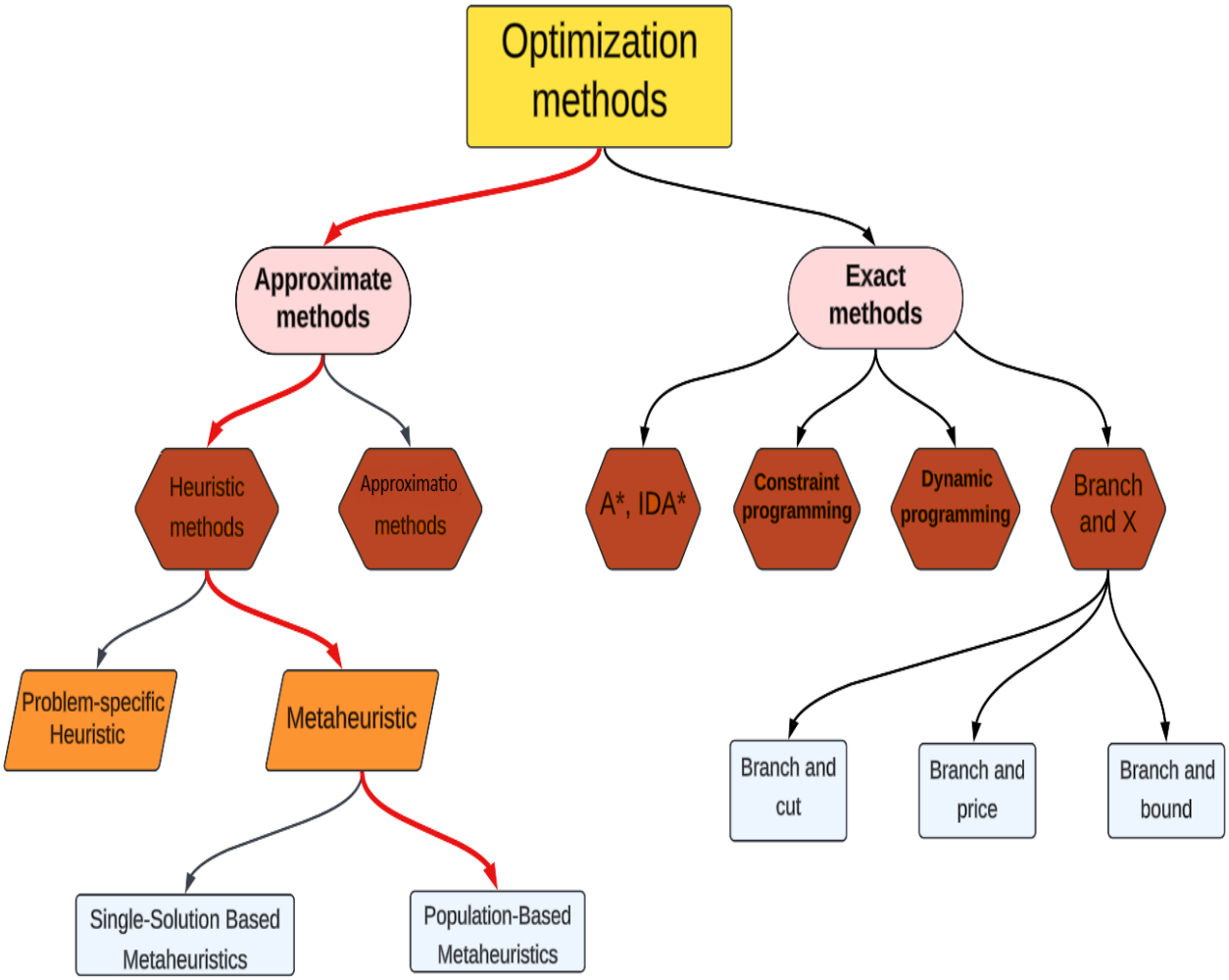

Figure 7.Classical optimization methods. Integration.[6]

I.7.1. Exact methods:

Within the category of exact methods, several classical algorithms can be identified: dynamic programming; the branch and X family of algorithms (including branch and bound, branch and cut, and branch and price) developed in the operations research field; constraint programming; and the A∗ family of search algorithms (such as A∗ and IDA∗ iterative deepening algorithms) originating from the artificial intelligence field. These methods, called complete, evaluate the search space in its entirety through intelligent enumeration. They guarantee finding the optimal solution for a finite-sized instance within a limited time and allow for proving its optimality. These

enumerative techniques can be considered tree search algorithms, exploring the entire relevant search space by breaking it down into simpler subproblems. [19] [8]

In the class of exact methods, we find the following classical algorithms:

- dynamic programming
- branch and X
    - ✓ branch and bound
    - ✓ branch and cut
    - ✓ branch and price
- constraint programming
- A*, IDA*.

I.7.2. **Approximate methods:**

In the class of approximate methods, two subclasses of algorithms can be distinguished: approximation algorithms and heuristic algorithms. Unlike heuristics, which typically find reasonably good solutions within a reasonable time frame, approximation algorithms provide provable solution quality and run-time bounds. [8]

I.7.2.1. Approximation algorithmes :

These are algorithms designed to find near-optimal solutions to optimization problems within a guaranteed bound of the optimal solution. They are particularly useful for NP-hard problems, where finding the exact optimal solution is computationally infeasible. The quality of an approximation algorithm is often measured by its approximation ratio, which compares the solution provided by the algorithm to the optimal solution.

Approximating an NP-hard optimization problem involves obtaining information about its optimal solution S_{0pt} without knowing it. Generally, this entails solving a simpler problem than the original, then transforming the optimal solution $S_{Opt'}$ of the simplified problem into a solution S of the original problem (see Figure 8). The challenge lies in guaranteeing a maximum distance between the approximation s and the optimum S_{Opt} of the original problem. [20]

The simplified problem is often obtained by relaxing constraints from the original problem. More formally, an approximation algorithm, denoted as p-approximation, is a polynomial algorithm that returns a guaranteed approximate solution, at worst-case factor p of the optimal solution. Sometimes, the approximation factor depends on the size of the problem instance. However, for some NP-hard problems, it is impossible to make approximations. Moreover, these algorithms are

problem-specific, and the provided approximations are often too far from the optimal solution, limiting their usefulness for many real-world applications.[20]

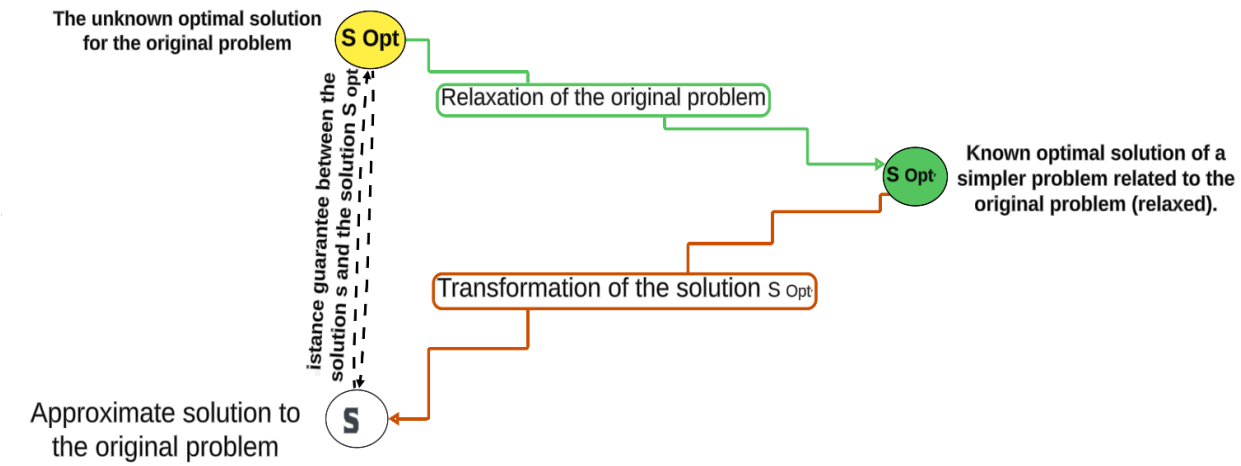

Figure 8.Principle related to approximation algorithms. [20]

I.7.2.2. Heuristics:

Heuristics are effective at finding good solutions for large problem instances. They enable acceptable performance at acceptable costs across a wide range of problems. Generally, heuristics do not offer an approximation guarantee on the solutions obtained. Heuristics can be categorized into two families: specific heuristics and metaheuristics. Specific heuristics are tailored to solve a particular problem or instance. Metaheuristics, on the other hand, are general-purpose algorithms that can be applied to almost any optimization problem. They serve as overarching methodologies that guide the design of underlying heuristics for solving specific optimization problems.[8]

I.7.2.2.1 Specific heuristic:

A specific heuristic is an approach to problem-solving and learning based on practical experience rather than theoretical knowledge. While it may not guarantee the best possible solution, it aims to produce satisfactory results within a reasonable amount of computational time. Specific heuristics are tailored to specific problems and often rely on methods such as rules of thumb, educated guesses, intuitive judgments, or common sense.[21]

Some important things to know about heuristics:[8]

- Heuristics are approximate resolution methods that provide solutions that are generally good, but not necessarily optimal.
- They are based on specific rules, strategies, or heuristic techniques that guide the process of finding solutions.

- Heuristics often simplify the optimization problem by focusing on the most important aspects and ignoring the less relevant details.
- They can be designed to consider specific constraints of the problem or to make use of specialized knowledge.
- Heuristics can be used to solve a wide variety of problems, such as scheduling, routing, resource optimization, scheduling, and more.
- They are often used when the exact methods are too time-consuming or resource-intensive.
- Heuristics can be iterative, that is, they gradually improve the solution by performing iterative steps of exploration and optimization.
- They can also use local search techniques to find solutions in complex search spaces.
- Heuristics do not usually provide mathematical guarantees about the quality of the solution obtained, but they are often evaluated empirically by comparing the results to optimal solutions when these are known.
- Problem-dependent (specific) heuristics can be connected to meta-heuristics to provide a concrete implementation of abstract optimization methods, the abstract part being represented by meta-heuristics.

I.7.2.2.2 Metaheuristic:

In 1986, Glover introduced the term "metaheuristic" to describe a collection of methodologies that transcend heuristics conceptually, as they influence the design of heuristics. A metaheuristic is a higher-level procedure or heuristic designed to discover, generate, or select a lower-level procedure or heuristic (partial search algorithm) that may yield a sufficiently satisfactory solution to an optimization problem. By exploring a large set of feasible solutions, metaheuristics often uncover good solutions with less computational effort compared to calculus-based methods or simple heuristics.[21]

A. Metaheuristics Based on a Single *Solution*:

Metaheuristics relying on a single solution comprise methodologies known as local search algorithms or neighborhood-based methods. These methods, categorized under Local Search (LS), are iterative algorithms that systematically explore the search space, transitioning from one solution to another incrementally. Among the most prominent techniques in this category are hill descent, Iterated Local Search (ILS), Simulated Annealing (SA), and Tabu Search (TS).[20]

B. Metaheuristics Based on a Population of Solutions:

Metaheuristics leveraging a population of solutions utilize this population as a means to enhance diversity and employ various evolutionary strategies. These strategies lead to methodologies such

as Differential Evolution (DE), Genetic Algorithms (GA), Scatter Search (SS), Ant Colony Optimization (ACO), and Particle Swarm Optimization (PSO).[20]

I.8. Optimization in Photovoltaic Systems

Optimization is crucial for enhancing the efficiency and performance of photovoltaic (PV) systems, which convert sunlight into electricity. Optimization techniques can improve various aspects of their design, operation, and management. Here are some examples of how mathematical optimization is used in PV systems:

System Design and Sizing: Optimization helps determine the optimal sizing of PV components, such as solar panels, inverters, and batteries. This involves finding the combination that maximizes energy production and minimizes costs over the system's lifetime, considering factors like solar radiation, load profiles, energy storage capacity, and equipment costs.

Orientation and Tilt Angle: The orientation and tilt angle of solar panels greatly influence their energy production. Optimization techniques can identify the best orientation and tilt angle to maximize energy output based on geographical location and season.

Tracking Systems: Solar tracking systems adjust the angle of solar panels to follow the sun's path, enhancing energy reception. Optimization can determine the best tracking strategy to maximize energy production, taking into account panel efficiency, tracking mechanism complexity, and maintenance costs.

Maximum Power Point Tracking (MPPT): PV panels have an optimal operating point called the maximum power point (MPP), which varies with environmental conditions. Optimization algorithms can dynamically track the MPP to ensure the panels operate at peak efficiency and produce maximum energy.

Energy Management and Storage: For PV systems with energy storage components like batteries, optimization aids in managing charge and discharge cycles to minimize costs, improve energy self-sufficiency, and ensure a reliable power supply.

Grid Integration: For grid-connected PV systems, optimization helps decide how much energy to consume, store, or feed back into the grid, balancing energy supply and demand while adhering to grid regulations and tariffs.

Maintenance Planning: Optimization schedules maintenance activities for PV systems, such as cleaning panels, repairing or replacing faulty components, and preventive maintenance, aiming to minimize downtime and costs while maximizing energy production.

Microgrid Design: In off-grid or remote areas, PV systems can be part of microgrids. Optimization helps design the microgrid layout by optimizing the use of renewable energy sources, diesel generators, and energy storage to meet energy demand while minimizing costs and environmental impact.

Predictive Analysis: Optimization techniques integrate predictive analysis and weather forecasts to anticipate variations in solar irradiation and adjust system parameters in advance, improving energy capture and grid integration.

Cost-Benefit Analysis: Optimization provides a comprehensive evaluation of different system configurations and operational strategies, helping decision-makers assess trade-offs between initial investment, operational costs, energy savings, and environmental benefits.

I.9. **Conclusion:**

This chapter has provided an overview of the diverse types of optimization problems that exist, along with the various methods employed to solve them. We have explored both complete methods, which guarantee finding an optimal solution, and approximate methods, which aim to find near-optimal solutions more efficiently. While complete methods, such as exact algorithms, are powerful and precise, they often become impractical for large-scale or highly complex problems due to their computational demands.

In contrast, approximate methods, including heuristics and metaheuristics, offer a pragmatic alternative by trading off some degree of optimality for significantly improved computational efficiency. These methods are particularly valuable in real-world applications where time and resources are limited.

In the next chapter, our focus will shift to incomplete (approximate) methods, with a specific emphasis on metaheuristics. Metaheuristics, such as genetic algorithms, simulated annealing, and ant colony optimization, are versatile and robust strategies designed to navigate large and complex search spaces effectively. They combine elements of randomness and structured search to explore potential solutions, making them highly adaptable to a wide range of optimization problems.

By delving into metaheuristics, we aim to uncover their underlying principles, strengths, and limitations, and demonstrate how they can be effectively applied to solve challenging optimization problems that are otherwise intractable using complete methods.

Chapter II

Meta-heuristics

Chapitre II. **Meta-heuristics**

II.1. **Introduction:**

The introduction of metaheuristics was a significant turning point in the field of optimization. It enables the attainment of high-quality solutions at a low cost for problems that classical methods cannot effectively solve. These versatile algorithms emerged in the 1980s and were designed to address various types of optimization problems: combinatorial, continuous, and mixed. Metaheuristics employ stochastic processes to explore the search space efficiently.

Metaheuristics are widely used across various fields, including renewable energy, electrical engineering, and mechanical engineering. For instance, in the field of renewable energy, they are used to optimize the placement and operation of wind turbines and solar panels to maximize energy production. In electrical engineering, they improve the design and operation of power grids to achieve better efficiency and stability. In mechanical engineering, they enhance the design of components and mechanical systems for optimal performance and reduced costs.

In this chapter, we will present the fundamental concepts of metaheuristic optimization algorithms, focusing on genetic algorithms (GA), differential evolution (DE), particle swarm optimization (PSO), and artificial bee colony (ABC). We will also discuss their applications in various domains, particularly in renewable energy, to highlight their versatility and effectiveness in solving complex optimization problems.

II.2. Basics and Applications:

Heuristic or approximate algorithms, a concept initially introduced by Polya in 1945, have been integral in addressing optimization problems for several decades. These algorithms, including metaheuristics, have gained rapid recognition within the research community and have been continuously evolving to tackle increasingly complex optimization challenges.

Metaheuristics, in particular, represent a family of stochastic methods designed to explore search spaces efficiently, leveraging random processes to combat the combinatorial explosion encountered in exact methods. Unlike traditional heuristics tailored to specific problems, metaheuristics are renowned for their adaptability across a wide range of problems without significant changes to their underlying algorithms, earning them the 'meta' qualifier. While they excel in optimizing problems with minimal information, they do not guarantee optimal solutions but rather provide approximations of the global optimum. These methods operate iteratively, repeating similar patterns throughout the optimization process, and are characterized by their direct nature, as they do not rely on gradient computation.

One of the key challenges for metaheuristics lies in simplifying method selection and configuration to best suit the problem at hand, addressing the demand for both speed and simplicity. Given the ongoing evolution in this field, numerous metaheuristic classes are continuously proposed, drawing from classical heuristic algorithms, artificial intelligence, biological evolution, neural systems, and statistical mechanics. Metaheuristics are considered an iterative generation process, intelligently combining different concepts to efficiently explore and exploit search spaces, aiming to find near-optimal solutions [22], [23].

Optimization problems are common and frequently intricate, making metaheuristics broadly applicable. The rising number of sessions, workshops, and conferences focused on metaheuristics underscores their growing significance within the research community [8].

II.3. Classification :

Meta-heuristic algorithms can be classified based on several criteria:

a) **Nature-inspired vs. non-nature-inspired:** Many metaheuristics draw inspiration from natural processes, such as evolutionary algorithms and artificial immune systems from biology, and swarm intelligence like ants, bees, and particle swarms, while others like simulated annealing are inspired by physics.

b) **Memory usage vs. memoryless methods:** Some metaheuristics, like local search, GRASP, and simulated annealing, are memoryless, meaning they do not use dynamically extracted information during the search. Others, like tabu search, utilize memory to store online-extracted information.

c) **Deterministic vs. stochastic:** Deterministic metaheuristics make decisions based on fixed rules, leading to the same solution from the same initial point (e.g., local search, tabu search). Stochastic metaheuristics apply random rules during the search, resulting in different solutions from the same initial point (e.g., simulated annealing, evolutionary algorithms).

d) **Iterative vs. greedy:** Iterative algorithms represent most of metaheuristic algorithms, they start with a complete solution (or a population of solutions) and transform it iteratively using search operators. While Greedy algorithms begin with an empty solution and assign decision variables step-by-step until a complete solution is achieved.

e) **Population-based vs. Single-solution Based Search:** Single-solution based algorithms, such as local search and simulated annealing, focus on iteratively improving one candidate solution throughout the search process. They are highly exploitation-oriented, intensively searching the local neighborhood of the current solution to refine it. In contrast, population-based algorithms,

like particle swarm optimization and evolutionary algorithms, evolve a whole population of solutions simultaneously. This approach enhances exploration by allowing the algorithm to search multiple areas of the search space in parallel, reducing the risk of getting stuck in local optima and increasing the chances of finding the global optimum.[8]

In this thesis, we focus on stochastic iterative population-based meta-heuristic algorithms. Hence, when referring to meta-heuristic algorithms, it is implied that we are discussing the stochastic iterative population-based variants.

II.4. Populations Based Meta-heuristic:

II.4.1. Fundamental concepts:

Population-based metaheuristics (P-metaheuristics) are optimization algorithms that use a population of solutions to explore and exploit the search space effectively. These methods are distinct from single-solution based metaheuristics (S-metaheuristics) due to their inherent ability to maintain and utilize a diverse set of potential solutions throughout the optimization process. This section delves into the fundamental concepts underlying P-metaheuristics, emphasizing the critical aspects that contribute to their performance and efficacy.

II.4.1.1. Initial population :

Due to the large diversity of initial populations, population-based metaheuristics (P-metaheuristics) are naturally more oriented towards exploration, whereas single-solution based metaheuristics (S-metaheuristics) focus more on exploitation. The determination of the initial population is often overlooked in the design of a P-metaheuristic. However, this step plays a crucial role in the algorithm's effectiveness and efficiency, warranting greater attention.

When generating the initial population, diversification is the main criterion to consider. If the initial population is not well diversified, premature convergence can occur in any P-metaheuristic. For instance, this might happen if the initial population is generated using a greedy heuristic or an S-metaheuristic (e.g., local search, tabu search) for each solution in the population.

II.4.1.2. Population size :

Population size plays a crucial role in population-based metaheuristic algorithms, impacting both solution quality and computational resources. Larger populations generally yield better solutions but require more computational resources, necessitating a balance between solution quality and computational efficiency.

Practical guidelines for determining population size vary based on the algorithm and problem dimensionality. For instance, Clerc suggests a population size of 20 to 30 for Particle Swarm Optimization (PSO), while Talbi recommends 20 to 100 for evolutionary algorithms. Storn and Price advise a population size of 5 to 10 times the problem's dimension for Differential Evolution (DE) algorithms.

Diversification in the initial population is vital to avoid premature convergence. Insufficient diversity can lead to suboptimal performance in any population-based metaheuristic algorithm.[6]

Some researchers propose dynamic strategies to adjust population size during algorithm execution instead of relying on preset rules. Michalewicz introduced the concept of chromosome "age" in Genetic Algorithms (GA), where fitter chromosomes have longer lifespans, resulting in varying population sizes. Clerc outlined adaptive Particle Swarm Optimization (PSO) methods, while adaptive Differential Evolution (DE) approaches adjust population size based on the current search state.[23]

II.4.1.3. Exploitation vs exploration :

Balancing exploitation and exploration is a fundamental concept in evolutionary algorithms, highlighting the need to strike a balance between two key strategies during the search for optimal solutions.

Exploitation, or local search, involves intensifying the search around promising solutions already identified. It's like focusing on refining current solutions or exploring their immediate surroundings to find better alternatives. While exploitation is beneficial for converging towards local optima, excessive reliance on it may lead to premature convergence if not counterbalanced effectively.

Exploration on the other hand, emphasizes diversifying the search space to uncover new regions possibly housing superior solutions. By continuously exploring different parts of the search space, exploration guards against getting stuck in local optima. Although exploration enhances the chances of discovering global optima, it may come with increased computational costs.

Finding the right balance between exploitation and exploration is paramount in evolutionary algorithms tackling multi-objective problems. Too much exploitation risks getting trapped in local optima, while overemphasis on exploration can lead to inefficient search. Hence, effective evolutionary algorithms dynamically adjust the trade-off between exploitation and exploration throughout the optimization process to ensure robust and efficient search performance.[24]

II.4.1.4. Number of Iterations:

the concept of the number of iterations is crucial for understanding the efficiency and performance of various algorithms. Iterations refer to the repeated application of a set of operations within an algorithm. The number of iterations can significantly impact both the speed and accuracy of the algorithm's outcome. For example, in graph traversal algorithms such as Breadth-First Search (BFS), the number of iterations determines how quickly the algorithm can explore the entire graph or reach a specific node. Efficient iteration management is vital for optimizing computational resources and ensuring the algorithm converges to a solution efficiently. Thus, controlling the number of iterations is essential for balancing performance and computational cost.[25]

II.4.1.5. stopping Criteria:

Several stopping criteria based on the evolution of a population can be used in optimization algorithms, with some being similar to those designed for single-solution metaheuristics (S-metaheuristics).[6]

- Static Procedure: In this approach, the end of the search is predetermined. For example, one might set a fixed number of iterations (generations), a limit on CPU resources, or a maximum number of objective function evaluations. This method is typically employed when there are time or resource constraints.
- Adaptive Procedure: Here, the end of the search cannot be known in advance. The criteria might include a fixed number of iterations without improvement, or stopping when an optimal or satisfactory solution is reached (e.g., achieving a specific error margin relative to the optimum or an approximation to it if a lower bound is known beforehand).

Some stopping criteria are specific to population-based metaheuristics (P-metaheuristics). These criteria are generally based on certain statistics of the current population or its evolution, and they are often related to the diversity of the population. For instance, the algorithm may stop when the diversity measure falls below a certain threshold, indicating that the population has stagnated. Continuing the execution of a P-metaheuristic under such conditions is usually unproductive.[6]

II.4.2. Solving Optimization Problems:

Let us reconsider the following generic NLP problem:

$$\min XFX \qquad \text{Equation II–1}$$

subject to:

$$\dot{G}(X) = 0 \qquad \text{Equation II–1}$$

$$H(X) \leq 0 \qquad \text{Equation II–2}$$

where $X \in \Re^{n_x}$ is the decision variable, F is the cost (objective) function, while G and H are the constraints functions.

Let $X \in \Re^{n_x}$ be the set of admissible solution X that satisfy constraints Equation II–2 and Equation II–1) and n_{pop} be the population size. The basic steps needed to solve the optimization problem (OP) Equation II–2 Equation II–3 using a population based meta-heuristic algorithm are summarized in algorithm 1.

Algorithm 1: general layout of population based algorithms [23]

for $h = 1{:}\, n_{\text{pop}}$ // Initial population
    Choose an initial solution for X_h from S
end for
Randomly choose one of the initial solutions as the best solution X_{Rovt}
$iter = 1$ //*Set* the current number of iterations
for $h = 1{:}\, n_{pop}$// Find the best solution in current population
    if $F(X_h) < F(X_{Best})$
        $X_{Best} = X_h$
    end if
end for
Repeat // Iterative process
    for $h = 1{:}\, n_{pop}$ // Generation/Replacement processes
        Generate a new solution $X_{h\text{Vew}}$
        Apply a replacement strategy
        if replacement is necessary
            $X_h = X_{hNew}$
        end if
    end for
    for $h = 1{:}\, n_{pop}$//Find the best solution in current population
        if $F(X_h) < F(X_{Best})$
            $X_{Best} = X_h$
        end if
end for
$iter++$// Set the current number of iterations
until (stopping criteria satisfied)

At the end of the process, the optimization solution will be stored in X_{Best} . This is a general layout of population-based algorithms; it cannot accurately describe all existing population based algorithms as each algorithm has its own peculiarities.[23]

II.5. SWARM INTELLIGENCE:

Algorithms inspired by the collective behaviors of species such as ants, bees, wasps, termites, fish, and birds are termed as swarm intelligence algorithms. This field emerged from the observation of social dynamics among these species as they compete for resources. Swarm-intelligence-based algorithms exhibit several defining characteristics: the simplicity of their individual agents, their cooperation through indirect communication channels, and their navigation within decision spaces. Notable examples of successful optimization algorithms inspired by swarm intelligence include ant colony optimization and particle swarm optimization. These algorithms emulate the collaborative and decentralized nature of natural swarms, leveraging it for solving complex optimization problems efficiently.

II.5.1. Particle Swarm Optimization: (PSO)

Particle Swarm Optimization (PSO) is a stochastic, population-based metaheuristic inspired by swarm intelligence. It mimics the social behavior of natural organisms, such as bird flocking and fish schooling, to locate areas with abundant resources. In these swarms, coordinated behavior emerges from local interactions without any central control. Initially, PSO was successfully designed for continuous optimization problems, with its first application to optimization problems proposed in the literature.

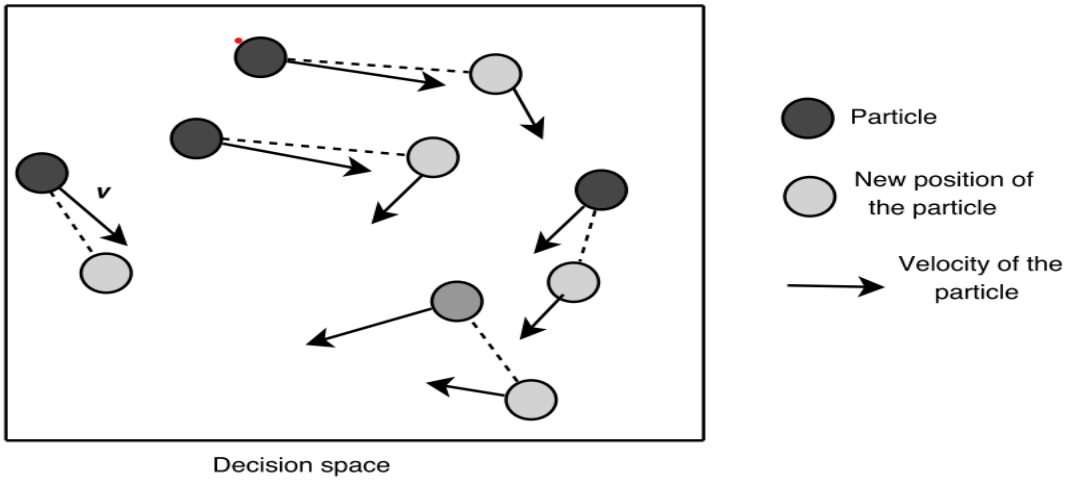

Figure 9:Particle swarm with their associated positions and velocities. At each iteration, a particle moves from one position to another in the decision space. PSO uses no gradient information during the search. [6]

In the basic PSO model, a swarm consists of N particles navigating a D-dimensional search space. Each particle i represents a candidate solution to the problem and is denoted by the vector xi in the decision space. A particle has its own position and velocity, indicating its direction and step size. Optimization is achieved through cooperation among the particles, where the success of some particles influences the behavior of others) Figure 9. Each particle adjusts its position xi iteratively towards the global optimum based on two factors: the best position it has visited (pbesti), denoted as $p_i = (p_{i1}, p_{i2}, \dots, p_{iD})$, and the best position visited by the entire swarm (gbest), or lbest, which is the best position within a given subset of the swarm. This is represented as $p_g = (p_{g1}, p_{g2}, \dots, p_{gD})$. The vector $(p_g - x_i)$ signifies the difference between the current position of particle i and the best position of its neighborhood.

Algorithm 2: Template of the particle swarm optimization algorithm. [6]

```
Random initialization of the whole swarm ;
Repeat
  Evaluate f(x_i) ;
  For all particles i
    Update velocities:
      v_i(t) = v_i(t − 1) + ρ_1 × (p_i − x_i(t − 1)) + ρ_2 × (p_g − x_i(t − 1)) ;
    Move to the new position: x_i(t) = x_i(t − 1) + v_i(t) ;
    If  f(x_i) < f(pbest_i) Then pbest_i = x_i ;
    If  f(x_i) < f(gbest) Then gbest = x_i ;
    Update(x_i, v_i) ;
  EndFor
Until Stopping criteria
```

II.5.2. Artificial Bee Colony: (ABC)

The bee colony optimization-based algorithm is a stochastic P-metaheuristic belonging to the class of swarm intelligence algorithms. Over the past decade, numerous studies have leveraged various bee colony behaviors to tackle complex combinatorial or continuous optimization problems. These algorithms draw inspiration from the behavior of honeybee colonies, which exhibit numerous features that can be modeled for intelligent and collective behavior.[6]

The Artificial Bee Colony (ABC) is a meta-heuristic optimization algorithm introduced by Karaboga in 2005 to address optimization problems in multivariable functions. This algorithm is based on observations of the social behavior of honeybee swarms. An artificial bee colony consists of three groups of bees:

1. Employed Bees: Each employed bee is responsible for exploiting a food source and communicates its position to onlooker bees by dancing near the hive. There is one employed bee per food source.

2. Onlooker Bees: Onlooker bees select the best food sources to exploit further based on the information provided by the employed bees. Therefore, better food sources attract more onlooker bees compared to poorer ones.
3. Scout Bees: When a food source is depleted, the employed bee associated with it becomes a scout and randomly searches for a new food source to replace the old one.[6]

II.5.2.1. Control Parameters :

The ABC algorithm has three control parameters:

1. Bee colony size (equal to twice the number of food sources).
2. Local search abandonment limit.
3. Maximum number of search cycles or a fitness-based termination criterion.

II.5.2.2. Algorithm Flowchart:

The Artificial Bee Colony (ABC) algorithm is a nature-inspired optimization technique that mimics the foraging behavior of honey bees to address complex optimization problems. This method employs three types of bees: employed bees, onlooker bees, and scout bees. These bees work together to explore and exploit potential solutions. In the ABC algorithm, each employed bee is linked to a specific food source, which represents a potential solution. Unlike real bee colonies, there is a direct correspondence between employed bees and food sources, ensuring that the number of food sources matches the number of employed bees. The upcoming sections will detail the phases of the algorithm, including the Initialization Phase and Iterative Phases.[21]

A. Initialization Phase:

During the initialization phase of the algorithm, a group of potential solutions, referred to as food sources, is created by scout bees. These scout bees randomly generate the initial set of food sources. Each food source represents a possible solution to the problem at hand. Alongside this, various control parameters required for the algorithm are also set up.

A scout bee plays a crucial role in this process by generating a food source randomly. Once a food source is generated, the scout bee is assigned to it, becoming an employed bee. This means the bee will start working on exploring and improving this particular solution. This initial setup ensures that the algorithm has a diverse set of starting points from which it can begin to search for optimal solutions.

B. Iterative Phases:

The ABC algorithm iterates through the following three phases:

1. Employed Bees Phase:

Employed bees search for new food sources with more nectar within the neighborhood of their current food source x_m. They find a neighboring food source and evaluate its fitness. A greedy selection is made between the two sources, performing a local search step. Employed bees share information about their food sources with onlooker bees by dancing in the dancing area.

2. Onlooker Bees Phase:

Employed bees share nectar information about their corresponding food sources with onlooker bees. Onlooker bees select a food source i with a probability Pi determined by roulette-wheel selection:

$$P_i = \frac{f_i}{\sum_{j=1}^{M} f_j} \qquad \text{Equation II–3}$$

where f_i is the fitness of the solution corresponding to food source i, and M is the total number of food sources (equal to the number of employed bees).

The fitness f_i of a solution is defined from its objective function $f(x_m)$ by:

$$f_i = \begin{cases} 1/(1+f(x_i)) & \text{if } f(x_i) \geq 0 \\ 1+|f(x_i)| & \text{if } f(x_i) < 0 \end{cases} \qquad \text{Equation II–4}$$

After a food source is chosen for an onlooker bee, a neighboring source is determined, and a greedy selection is made between the two sources. This phase ends when the new locations of all food sources are determined.

3. Scout Bees Phase:

In the scout bees phase of the algorithm, employed bees that fail to improve their solutions after a certain number of attempts are converted into scout bees. When a solution cannot be enhanced despite multiple trials, it is considered exhausted, and the corresponding food source is abandoned. These employed bees then transition into scouts.

As scouts, these bees begin searching for new, random solutions (denoted as x_i). Upon finding a new food source, each scout bee becomes an employed bee again, working on the newly discovered solution. If the nectar (quality) of this new food source is equal to or better than that of the old one, it replaces the old source in the algorithm's memory.

This process ensures that poor or depleted food sources are abandoned, making room for potentially better solutions. These three steps include converting unsuccessful employed bees to scouts, searching for new solutions, and updating the memory with better food sources. This cycle

is repeated until the algorithm meets a predefined termination criterion, such as a maximum number of iterations or a satisfactory solution quality.

Algorithm 3: (Artificial Bee Colony).[21]

1. Initialize the parameters.
2. Generate randomly distributed food sources $\boldsymbol{x}_i, i = 1, \ldots, M$, over the search space, and evaluate their nectar (fitness).
3. Send the employed bees to the current food sources.
4. **Repeat**:
   a. **for each** employed bee:
      Find a new food source in its neighborhood, and evaluate the fitness.
      Apply greedy selection on the two food sources.
   end for
   b. Calculate the probability P_i for each food source.
   c. **for each** onlooker bee:
      for each food source i:
      if $(rand() < P_i)$
      Send the onlook bee to the food source of the ith employed bee.
      Find a new food source in the neighborhood, and evaluate the fitness.
      Apply greedy selection on the two food sources.
      end if
      continue
      end for
   end for
   d. **if** any employed bee becomes sout bee
      Send the scout bee to a randomly produced food source.
   end if
   e. Memorize the best food source (solution) found so far.
   until termination criterion is satisfied.

II.5.3. Ant Colony Optimization : (ACO)

The fundamental concept behind ant colony optimization (ACO) algorithms is to emulate the cooperative behavior of real ants to tackle optimization problems. ACO metaheuristics were introduced by M. Dorigo. These algorithms function as multiagent systems, with each agent drawing inspiration from the behavior of real ants. Traditionally, ACO has been applied to combinatorial optimization problems, achieving significant success in various applications such as scheduling, routing, and assignment. The primary interest in the behavior of real ants lies in their ability to perform complex tasks, like transporting food and finding the shortest paths to food sources, through collective behavior. ACO algorithms replicate the principle that an ant colony can identify the shortest path between two points using very simple communication mechanisms.

Figure 10 depicts an experiment conducted by Goss et al. with a real colony of Argentine ants (Iridomyrmex humilis). These ants have poor vision. The colony has access to a food source via two paths connected to the nest. As ants travel, they leave a chemical trail (pheromone) on the ground, which is an olfactory and volatile substance. This pheromone trail guides other ants toward the target point. The greater the amount of pheromone on a specific path, the higher the likelihood

that ants will choose that path. For any given ant, the path is selected based on the quantity of pheromone detected. [6]

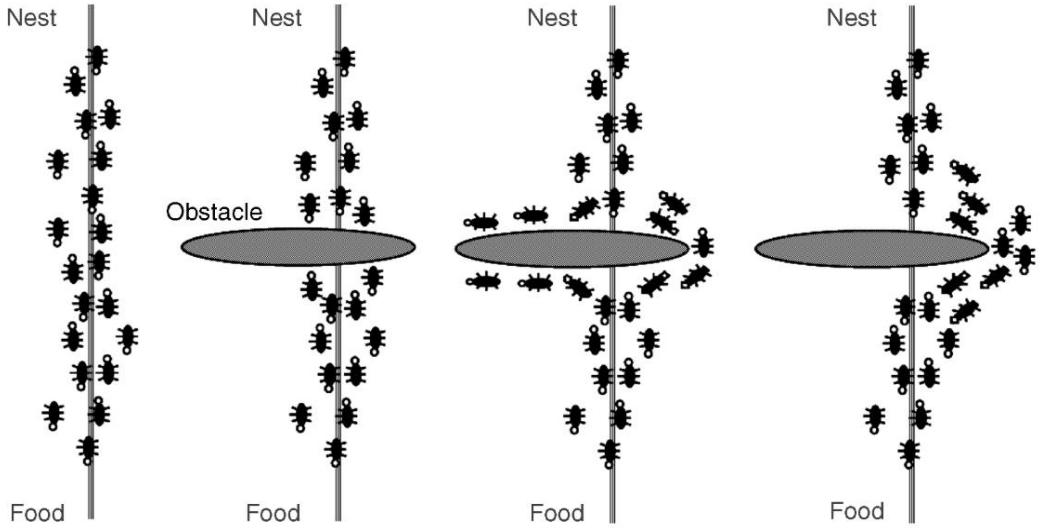

Figure 10.Inspiration from an ant colony searching an optimal path between the food and the nest.[6]

Furthermore, this chemical substance gradually diminishes over time due to evaporation, and the amount left by an ant depends on the quantity of food (reinforcement process). As illustrated in Figure, when encountering an obstacle, each ant has an equal probability of choosing the left or right path. Since the left path is shorter than the right, requiring less travel time, the ant will leave a higher concentration of pheromone on this path. The more ants that take the left path, the stronger the pheromone trail becomes. Consequently, the shortest path emerges over time. This effect is further enhanced by the evaporation process. This indirect form of cooperation among ants is known as stigmergy. (Algorithm 5) outlines the basic structure of the Ant Colony Optimization (ACO) algorithm. Initially, the pheromone information is set up. The algorithm then primarily consists of two repeated steps: solution construction and pheromone update.

Algorithm 4: Template of the ACO.[6]

```
Initialize the pheromone trails;
Repeat
    For each ant Do
        Solution construction using the pheromone trail;
        Update the pheromone trails:
        Evaporation;
        Reinforcement;
Until Stopping criteria
```

Output: Best solution found or a set of solutions.

II.6. Evolutionary Algorithms (EA):

In the nineteenth century, J. Mendel laid the groundwork for the principles of heredity from parents to offspring. Subsequently, in 1859, C. Darwin introduced the theory of evolution in his seminal work, "On the Origin of Species." These theories of species creation and evolution served as inspiration for computer scientists in the 1980s, leading to the development of evolutionary algorithms. [6]

Over the past 40 years, different main schools of evolutionary algorithms have emerged independently. Genetic algorithms (GA) were primarily developed in Michigan, USA, by J. H. Holland. Evolution strategies originated in Berlin, Germany, by I. Rechenberg and H-P. Schwefel. Meanwhile, evolutionary programming was pioneered by L. Fogel in San Diego, USA. Later, in the late 1980s, genetic programming was proposed by J. Koza. Although each of these approaches constitutes a different methodology, they are all inspired by the same principles of natural evolution. A comprehensive introductory survey can be found in the referenced literature.

Evolutionary algorithms (EA) are stochastic population-based metaheuristics that have been successfully applied to numerous real and complex problems, including epistatic, multimodal, multiobjective, and highly constrained problems. They represent the most extensively studied population-based algorithms. The success of EA in solving challenging optimization problems across various domains has led to the establishment of the field known as evolutionary computation (EC). EAs are based on the concept of competition and simulate the evolution of species, as depicted in Figure 13. [6]

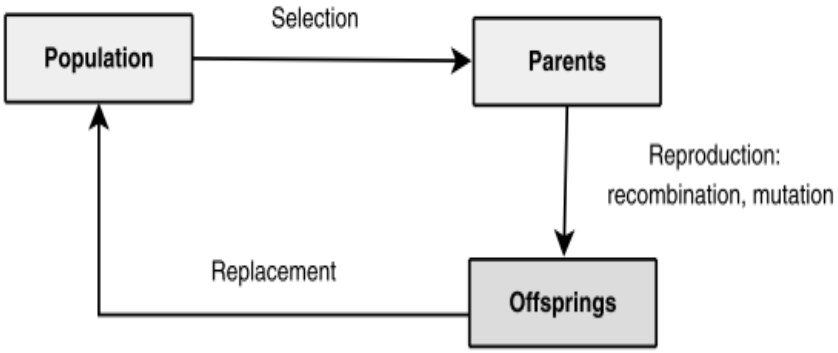

Figure 11.A generation in evolutionary algorithms. [6]

These algorithms operate on a population of individuals, where each individual represents a tentative solution encoded in a specific manner. An objective function evaluates the fitness of each individual, indicating its suitability for the problem at hand. At each iteration, individuals are selected as parents based on their fitness, and reproduction operators (e.g., crossover, mutation) are applied to generate offspring. A replacement scheme is then employed to determine which individuals survive from the offspring and parents. This iterative process continues until a stopping criterion is met. Algorithm 6. illustrates the general template of an evolutionary algorithm.

Algorithm 5: Template of an evolutionary algorithm. [6]

```
Generate(P(0)) ; /* Initial population */
t = 0 ;
While not Termination_Criterion(P(t)) Do
   Evaluate(P(t)) ;
   P'(t)      = Selection(P(t)) ;
   P'(t)      = Reproduction(P'(t)); Evaluate(P'(t)) ;
   P(t + 1)   = Replace(P(t), P'(t)) ;
   t = t + 1 ;
End While
Output Best individual or best population found.
```

II.6.1. Evolution Strategies: (ES)

Evolution strategies (ES) represent another subclass of evolutionary algorithms (EA), alongside genetic algorithms (GA) or genetic programming (GP). They were initially pioneered by Rechenberg and Schewefel in 1964 at the Technical University of Berlin. ESs are primarily utilized for continuous optimization tasks where representations are based on real-valued vectors. Early applications include real-valued parameter shape optimization. Typically, ES employ an elitist replacement strategy and a specific normally (Gaussian) distributed mutation, with crossover being seldom used. In an ES, a clear distinction exists between the population of parents, denoted by μ, and the population of offspring, denoted by $\lambda \geq \mu$. An individual in an ES is comprised of floating-point decision variables along with other parameters guiding the search. This setup enables ESs to facilitate a kind of self-adaptation by evolving both the solution and the strategy parameters (e.g., mutation step size) simultaneously. The selection operation within an ES is deterministic and relies on fitness ranking, allowing for highly customizable parameterization. Recombination in ES can be either discrete (similar to uniform crossover in GA) or intermediary (such as arithmetic crossover). One of the main advantages of ESs lies in their efficiency in terms of time complexity. Furthermore, there exists a more comprehensive theoretical understanding of convergence for evolution strategies compared to other evolutionary algorithms

Algorithm 6: illustrates the evolution strategy template. [6].

Initialize a population of μ individuals ;
Evaluate the μ individuals ;
Repeat
  Generate λ offsprings from μ parents ;
  Evaluate the λ offsprings ;
  Replace the population with μ individuals from parents and offsprings ;
Until Stopping criteria
Output Best individual or population found.

II.6.2. Genetic Algorithm: (GA)

The genetic algorithm is renowned as one of the most impactful and successful metaheuristic optimization methods in the research community. Initially conceived by Holland and his colleagues during the 1960s and 1970s, this algorithm has undergone extensive scrutiny, exploitation, and refinement since its inception. Rooted in the principles of biological evolution and natural selection, it serves as a model for simulating evolutionary processes. Beginning with an initial population, the algorithm iteratively generates increasingly fit populations (or generations) of chromosomes using genetic operators such as crossover, recombination, and mutation. The main steps of the algorithm are outlined in the flowchart depicted in Algorithm 9.

Each chromosome, denoted as $x_h = \{x_{h1}, \dots, x_{hD}\}(h = 1, \dots, n_{pop})$ represents a D-dimensional vector of variables, offering a potential solution to the optimization problem, alongside its associated fitness value, $F_{itness}(x_h)$. After establishing the initial population and evaluating the fitness of its chromosomes, the algorithm proceeds to iterate through the following steps:

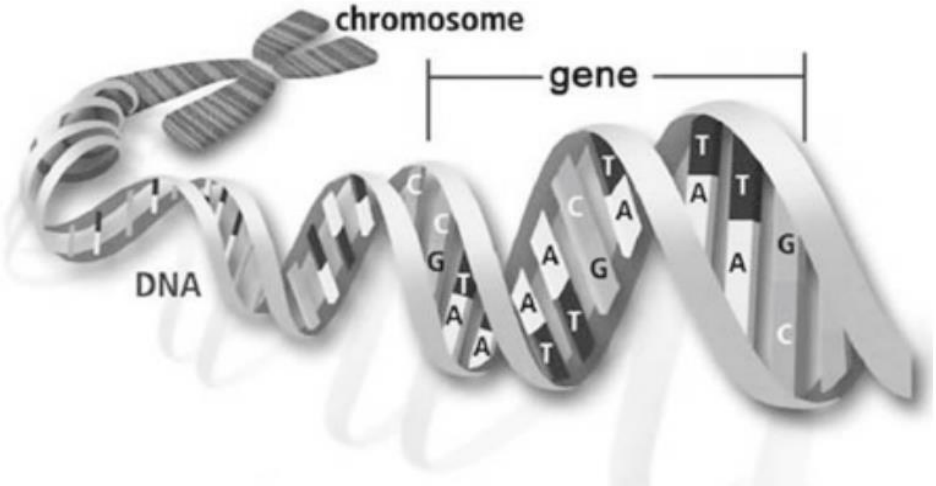

Figure 12.Ageneona chromosome (Courtesy U.S. Department of Energy, Human Genome Program).[21]

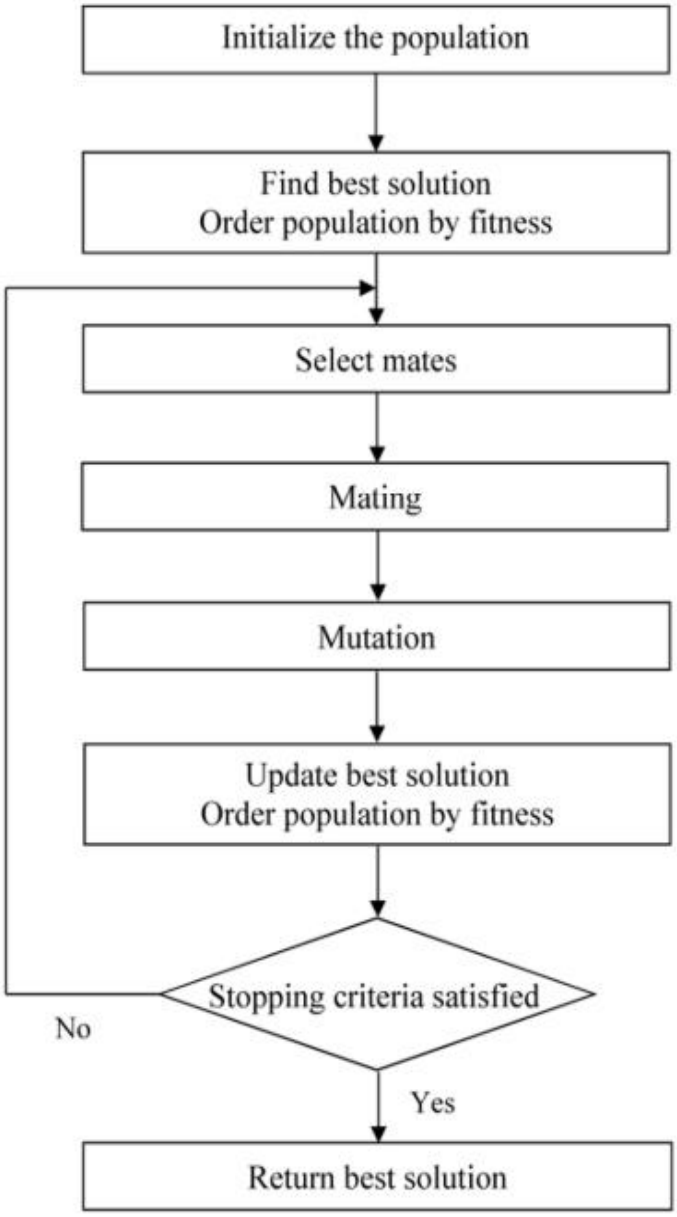

Figure 13: Flowchart of the standard GA. [23]

II.6.2.1. Natural Selection:

Based on the fitness information obtained in the preceding phase, chromosomes undergo sorting in a descending order according to their fitness. Only the most fit chromosomes are chosen to survive and potentially produce offspring for the subsequent generation, while the fewer fit ones are eliminated. From the population of n_{pop} chromosomes, only the top N individuals are retained for mating, while the rest are discarded to accommodate new offspring. This continual process enables the population to evolve across generations. [23]

II.6.2.2. Pairing

In this stage, two parents are selected from the surviving population to generate two offspring, each inheriting traits from both parents. Various methods for pairing exist, including random selection of parents, roulette wheel selection based on fitness, or pairing from top to bottom according to the ordered chromosomes in the population. The selected parents are then added to the new population. This cycle repeats until the new population is fully regenerated. [23]

II.6.2.3. Mating

The offspring will be generated by merging the parents to pass on genetic material. The simplest method consists of choosing randomly a single or multiple crossover points in the chromosome. The first offspring will be built by copying the first parent until the crossover point, after which the second parent will be used. This procedure is inversed for the second offspring.

Let $x_f = \{x_{f1}, \dots, x_{JD}\}$ and $x_m = \{x_{m1}, \dots, x_{mD}\}$ be the parent, then the offspring

x_1 and x_2 are given:

$$x_1 = \{x_{f1}, x_{f2}, x_{f3}, \uparrow x_{m4}, x_{m5}, \dots, x_{mD}\} \quad \text{Equation II–5}$$

$$x_2 = \{x_{m1}, x_{m2}, x_{m3}, \uparrow x_{f4}, x_{f5}, \dots, x_{fD}\} \quad \text{Equation II–6}$$

This approach of generating offspring is not attractive since no new genetic material is introduced once an initial population has been chosen. We are merely interchanging variables between chromosomes; no new variables will be added to the chromosomes in this stage. Another more interesting method is the 'blending methods' in which the offspring are built by combining variables values of the parents as follows:

$$x_{1i} = \beta x_{fi} + (1-\beta)x_{mi}, \quad i \in \{1, \dots, D\} \quad \text{Equation II–7}$$

$$x_{2i} = \beta x_{mi} + (1-\beta)x_{fi}, \quad i \in \{1, \dots, D\} \quad \text{Equation II–8}$$

where β is a random number in [0,1]

This blending could be done to all variables or only to a limited number. [23]

II.6.2.4. Mutation

To allow the algorithm to explore other regions of the search space and escape local optima, a change or a mutation in some of the variables is randomly introduced. A parameter called mutation rate is used to determine the probability of a variable being mutated. For example, a mutation rate of 20% indicates that 1/5 of the variables in all of the chromosomes will be replaced by randomly generated values. The variables to mutate are also chosen randomly.

The algorithm will continue iterating by repeating the previous four phases until the stopping criterion has been satisfied. Originally, GA algorithm used a binary representation as chromosomes were represented by binary strings of 0 and 1. However, this discrete representation worked well only for problems requiring solutions of low dimensionality and precision. To overcome this limitation, the concept of real coded GA was introduced where a vector of real-valued genes was used to represent a chromosome. The remaining phases of the algorithm are the same as in the binary representation.

Genetic algorithm is one of the most widely used optimization algorithm in modern nonlinear optimization, nonetheless, it has several known deficiencies. Namely, its tendency to converge toward local optimum if the fitness function has not been correctly formulated, its slow convergence rate and the huge computing requirement needed to find a solution. In fact, given the same problem and computation time, simpler optimization algorithms may find better solutions. In order to overcome these issues, the balance between exploration and exploitation must be enhanced. Within the GA, the crossover operation affects decisively the exploration capability of the algorithm. As such, a lot of research has been conducted on how to produce more efficient crossover operators.[23]

II.6.3. Coevolutionary Algorithms:

Coevolution is the concept of intertwined evolutionary changes among closely associated species. This phenomenon is often observed in nature, where species evolve in response to each other's adaptations. For instance, many flowering plants and their pollinating insects have evolved in tandem, each adapting to the other's characteristics for successful reproduction. Similarly, predator-prey relationships demonstrate coevolution, where advancements in predator traits lead to corresponding evolutionary responses in prey species. These interactions drive the development of complex adaptations within populations.

In traditional evolutionary algorithms (EAs), populations typically consist of a single species. However, coevolutionary algorithms take a different approach, employing a cooperative-competitive strategy involving multiple populations, each representing a distinct species. In this setup, populations interact with each other while optimizing interconnected objectives. Competitive coevolutionary models involve populations competing to solve global problems, such as accessing limited resources, while cooperative coevolutionary models focus on populations collaborating to overcome challenges, such as acquiring scarce resources.

In competitive coevolutionary algorithms, individual fitness is influenced by competition with individuals from other populations. This competition drives populations to improve their capabilities until reaching an equilibrium where local objectives cannot be further enhanced,

hopefully resulting in the achievement of global objectives. Notably, the global solution is not directly assessed but emerges from the competition between populations.

A classic example of competitive coevolutionary algorithms is the predator-prey model, inspired by animals' behavior in flocks. In this model, the success of one population, such as predators, necessitates responses from other populations, like prey, leading to inverse fitness interactions. This dynamic interplay of inverse objectives serves as the primary driver for the evolution of diverse populations.

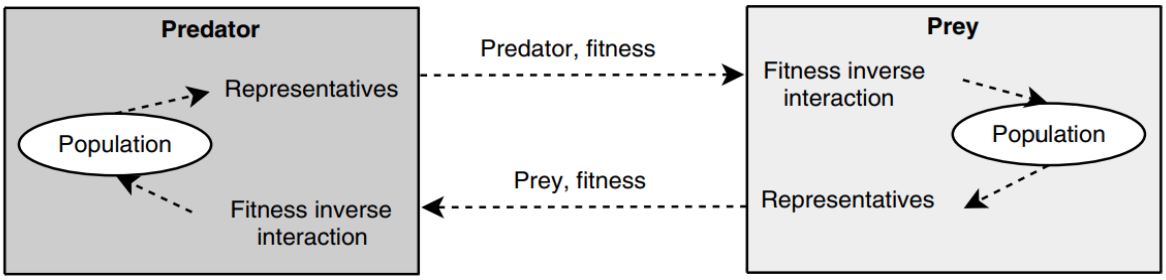

Figure 14: Competitive coevolutionary algorithms based on the predator–prey model.

II.6.4. Cultural Algorithms: (CA)

Cultural algorithms (CA) are specialized variants of evolutionary algorithms introduced by R. G. Reynolds in 1994. These algorithms are computational models of cultural evolution based on the principles of human social evolution. They employ a model of cultural change within optimization problems, where culture is symbolically represented and transmitted between successive populations. The main principle behind CAs is to preserve beliefs that are socially accepted while discarding unacceptable ones.

Cultural algorithms consist of two main elements: a population space at the microevolutionary level and a belief space at the macroevolutionary level (Figure 14). These elements interact through a vote–inherit–promote (VIP) protocol. This protocol enables individuals to alter the belief space and allows the belief space to influence the evolution of individuals. The population space at the microevolutionary level is typically managed by evolutionary algorithms (EA). In each generation, the knowledge acquired from the population’s search, such as the best solutions, can be stored in the belief space in various forms. These forms include logic- and rule-based models, schemata, graphical models, semantic networks, and version spaces, among others, to model the macroevolutionary process of a cultural algorithm.

The belief space is divided into distinct categories representing different domains of knowledge acquired by the population during the search process. These categories include normative knowledge, which is a collection of desirable value ranges for some decision variables of the individuals in the population; domain-specific knowledge, which is information about the domain

of the problem to which the CA is applied; situational knowledge; temporal knowledge, which is information about important events during the search; and spatial knowledge, which is information about the landscape of the tackled optimization problem.

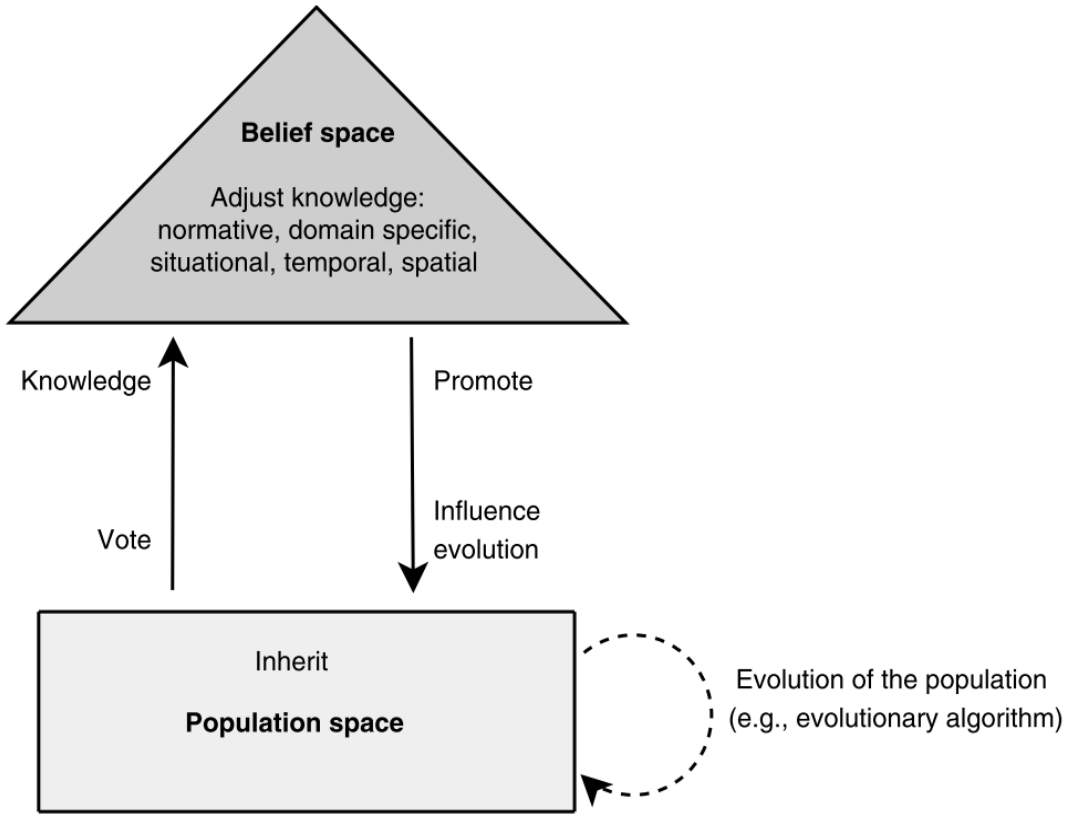

Figure 15.Search components of cultural algorithms. [6]

Algorithm 7: Template of the cultural algorithm. [6]

Initialize the population $Pop(0)$;
Initialize the Belief $BLF(0)$;
$t = 0$;
Repeat
  Evaluate population $Pop(t)$;
  Adjust(BLF(t), Accept(POP(t))) ;
  Evolve(Pop(t+1), Influence(BLF(t))) ;
  $t = t + 1$;
Until Stopping criteria
Output: Best found solution or set of solutions.

Thus, cultural algorithms represent a P-metaheuristic based on hybrid evolutionary systems that integrate evolutionary search and symbolic reasoning. They are particularly useful for problems that require extensive domain knowledge, such as constrained optimization problems, and for dynamic environments, such as dynamic optimization problems. [6]

II.6.5. Differential Evolution: (DE)

The Differential Evolution (DE) is among the most effective methods for continuous optimization. This assertion is supported by a comparison study conducted by our colleagues, which demonstrates that DE outperforms other algorithms (GA, PSO, and ABC). According to their findings, DE consistently achieves the best solutions across a wider range of functions and exhibits significantly superior convergence, especially as the dimensionality of the problem increases. [6] Originating from K. Price's efforts to tackle the Chebycheff polynomial fitting problem proposed by R. Storn, DE utilizes vector differences to perturb the vector population. This innovative approach includes a recombination operator that combines two or more solutions, alongside a self-referential mutation operator that directs the search towards optimal solutions. Further details on this topic will be elaborated in the next chapter of this thesis.[6]

II.7. Conclusion:

In this chapter, we have familiarized ourselves with the fundamental concepts of metaheuristic optimization and the details of several algorithms. As we move forward, our focus will shift to a more in-depth exploration of Differential Evolution (DE). This algorithm, which has shown remarkable success in continuous optimization problems, will be the subject of our detailed study and analysis.

We will delve into the intricacies of DE, examining its underlying principles, the unique mechanisms it employs for perturbing vector populations, and the innovative recombination and mutation operators that set it apart from other optimization techniques. By understanding these core components, we can appreciate how DE effectively directs the search towards optimal solutions.

Chapter III

Differential Evolution

Chapitre III. **Differential Evolution**

III.1. Introduction:

Differential Evolution (DE) is a powerful and versatile evolutionary algorithm renowned for its simplicity and effectiveness in solving complex optimization problems. Since its introduction by Rainer Storn and Kenneth Price in the mid-1990s, DE has been widely adopted due to its robust performance across various domains.

DE operates by iteratively improving a population of candidate solutions through mechanisms inspired by natural selection and genetic variation. This enables DE to navigate complex, multimodal landscapes and high-dimensional spaces effectively, making it suitable for a wide range of optimization challenges.

This chapter delves into the fundamental components and mechanics of DE, including its control parameters, initialization methods, mutation strategies, crossover operations, and selection mechanisms. Additionally, it explores various DE variants, each designed to enhance the algorithm's performance for different types of optimization problems.

By understanding these core elements and their interactions, one can leverage DE to address real-world challenges, advancing optimization methodologies in fields such as logistics and energy management.

III.2. Notation:

The standard notation for differential evolution (DE) algorithms is DE/x/y/z, where each element provides specific information about the variant of the DE algorithm being used. Here's a detailed explanation of each component: [26]

DE: This stands for Differential Evolution, indicating the type of evolutionary algorithm.

x: This specifies the method used to select the vectors involved in the mutation process. The common options for x include rand, best, current-to-best, rand -to- best.

y: This denotes the number of difference vectors used in the mutation process. A difference vector is the difference between two vectors in the population, and multiple difference vectors can be combined. The value of y is usually an integer such as 1, 2, or 3.

z: This represents the crossover scheme employed to create the trial vector. Common crossover schemes include Bin and exp.

In the context of the Differential Evolution (DE) algorithm, several key concepts and notations are essential for understanding the optimization process. The target vector, often denoted as x_i represents the current solution within the population and is sometimes referred to as the parent vector. The mutant vector, denoted as v_i is created by combining elements from different target vectors according to specific mutation strategies. This vector serves as a potential candidate for generating new solutions. The trial vector, denoted as c_i, is formed by recombining elements from the target vector and the mutant vector through a crossover operation. This vector is also known as the child vector. The goal of the DE algorithm is to iteratively improve these vectors, selecting the best solutions based on their fitness.

the primary notations are summarized in Table 1. These notations will be consistently applied throughout this chapiter unless specified otherwise.

Table 1: Primary Notations [26]

notation	Legend	notation	legend
f(x)	objective function to minimize	X	N-dimensional vector of optimization parameters
x_j	the *j*th optimization parameter	p_c	crossover probability
P	Population	$\mathbf{P}^n$	population of generation *n*
$\mathbf{x}^i$	vector of optimization parameters of $\mathbf{p}^i$	$\mathbf{x}^{n,i}$	vector of optimization parameters of $\mathbf{P}^{n,i}$
x_j^i	the jth optimization parameter in $\mathbf{x}^i$of $\mathbf{p}^i$	$x_j^{n,i}$	the *j*th optimization parameter in $\mathbf{x}^{n,i}$ of $\mathbf{P}^{n,i}$
$\mathbf{v}^i$	mutant for $\mathbf{p}^i$	$\mathbf{v}^{n+1,i}$	mutant for $\mathbf{P}^{n,i}$
$\mathbf{x}^{\mathrm{v},i}$	vector of optimization parameters of $\mathbf{v}^i$	$\mathbf{x}^{n+1,\mathrm{v},i}$	vector of optimization parameters of $\mathbf{v}^{n+1,i}$
$x_j^{\mathrm{v},i}$	the jth optimization parameter in $\mathbf{x}^{\mathrm{v},i}$of $\mathbf{v}^i$	$x_j{}^{n+1,\mathrm{v},i}$	the *j*th optimization parameter in $\mathbf{x}^{n+1,\mathrm{v},i}$ of $\mathbf{v}^{n+1,i}$
$\mathbf{x}^{b,i}$	vector of optimization parameters of b^i	$\mathbf{x}^{n,\mathrm{b},i}$	vector of optimization parameters of $\mathbf{P}^{n,i}$
$x_j^{b,i}$	the jth optimization parameter in $\mathbf{x}^{b,i}$ of b^i	$x_j^{n,\mathrm{b},i}$	the *j*th optimization parameter in$\mathbf{x}^{n,\mathrm{b},i}$ of $\mathbf{P}^{n,i}$
c^i	the ith child	$\mathbf{c}^{n+1,i}$	the *i*th child of generation *n* + 1
$\mathbf{x}^{c,i}$	vector of optimization parameters of c^i	$\mathbf{x}^{n+1,\mathrm{c},i}$	vector of optimization parameters of $\mathbf{c}^{n+1,i}$
$x_j^{c,i}$	the *j*th optimization parameter in $\mathbf{x}^{c,i}$ of c^i	$x_j{}^{n+1,\mathrm{c},i}$	the *j*th optimization parameter in $\mathbf{x}^{n+1,c,i}$ of $\mathbf{c}^{n+1,i}$

III.3. Basics and Components of DE:

This section explores the fundamental elements that constitute the Differential Evolution (DE) algorithm. Understanding these components is essential for effectively applying DE to various optimization problems. We will cover the primary aspects of DE, including its control parameters, initialization process, mutation, crossover, and selection mechanisms. Each of these components plays a crucial role in guiding the evolutionary search process and ensuring the robustness and efficiency of the algorithm.

In particular, we will focus on the DE/rand/1/bin scheme as an illustrative example in this section, providing detailed insights into its workings. Later, in subsequent sections, we will explore additional variants of DE, broadening our understanding of its versatility and applicability in optimization contexts.

III.3.1. Setting Control Parameters:

To achieve optimal performance, we must fine-tune several optimization parameters, collectively referred to as control parameters. Although there are only three primary control parameters in the algorithm, they are crucial:

Population Size (NP): This parameter defines the number of candidate solutions in each generation. A larger population size increases the exploration capability of the algorithm but also increases computational costs.

Scaling Factor (F): This factor controls the amplification of the differential variation and typically ranges between 0 and 2. It impacts the mutation process and helps balance exploration and exploitation.

Crossover Rate (CR): The crossover rate determines the probability of crossover occurring between the parent and mutant vectors. A higher CR increases the diversity of the population but can slow down convergence.[27]

In addition to these, there are other parameters that influence the optimization process: (a) the problem dimension D, which affects the complexity of the optimization task, (b) the maximum number of generations (or iterations) GEN, which serves as a stopping criterion, and (c) the lower and upper boundary constraints L and H, which define the feasible search area. These parameters can be adjusted as needed. [28]

III.3.2. Population Initialization:

Before diving into the optimization process, it's crucial to establish a group of individuals, known as a population, and evaluate their effectiveness in solving the problem at hand. To begin, we

generate these individuals randomly, ensuring they fall within predefined limits or boundaries. These individuals represent potential solutions to our optimization problem.

Next, we assess the 'fitness' of each individual, 'Fitness' here refers to how well-suited an individual is to tackle the optimization task. This assessment involves applying a specific formula or function to each individual's characteristics or attributes. Essentially, we're determining how close each individual comes to an optimal solution based on our problem's criteria.

This preparatory step lays the foundation for the optimization process, helping us understand the initial landscape of potential solutions and guiding us toward finding the best possible outcome.[28]

Algorithm 8: a MATLAB implementation of the population initialization

```
% Initialize the population and their costs
pop_Solution = ones(nPop, n_dimension);
pop_Cost = ones(nPop, 1);
BestSol_Cost = Inf;

% Loop through the population to initialize each solution and evaluate its cost
for i = 1:nPop
    % Initialize each solution within the boundary constraints
    pop_Solution(i, :) = unifrnd(Search_Space_Min, Search_Space_Max, 1, n_dimension);

    % Evaluate the cost (fitness) of each solution
    pop_Cost(i, 1) = objective_function(pop_Solution(i, :), f);

    % Update the best solution found so far
    if pop_Cost(i, 1) < BestSol_Cost
        BestSol_Cost = pop_Cost(i, 1);
        BestSol = pop_Solution(i, :);
    end
end
```

In this code :

- (nPop) represents the number of individuals in the population.
- (n_dimension) is the number of dimensions for each solution.
- (Search_Space_Min) and (Search_Space_Max) define the boundary constraints for the initialization of the population.
- (objective_function) is the function used to evaluate the fitness of each solution.
- (BestSol) stores the best solution found.
- (BestSol_Cost) stores the corresponding cost of the best solution.

III.3.3. Mutation:

After initialization, Differential Evolution (DE) mutates and recombines the population to create a new set of trial vectors. The mutation operation involves adding a scaled difference vector to a third vector, as shown in Equation III–1:

$$v_{i,g} = x_{r0,g} + F \cdot (x_{r1,g} - x_{r2,g}) \qquad \text{Equation III–1}$$

Here, the scale factor F (a positive number between 0 and 1) controls how much the population evolves. Typically, F values are kept below 1.0 for effectiveness.

The indices r0, r1, and r2 denote randomly selected vectors for mutation. r0 is distinct from the target vector index *i*, while r1 and r2 are also randomly chosen but must be different from each other and from r0 and *i*.

Fig.16 illustrates how the mutant vector $v_{i,g}$ is created in a two-dimensional parameter space.

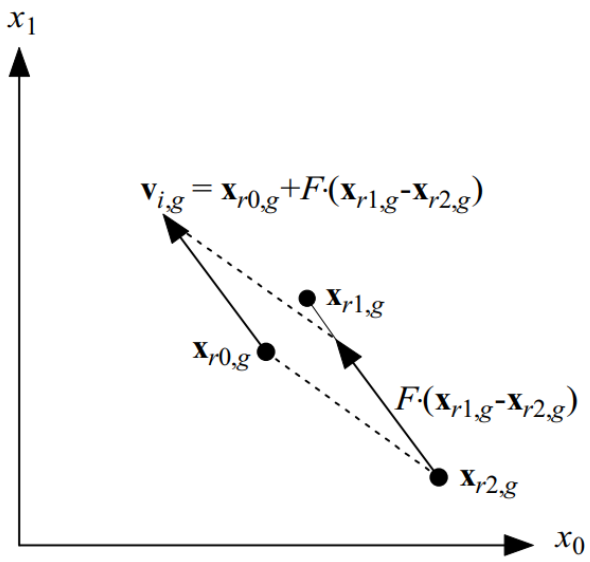

Figure 16:Differential mutation: the weighted differential, $F \cdot (x_{r1,g} - x_{r2,g})$, is added to the base vector $x_{r0,g}$, to produce a mutant, $v_{i,g}$. [29]

III.3.4. Crossover:

To complement the differential mutation search strategy, DE also employs binomial crossover which builds trial vectors out of parameter values that have been copied from two different vectors. In particular, DE crosses each target vector with a mutant vector:

$$c_{i,j,n} = \begin{cases} v_{i,j,n} & \beta_{i,j,n} \le C_r \\ x_{i,j,n} & \text{otherwise} \end{cases} \qquad \text{Equation III–2}$$

The crossover probability, Cr in [0,1], is a user-defined value that controls the fraction of parameter values that are copied from the mutant. To determine which source contributes a given parameter, binomial crossover compares Cr to the output of a uniform random number generator, rand (0,1) denoted as $\beta_{i,j,n}$. If the random number is less than or equal to Cr, the trial parameter is inherited from the mutant, $v_{i,g}= c_{i,g}$; otherwise, the parameter is copied from the target vector, $v_{i,g}= x_{i,g}$. Additionally, the trial parameter with a randomly chosen index, j rand, is taken from the mutant to ensure that the trial vector does not duplicate $x_{i,g}$. Due to this additional constraint, Cr only approximates the true probability p_c that a trial parameter will be inherited from the mutant.

It is important to verify that each parameter of the trial vector remains within the predefined boundary constraints. If any parameter falls outside these boundaries, it should be adjusted to fall within the acceptable range. This verification step ensures that all solutions are valid and feasible within the problem's constraints, maintaining the integrity and applicability of the optimization process. [30]

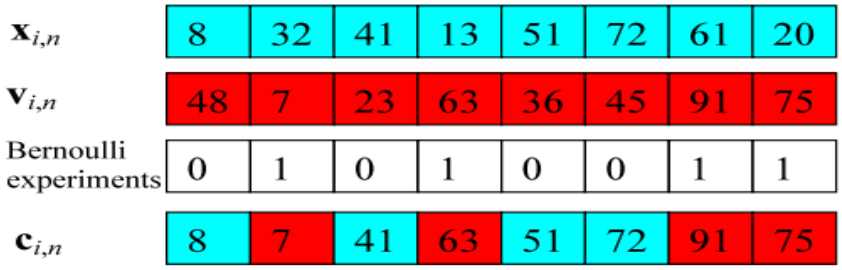

Figure 17: A Non-consecutive binomial crossover[30]

III.3.5. Selection:

If the trial vector, $c_{i,g}$, achieves an objective function value that is equal to or lower than its corresponding target vector, $x_{i,g}$, it takes the place of the target vector in the subsequent generation. Otherwise, if the trial vector's objective function value is higher than that of its target vector, the target vector remains in the population for at least one more generation (see Equation III–3). This mechanism ensures a gradual exploration of the search space, allowing promising solutions to persist while still making room for potentially better solutions in subsequent generations.

$$f(c_{i,g}) \leq f(x_{i,g}) \rightarrow x_{i,g+1} = c_{i,g} \quad \text{Equation III–3}$$
$$\text{otherwise, } x_{i,g+1} = x_{i,g}$$

In Equation III–4, the decision is made based on whether the trial vector's objective function value ($f(c_{i,g})$)) is less than or equal to the objective function value of its target vector ($f(x_{i,g})$). If it is,

the trial vector replaces the target vector; otherwise, the target vector remains unchanged in the population. [30]

Once this selection process is completed, the mutation, recombination, and selection steps are repeated until the optimization process either locates the optimal solution or satisfies a predefined termination criterion. This criterion could be reaching a maximum number of generations (g_{max}), indicating that further iterations may not yield significant improvements.[30]

The following is a MATLAB implementation of the DE algorithm main loop that comes after the population initializing phase which include the mutation, crossover, and selection phases:

Algorithm 9: A MATLAB implementation of the DE algorithm main loop

```
for iteration_index = 1:Max_number_of_Iterations
    for i = 1:nPop
        %% Mutation/Crossover %%
        A = randperm(nPop);
        A(A == i) = [];
        a = A(1);
        b = A(2);
        c = A(3);
        dimensionToupdate = randi([1 n_dimension]);
        y =pop_Solution(i,:);  % we set the trial vector equal to The target vector
        for indDimension = 1:n_dimension
            randomValur=rand();
% we change the dimension d of the target vector to the one of the mutent vector
% accoreding the difined probability which give us the trial vector.
            if (randomValur<cr || indDimension==dimensionToupdate)
                y(1,indDimension) = pop_Solution(a,indDimension)+F*(pop_Solution(b,indDimension)-pop_Solution(c,indDimension));
                y(1,indDimension) = max(y(1,indDimension), Search_Space_Min);
                y(1,indDimension) = min(y(1,indDimension), Search_Space_Max);
            end
        end
        %% Selction %%
        NewSol_Cost = objective_function(NewSol, function_index);
        if NewSol_Cost<=pop_Cost(i,:)
            pop_Cost(i,:) = NewSol_Cost;
            pop_Solution(i,:) = NewSol;
            if pop_Cost(i,:)<BestSol_Cost
                BestSol_Cost = pop_Cost(i,:);
                BestSol = pop_Solution(i,:);
            end
        end
    end  % the end of the population loop
end  %  the end of iterations
```

Figure 18 outlines the Differential Evolution (DE) algorithm process, starting with the generation of an initial population of potential solutions. Each solution's objective function is calculated to measure its performance. The algorithm identifies the best solution among the current population and applies mutation and crossover operations to generate new candidate solutions (offspring). The objective function is recalculated for each offspring and compared with their parent solutions. If an offspring has a better objective function value, it replaces the parent in the population, guiding the population towards better solutions. The algorithm checks if stopping criteria, such as the number of generations or a specific objective function value, are met. If not, the process repeats; if met, the algorithm stops, providing the best-found solution as the optimal result. [31]

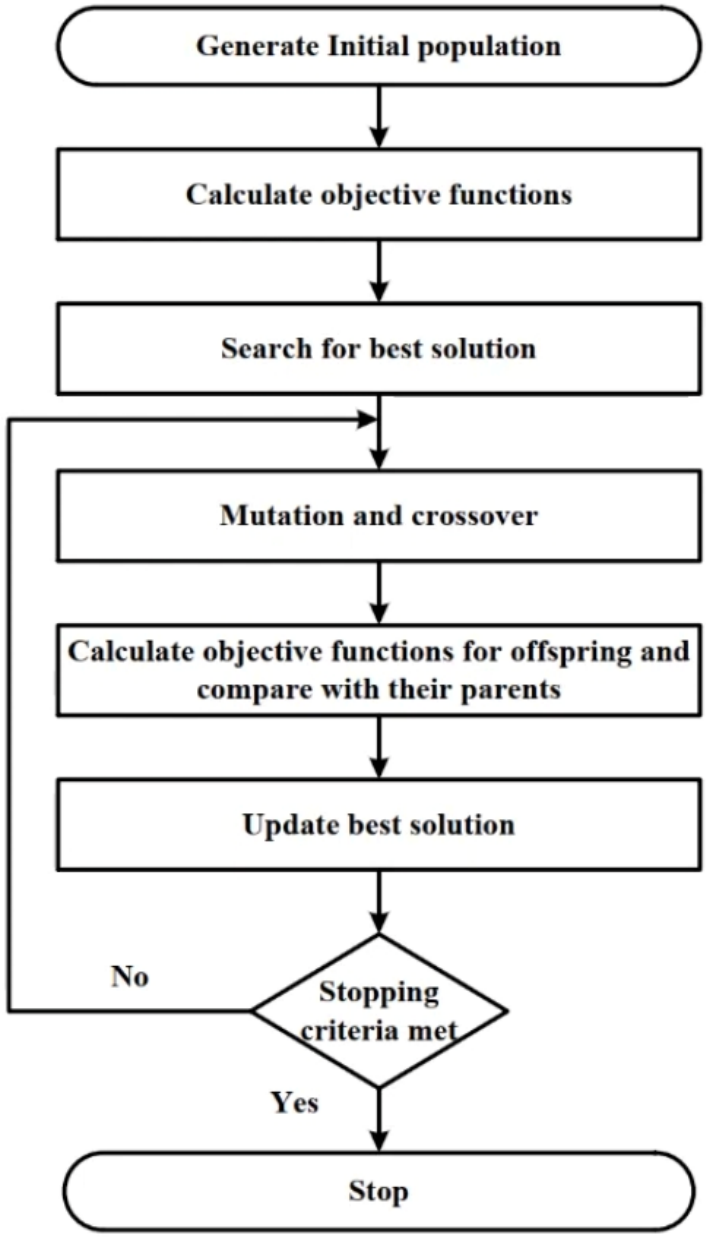

Figure 18: DE algorithm flowchart

III.4. DE Variants:

The performance of classic differential evolution (DE) can be inadequate for certain complex problems. To address this, researchers have developed various modifications to improve its effectiveness.

Classic differential evolution operates on a population consisting of N_p individuals and follows two main stages: initialization and evolution. During the initialization stage, the initial population is generated. The evolution stage involves iterative processes of differential mutation, crossover, and selection in each evolution loop, driving the population toward better solutions.

Consequently, the different variants of differential evolution are categorized based on several key aspects: control parameters, the population structure, the initialization process, the differential mutation and the crossover strategies, and the selection methods. These classifications help in understanding and applying the most suitable variant for specific problem scenarios[31]

III.4.1. Control Parameters:

Differential Evolution (DE) is widely recognized as a simple yet powerful evolutionary algorithm for global optimization in many real-world problems. However, like other evolutionary algorithms, DE's performance is highly dependent on the settings of control parameters, such as the mutation factor and crossover probability. Although there are recommended values for these parameters, the relationship between parameter settings and convergence performance is complex and not fully understood. This complexity arises because no single set of parameter values works universally well across different problems or even throughout different stages of solving a single problem.

Typically, tuning these control parameters involves a trial-and-error approach, requiring numerous optimization trials. This is true even for algorithms like the classic DE/rand/1/bin, which maintain fixed parameters throughout the evolutionary search. To address this challenge, researchers have developed various adaptive and self-adaptive mechanisms that dynamically update control parameters during the search process. These mechanisms do not require the user's prior knowledge of the problem or interaction during the search. When well-designed, such adaptive strategies can significantly improve an algorithm’s convergence performance. [31]

Adaptation mechanisms in evolutionary algorithms can be categorized based on how the control parameters are modified. According to the classification scheme introduced by Angeline and further refined by Eiben et al., there are three main classes of parameter control mechanisms:

1. **Deterministic Parameter Control:**

Parameters are adjusted according to a predetermined rule without considering feedback from the evolutionary search. For example, Holland proposed a method where mutation rates change based on time.

2. **Adaptive Parameter Control:**

Parameters are dynamically adjusted based on feedback from the evolutionary search process. This means the algorithm can change its parameters in response to its performance. Examples include Rechenberg's "1/5-th rule" and fuzzy-logic adaptive evolutionary algorithms. Some recently developed DE algorithms, such as SaDE and jDE, as well as the JADE algorithm, also fall into this category. These methods observe how well the current parameter settings are performing and make adjustments to improve outcomes.

3. **Self-Adaptive Parameter Control:**

In this approach, the control parameters themselves evolve along with the solution candidates. Parameters are associated with individuals in the population and undergo mutation and recombination. Successful parameter values, which produce better solutions, are more likely to be passed on to future generations. This method embodies the concept of 'the evolution of evolution'. Some DE algorithms incorporate this self-adaptive mechanism.

Compared to deterministic methods, adaptive and self-adaptive parameter control can significantly enhance an algorithm's robustness and performance. These methods allow the algorithm to dynamically adjust its parameters to suit the characteristics of different fitness landscapes, making them versatile and effective across various optimization problems without requiring extensive manual tuning. Additionally, by continuously adapting control parameters to optimal values at different stages of the optimization process, these methods can improve the convergence rate, leading to faster and more reliable solutions. [31]

III.4.2. Population structure:

III.4.2.1. One-population:

In Differential Evolution (DE), the one-population strategy refers to the concept where there is a single evolving population that iterates through generations. Although two populations, $\boldsymbol{P}_n$ (current generation) and $\boldsymbol{P}_{n+1}$ (next generation), are used in implementation, they represent the same set of individuals at different stages of evolution. Each generation, individuals in $\boldsymbol{P}_n$ undergo mutation, crossover, and selection processes to create $\boldsymbol{P}_{n+1}$. After selection, P_{n+1} becomes the new $\boldsymbol{P}_n$

continuing the evolution. Thus, despite having two labels, DE operates on a single evolving population, making it a one-population strategy. [32]

III.4.2.2. Multi-population:

Multi-population differential evolution strategies are designed to enhance the performance and robustness of the differential evolution algorithm by using multiple interacting populations instead of a single evolving population. These strategies aim to improve the search efficiency, maintain diversity, and find multiple solutions in complex optimization problems. By dividing the population into subpopulations or incorporating auxiliary populations, these methods help in exploring the search space more thoroughly and avoiding premature convergence to suboptimal solutions. The following are specific multi-population strategies that have been developed to address various challenges in differential evolution. [32]

III.4.2.3. Auxiliary Population:

In classic differential evolution, a child $c_{n+1,i}$ is rejected if it is dominated by its parent p_{n+1}. In the worst-case scenario, the parent could be the most dominant individual in the population p_n, while the child might be better than all other individuals in p_n. To address this issue, an auxiliary population containing rejected children is introduced. Periodically, high-quality individuals from the auxiliary population replace low-quality individuals in the main population, ensuring that promising solutions are not discarded prematurely. [32]

III.4.2.4. Differential Evolution with Individuals in Groups:

For certain engineering problems, such as the benchmark electromagnetic inverse scattering problem, the problem dimension may be unknown, with only a finite set of possible dimensions available. Instead of trying each dimension one by one, which is highly time-consuming, differential evolution with individuals in groups is proposed. This strategy organizes the population into different groups, each focusing on one possible dimension and searching for the optimal solution. Groups dynamically compete and adjust their sizes, enhancing the efficiency of the search process. [32]

III.4.2.5. Surpopulation :

Classic differential evolution typically searches for a single solution using one evolving population. However, problems with multiple solutions require a different approach. By subdividing the population into multiple subpopulations based on the distances between individuals, multiple solutions can be found simultaneously. Each subpopulation evolves independently, but periodically, subpopulations interact through reorganization or migration to

avoid isolation. This method also aids in parallelization and maintaining diversity within the population. [32]

III.4.2.6. Opposition-Based Differential Evolution:

In opposition-based differential evolution, the opposite number $\hat{x}$ of a number x in the range [a,b] is defined as $\hat{x} = a + b - x$. For a vector of optimization parameters $x_{n,i}$, the opposite vector $\hat{x}_{n,i}$and the corresponding opposite individual $\hat{p}_{n,i}$ are defined. Inspired by the observation that $\hat{p}_{n,i}$ and $p_{n,i}$ have an equal chance of being dominant, opposite populations are introduced. This method employs an ordinary evolving population p_n along with an opposite population $\hat{p}_n$. Although the opposite population does not evolve independently, it is crucial in enhancing the search process. The Fortran-style pseudo-code for this method includes an additional control parameter, the jumping rate p_j, which regulates the integration of the opposite population.[32]

III.4.3. Initialization process:

III.4.3.1. Unbiased Initialization:

Typically, the initial population in differential evolution is created randomly and uniformly across the entire search space. This approach ensures a broad exploration of potential solutions, as every region of the search space has an equal chance of being sampled. By starting with a diverse population, the algorithm can effectively explore different areas of the search space, which helps in avoiding premature convergence and increases the likelihood of finding the global optimum.

III.4.3.2. Biased Initialization:

When prior knowledge about the problem is available, generating a biased initial population can be more advantageous. This approach uses a non-uniform probability density function to generate the initial optimization parameters, concentrating the initial population in regions of the search space that are more likely to contain optimal or near-optimal solutions. By focusing the search in these promising areas from the start, the algorithm can potentially converge faster and more effectively towards high-quality solutions. [32]

III.4.4. Mutation Strategies:

Differential mutation is widely recognized as the cornerstone of differential evolution's success because it drives the algorithm's ability to explore and exploit the search space effectively. By combining differences between randomly selected individuals, differential mutation creates new candidate solutions that can potentially lead to better performance. Given its critical role, numerous research efforts have focused on developing and refining various forms of differential

mutation to enhance the overall efficiency and effectiveness of the differential evolution algorithm. These efforts aim to optimize how new solutions are generated, thereby improving the algorithm's ability to solve complex optimization problems. [32]

III.4.4.1. Common Classical mutation schemes:

Scheme DE/rand/y:

Scheme DE/rand/y is a differential evolution strategy where "DE" stands for differential evolution, "rand" indicates that the base vector is selected randomly, and "/y" denotes the number of differential vectors used in the mutation process. This scheme can be scaled by adjusting "y" to include more differential vectors.

- Base Vector Selection: A random vector from the current population is chosen as the base vector.
- Differential Vectors: The scheme involves y differential vectors, which are the differences between randomly selected pairs of vectors from the population.
- Mutation Process: The mutant vector is generated by adding the weighted sum of these y differential vectors to the base vector.

For example, if y=2, the mutation can be represented as:

$$\mathbf{v}_i = \mathbf{x}_{r1} + F \cdot (\mathbf{x}_{r2} - \mathbf{x}_{r3}) + F \cdot (\mathbf{x}_{r4} - \mathbf{x}_{r5}) \quad \text{Equation III–5}$$

Here, $\mathbf{x}_{r1}$ is the random base vector, and F is the mutation factor that controls the amplification of the differential vectors $(\mathbf{x}_{r2} - \mathbf{x}_{r3})$ and $(\mathbf{x}_{r4} - \mathbf{x}_{r5})$.

Scheme DE/best/y:

Scheme DE/best/y is a differential evolution strategy where the base vector is the best-performing individual in the current population, and "/y" again denotes the number of differential vectors used.

- Base Vector Selection: The best vector in the current population, based on objective function values, is chosen as the base vector.
- Differential Vectors: This scheme also involves p differential vectors, similar to DE/rand/y.
- Mutation Process: The mutant vector is generated by adding the weighted sum of y differential vectors to the best vector.

For instance, if $y = 1$, the mutation can be represented as:

$$\mathbf{v}_i = \mathbf{x}_{lest} + F \cdot (\mathbf{x}_{r1} - \mathbf{x}_{r2}) \quad \text{Equation III–6}$$

Here, $\mathbf{x}_{lest}$ is the best vector, and $\mathbf{x}_{r1}$ and $\mathbf{x}_{r2}$ are randomly selected vectors.

Scheme DE/rand-to best/y:

Scheme DE/rand-to-best/y is a hybrid differential evolution strategy that combines elements of both random and best schemes.

- Base Vector Selection: A random vector from the current population is selected as the base vector.
- Differential Vectors: This scheme involves two differential components: one directed towards the best vector and y other differential vectors.
- Mutation Process: The mutant vector is generated by adding a weighted difference between the best vector and the base vector to the weighted sum of y differential vectors.

For example, if $y = 1$, the mutation can be represented as:

$$\mathbf{v}_i = \mathbf{x}_{r1} + F \cdot (\mathbf{x}_{best} - \mathbf{x}_{r1}) + F \cdot (\mathbf{x}_{r2} - \mathbf{x}_{r3}) \quad \text{Equation III–7}$$

Here, $\mathbf{x}_{r1}$ is the random base vector, $\mathbf{x}_{best}$ is the best vector, and $\mathbf{x}_{r2}$ and $\mathbf{x}_{r3}$ are additional randomly selected vectors.

Scheme DE/current-to best/y:

Scheme DE/current-to-best/y is another hybrid strategy where the base vector is the current vector, and there is a directed component towards the best vector.

Base Vector Selection: The current vector (the vector being mutated) is used as the base vector.

Differential Vectors: This scheme includes a differential component directed towards the best vector, in addition to p other differential vectors.

Mutation Process: The mutant vector is created by adding a weighted difference between the best vector and the current vector to the weighted sum of y differential vectors.

For instance, $y = 1$ the mutation can be represented as:

$$\mathbf{v}_i = \mathbf{x}_i + F \cdot (\mathbf{x}_{best} - \mathbf{x}_i) + F \cdot (\mathbf{x}_{r1} - \mathbf{x}_{r2}) \quad \text{Equation III–8}$$

Here, $\mathbf{x}_i$ is the current vector, $\mathbf{x}_{best}$ is the best vector, and $\mathbf{x}_{r1}$ and $\mathbf{x}_{r2}$ are randomly selected vectors. [28][27]

III.4.4.2. Differential-Free Mutation:

Differential-Free Mutation addresses the computational inefficiency associated with traditional differential evolution. Ali and colleagues recognized that calculating vector differences for each mutant can be time-consuming and may restrict the algorithm's exploration capabilities. To

overcome this limitation, they introduced the differential-free point generation scheme. Unlike conventional methods, where vector differences are computed afresh for each mutant, the differential-free mutation approach selects differences from a pre-established array of difference vectors. These vectors are periodically updated to adapt to changes in the search landscape, providing a more efficient and adaptable way to generate mutants in the differential evolution process. [32]

III.4.4.3. Binomial Mutation:

Binomial Mutation typically follows a uniform approach in generating all genes of the mutant. In 2003, B.V. Babu and A. Angira proposed an innovative mutation scheme inspired by binomial crossover techniques, thus termed as binomial mutation. This mutation strategy comprises two distinct mutation schemes, with each gene of the mutant being generated using one of these schemes. The selection of a particular mutation scheme for a gene is often determined through a Bernoulli experiment, where the choice is probabilistic. Binomial mutation is essentially a hybrid of two different mutation schemes, blending aspects of both into a unified approach. In this sense, it can also be referred to as a hybrid mutation strategy. [32]

III.4.4.4. Multiple Mutations:

In this particular approach, the strategy involves implementing multiple mutations to generate multiple offspring. This means that after the initial mutation, which produces the first child, further mutations are applied to create additional offspring. These subsequent mutations are categorized into two types: unconditional and conditional. In the case of an unconditional follow-up mutation, a child is directly produced without any conditions. However, in a conditional follow-up mutation, the creation of a child depends on the satisfaction of certain conditions. If the children born from previous mutations are not deemed satisfactory, a conditional follow-up mutation is triggered to generate a new child. This iterative process continues until a satisfactory child is obtained. Notably, this mutation strategy is integral to opposition-based differential evolution, where it plays a crucial role in generating diverse offspring to explore the search space effectively. [32]

III.4.4.5. Trigonometric Mutation:

Trigonometric Mutation, proposed by H.Y. Fan and J. Lampinen in 2003, introduces a novel strategy aimed at biasing the differential mutation towards the most dominant individual among the three individuals involved. This strategy leverages the objective function values of these individuals to guide the mutation process. Specifically, the mutation equation incorporates the objective function values of the individuals, aiming to highlight the influence of the most dominant individual. By applying trigonometric functions to the objective function values, the mutation

process is steered towards regions of the search space where the most promising solutions are likely to reside. This approach enhances the algorithm's ability to exploit promising areas of the search space, potentially leading to improved performance in optimization tasks. [32]

III.4.4.6. Perturbation mutation:

The perturbation mutation serves the purpose of averting premature convergence in the optimization process. This is achieved by injecting an independent noise term into the mutant vector, thereby introducing variability that prevents the algorithm from prematurely settling into a local optimum. Essentially, the mutation process is augmented with random perturbations to explore a broader solution space and potentially discover superior solutions. [32]

III.4.5. Crossover Strategies:

Crossover has often been considered non-essential for differential evolution (DE). In fact, some DE strategies don't use crossover at all. However, recent studies suggest that the importance of crossover in DE might be significantly underestimated.

Crossover has been extensively studied in genetic algorithms, and almost all crossover schemes used there can be implemented in DE with little to no modification. Besides the previously mentioned binomial crossover, several other crossover schemes are commonly applied in DE and are summarized here.

In most evolutionary algorithms, the trial vector U_{n+1} is required to be different from its parents (target and mutant vectors). This convention is generally followed in DE, although it is not strictly necessary.[26]

III.4.5.1. Exponential crossover:

Exponential crossover is a method used in differential evolution to combine elements from the mutant vector and the target vector to create a trial vector. Here's how it works in more detail:

1. *Random Starting Point:* An integer r is randomly selected from the range [1, N], where N is the dimensionality of the vectors. This r marks the starting position for the crossover operation.
2. *Initial Component Assignment:* The component $\mathbf{c}_{i,r,n}$ of the trial vector $\mathbf{c}_{i,n}$ is taken from the corresponding component $\mathbf{v}_{i,r,n}$ of the mutant $\mathbf{v}_{i,n}$.
3. *Bernoulli Experiments:* From this starting point, the process continues to the next component (in a cyclic manner) and decides whether to take each subsequent component from the mutant

vector or to stop and take the rest from the target vector. This decision is based on a series of Bernoulli experiments, each with a probability Cr of success.

4. *Component Donation:* The mutant vector continues to donate its components to the trial vector as long as the Bernoulli experiments succeed. If the experiment fails, or if the crossover length reaches N−1, the remaining components of the trial vector are taken from the target vector $\mathbf{x}_{i,n}$.
5. *Cyclic Nature:* If the end of the vector is reached, the crossover continues cyclically from the beginning of the vector. This ensures that the crossover can span the entire length of the vector without interruption. [26]

The purpose of this crossover scheme is to introduce diversity into the population by combining genetic material from both the mutant and target vectors, thus creating trial vectors that may have better fitness values. The probabilistic nature of the Bernoulli experiments ensures that the crossover is not deterministic, allowing for a more diverse exploration of the solution space.

$\mathbf{x}^{n,i}$	28.69	35.09	57.82	12.06	26.99	82.96	65.30	52.68
$\mathbf{x}^{n+1,v,i}$	15.23	16.22	78.33	68.12	32.88	67.55	99.28	85.86
Bernoulli experiments	1	0						1
$\mathbf{x}^{n+1,c,i}$	15.23	35.09	57.82	12.06	26.99	82.96	99.28	85.86

Figure 19: Consecutive exponential[26]

III.4.5.2. One-Point Crossover:

One-point crossover is a straightforward yet effective technique used in differential evolution (DE) to combine the genetic material from the mutant and target vectors to form a trial vector. This process enhances diversity within the population, potentially leading to better optimization outcomes.

In this method, a single crossover point r is randomly chosen within the $1 < r \leq N$, where N is the number of components in the vectors. The crossover point rrr determines where the combination of the two parent vectors will occur. This point effectively divides both the target vector $\mathbf{x}_{i,n}$ and the mutant vector $\mathbf{v}_{i,n}$ into two segments. The first segment includes the components from the beginning of the vector up to, but not including, the crossover point r.

The second segment comprises the components from the crossover point r to the end of the vector.Once the crossover point is determined, the construction of the trial vector $\mathbf{c}_{i,n}$ begins. The components from the start of the vector up to the crossover point are copied from the target

vector $\mathbf{x}_{i,n}$. The remaining components, from the crossover point rrr to the end, are taken from the mutant vector v . This way, the trial vector incorporates genetic information from both the target and mutant vectors, ensuring that it is not an identical copy of either parent. [26]

$\mathbf{x}^{n,i}$	28.69	35.09	57.82	12.06	26.99	82.96	65.30	52.68
$\mathbf{x}^{n+1,v,i}$	15.23	16.22	78.33	68.12	32.88	67.55	99.28	85.86
$\mathbf{x}^{n+1,c,i}$	28.69	35.09	57.82	68.12	32.88	67.55	99.28	85.86

Figure 20: One-point Crossover[26]

III.4.5.3. Multi-point Crossover:

Multi-point crossover is an advanced method used in differential evolution (DE) to create trial vectors by combining elements from both the mutant and target vectors at multiple crossover points. This technique aims to increase genetic diversity in the population, potentially leading to more robust solutions.

In multi-point crossover, several crossover points are randomly selected within the vector's range. These points determine where the switching between the target vector $\mathbf{x}_{i,n}$ and the mutant vector $\mathbf{v}_{i,n}$ will occur. By having multiple crossover points, the trial vector $\mathbf{c}_{i,n}$ inherits segments from both parent vectors, creating a more intricate combination of genetic material.

Once the crossover points are determined, the trial vector is constructed by alternately taking segments from the target and mutant vectors. Starting from the beginning of the vector, components are copied from one parent vector until the first crossover point is reached. Then, the source of the components switches to the other parent vector until the next crossover point is reached, and so on. This process continues until the end of the vector is reached, resulting in a trial vector that is a mosaic of the target and mutant vectors. [26]

$\mathbf{x}^{n,i}$	28.69	35.09	57.82	12.06	26.99	82.96	65.30	52.68
$\mathbf{x}^{n+1,v,i}$	15.23	16.22	78.33	68.12	32.88	67.55	99.28	85.86
$\mathbf{x}^{n+1,c,i}$	28.69	35.09	78.33	68.12	32.88	82.96	65.30	52.68

Figure 21: Two-point Crossover[26]

III.4.5.4. Arithmetic Crossover:

Arithmetic Crossover involves generating a new vector, denoted as $\mathbf{c}_{i,n}$ that lies on a line between two parent vectors, $\mathbf{v}_{i,n}$ and $\mathbf{x}_{i,n}$. This is achieved through a linear combination of the two parent vectors. Specifically, the new vector ccc is calculated as follows:

$$x_j{}^{n+1,c,i} = x_j{}^{n+1,v,i} + h(x_j^{n,i} - x_j{}^{n+1,v,i}) \qquad \text{Equation III–9}$$

where *h* represents the crossover intensity, an intrinsic control parameter that dictates the extent of influence each parent vector has on the offspring. This method ensures that the new vector is a blend of both parent vectors, providing a balance between exploration and exploitation in the search space.

$\mathbf{x}^{n,i}$	28.69	35.09	57.82	12.06	26.99	82.96	65.30	52.68
$\mathbf{x}^{n+1,v,i}$	15.23	16.22	78.33	68.12	32.88	67.55	99.28	85.86
$\mathbf{x}^{n+1,c,i}$	16.576	18.107	76.279	62.514	32.291	69.091	95.891	82.542

Figure 22: Arithmetic Crossover[26]

Additionally, there are various types of arithmetic crossover methods, each introducing different ways to combine the parent vectors:

- Arithmetic One-Point Crossover: Combines the parent vectors at a single crossover point, resulting in one segment from one parent and the rest from the other.
- Arithmetic Multi-Point Crossover: Uses multiple crossover points to mix segments from both parents, increasing diversity in the offspring.
- Arithmetic Binomial Crossover: Involves a binomial distribution to decide the contribution from each parent for each gene.
- Arithmetic Exponential Crossover: Uses an exponential function to determine the contribution from each parent, providing a different blending mechanism that can affect convergence rates.

These variations on arithmetic crossover allow for greater flexibility and adaptability in the search process, enhancing the algorithm's ability to explore and exploit the search space effectively. [26]

III.4.6. Selection methods:

III.4.6.1. Classical differential evolution selection scheme:

In the classic scheme of differential evolution, during the competition phase, each trial vector $c_{n+1,i}$ contends directly against its target vector $p_{n,i}$ There's no competition between target vectors or between trial vectors. Thus, a trial vector is discarded if it doesn't surpass its corresponding

target vector even if it outperforms all other vectors in the population p_n except its target vector. Moreover, if a trial vector is superior to its target vector, it replaces the target vector, even if the target vector is the best in the population p_n. Consequently, the generation of very similar trial vectors is probable, posing a risk to population diversity. This observation has spurred the development of various alternative selection schemes to mitigate this issue. [32]

III.4.6.2. Cross-selection:

In the cross-selection scheme, a trial vector $c_{n+1,i}$ competes with its target vector $p_{n,i}$but doesn't displace it. Instead, it replaces other parent individuals. The removed parent can be chosen randomly from the parent population p_n, randomly selected from parents dominated by $c_{n+1,i}$, or the worst individual in p_n.[32]

III.4.6.3. Group Selection:

Group selection aims to maintain dominant individuals in the combined set of trial vectors c_{n+1} and target vectors p_n. This approach is widely used in various evolutionary algorithms. However, implementing group selection requires ranking all individuals in $C_{n+1} \cup P_n$ based on specific criteria, which can be computationally demanding, particularly with large population sizes. [32]

III.4.6.4. Similarity Selection:

Similarity selection seeks to enhance population diversity. Here, a trial vector $c_{n+1,i}$ competes not with its direct target vector, but with the most similar parent individual in p_n. If $c_{n+1,i}$ dominates this parent individual, the latter is replaced. However, similarity selection can also be computationally expensive as it necessitates computing the similarities (or distances) between the trial vector and all parent individuals in p_n.[32]

III.4.6.5. Threshold Margin Selection:

Threshold margin selection addresses challenges posed by noisy problems in differential evolution. Under this scheme, a trial vector $c_{n+1,i}$ replaces its target vector in p_n if it dominates it by a margin threshold, which is proportional to the strength of the noise present in the objective and/or constraint functions.[32]

III.5. Conclusion:

In conclusion, Differential Evolution (DE) remains a powerful and flexible algorithm for addressing a diverse range of complex optimization problems. This chapter has provided a comprehensive overview of DE's fundamental components and operational mechanisms, including control parameters, initialization methods, mutation strategies, crossover processes, and selection techniques. Additionally, it has highlighted various DE variants, demonstrating how these adaptations can be implemented.

In the next chapter, we will delve deeper into the influence of control parameters and different DE variants on the algorithm's performance. This will involve conducting extensive tests using benchmark functions and performing rigorous statistical analyses on the results. Moreover, our study will extend beyond theoretical exploration; we will implement DE in MATLAB to offer a practical, hands-on understanding of its application and performance. This approach will enable us to observe the algorithm in action, showcasing its capabilities and uncovering any potential limitations. Through this comprehensive investigation, we aim to provide valuable insights into DE's practical effectiveness and areas for improvement.

Chapter IV
Experimental study

Chapitre IV. **Experimental Study**

IV.1. Introduction:

In this chapter, we present a study on the impact of control parameters on the performance of the DE algorithm, followed by a comparative study of different DE variants. To achieve this, we conduct several tests on benchmark functions, and their results will be analyzed statistically.

The first study focuses on examining the impact of control parameters on the DE algorithm's performance, the aim of this study is to analyze the effect each control parameter on the performance of the DE algorithm and identify their optimal value in the context of global optimization. We conducted five individual tests, one for each control parameter: problem dimension, number of iterations, mutation rate, population size, and scaling factor. While studying each parameter, the others will be set to average values to maintain consistency. Additionally, we investigated the relationship between the population and the scaling factor.

The second study involves a comparative analysis of different DE variants to identify the most effective one for global optimization. We performed three distinct tests to evaluate their performances. The first test assessed the convergence quality by evaluating the solution within a limited number of iterations (fixed-cost solution results). The second test measured the convergence speed by determining the number of iterations required to achieve an acceptable solution (fixed-target cost results)[33]. The third test evaluated the solve time for a fixed-cost solution, which involves measuring the actual computation time taken to reach a solution within a predefined number of iterations. The results from these tests were analyzed statistically to draw meaningful conclusions.

IV.2. Benchmark Functions:

We present below twenty-four benchmark functions that are frequently used to evaluate optimization algorithms, and which we use in our study. All these functions are diverse and have different ranges and minimums. Some of them are fixed dimension and others are scalable.

Table 2: Benchmark Fonctions.

Name	Function	Search Limit	Min
Ackley	$\boldsymbol{f_1 = 20 + \exp(1) - 20\exp\left(-0.2\sqrt{(1/D)\sum_{i=1}^{D} x_i^2}\right) - \exp\left((1/D)\sum_{i=1}^{D}\cos(2\pi x_i)\right)}$	$[-32.768,32.768]^D$	1E-6
Alpine	$f_2 = \sum_{i=1}^{D} \lvert x_i \sin(x_i) + 0.1x_i \rvert$	$[-10,10]^D$	0
Discus	$f_3 = 10^6 x_1^2 + \sum_{i=2}^{D} x_i^2$	$[-100,100]^D$	0
Expanded Schaffer's	$f_4 = g(x_1,x_2) + g(x_2,x_3) + \cdots + g(x_{D-1},x_D) + g(x_D,x_1)$ Where $g(x,y) = 0.5 + \left(\sin^2\left(\sqrt{x^2+y^2}\right) - 0.5\right)/\left(1 + 0.001(x^2+y^2)\right)^2$	$[-100,100]^D$	0
Exponential	$f_s = -\exp\left(-0.5\sum_{i=1}^{D} x_i^2\right)$	$[-1,1]^D$	-1
Griewank	$f_6 = \sum_{i=1}^{D}(x_i^2 / 4000) - \prod_{i=1}^{D}\cos\left(x_i / \sqrt{i}\right) + 1$	$[-600,600]^D$	0
Happycat	$f_7 = \left\lvert \sum_{i=1}^{D} x_i^2 - D \right\rvert^{1/4} + \left(0.5\sum_{i=1}^{D} x_i^2 + \sum_{i=1}^{D} x_i\right)/D + 0.5$	$[-100,100]^D$	0
HGBat	$f_8 = \sqrt{\left\lvert \left(\sum_{i=1}^{D} x_i^2\right)^2 - \left(\sum_{i=1}^{D} x_i\right)^2 \right\rvert} + \left(0.5\sum_{i=1}^{D} x_i^2 + \sum_{i=1}^{D} x_i\right)/D + 0.5$	$[-5,5]^D$	0
High Conditioned Elliptic	$f_9 = \sum_{i=1}^{D} (10^6)^{\frac{i-1}{D-1}} x_i^2$	$[-100,100]^D$	0
Levy	$f_{10} = \sin^2(\pi\omega_1) + \sum_{i=1}^{D-1}\left[(\omega_i - 1)^2\left(1 + 10\sin^2(\pi\omega_i + 1)\right)\right] + (\omega_D - 1)^2\left(1 + \sin^2(2\pi\omega_D)\right)$ Where $\omega_j = 1 + \left((x_i - 1)/4\right)$	$[-10,10]^D$	0
Quartic	$f_{11} = \sum_{i=1}^{D}(ix_i^4) + \text{random}[0,1)$	$[-1.28,1.28]^D$	0
Rastrigin	$f_{12} = \sum_{i=1}^{D}(x_i^2 - 10\cos(2\pi x_i) + 10)$	$[-5.12,5.12]^D$	0
Rosenbrock	$f_{13} = \sum_{i=1}^{D-1}[\,100(x_i^2 - x_{i+1})^2 + (x_i - 1)^2]$	$[-2.048,2.048]^D$	0
Sphere	$f_{14} = \sum_{i=1}^{D} x_i^2$	$[-100,100]^D$	0
SumSquares	$f_{15} = \sum_{i=1}^{D} i^* x_i^2$	$[-10,10]^D$	0
Weierstrass	$f_{16} = \sum_{i=1}^{D}[\sum_{k=0}^{20}(0.5^k \cdot \cos(2\pi 3^k(x_i + 0.5)))] - D\sum_{k=0}^{20}(0.5^k\cos(2\pi \cdot 3^k \cdot 0.5))$	$[-100,100]^D$	0
Whitley	$f_{17} = \sum_{i=1}^{D}\sum_{j=1}^{D}\left[\left[\frac{1}{4000}(100(x_i^2 - x_j)^2 + \left(1 - x_j\right)^2\right)^2\right] - \cos\left(100(x_i^2 - x_j)^2 + \left(1 - x_j\right)^2\right) + 1$	$[-10.24,10.24]^D$	0
Zakharov	$f_{18} = \sum_{i=1}^{D} x_i^2 + \left(\sum_{i=1}^{D} 0.5 \cdot i \cdot x_i\right)^2 + \left(\sum_{i=1}^{D} 0.5 \cdot i \cdot x_i\right)^4$	$[-5,10]^D$	0
	Fixed dimension Numerical Benchmark Functions		0
Beale	$f_{19}(x_1,x_2) = (1.5 - x_1 + x_1x_2)^2 + (2.25 - x_1 + x_1x_2^2)^2 + (2.625 - x_1 + x_1x_2^3)^2$	$[-4.5,4.5]^2$	0
Booth	$f_{20}(x_1,x_2) = (x_1 + 2x_2 - 7)^2 + (2x_1 + x_2 - 5)^2$	$[-10,10]^2$	0
Branin	$f_{21}(x_1,x_2) = 1(x_2 - (5.1/(2\pi)^2)x_1^2 + (5/\pi)x_1 - 6)^2 + 10(1 - (1/(8\pi)))\cos(x_1) + 10$	[0,15]	0.387
Easom	$f_{22}(x_1,x_2) = -\cos(x_1)\cos(x_2)\exp(-(x_1 - \pi)^2 - (x_2 - \pi)^2)$	$[-100,100]^2$	-1
Matyas	$f_{23}(x_1,x_2) = 0.26(x_1^2 + x_2^2) - 0.48x_1x_2$	$[-10,10]^2$	0
Shubert	$f_{24}(x_1,x_2) = \sum_{i=1}^{3} i\cos\left((i+1)x_1 + i\right)\sum_{i=1}^{3} i\cos\left((i+1)x_2 + i\right)$	$[-10,10]^2$	-187

IV.3. Experimental setup:

In the first study (control parameters effects), we used the DE/rand/1/bin algorithm to examine the impact of control parameters in terms of the solution quality by conducting five tests, one for each control parameter. While in the second study (DE variants comparison), we compared five different mutation strategies in the DE algorithm: Rand/1, Rand/2, Best/1, Best/2, and Rand-to-Best/1. Before starting the comparison, we needed to select a crossover method. To do this, I compared two DE crossover variants (binomial and exponential) using the same mutation strategy (Rand/1). The results showed that the binomial method (bin) performed better. Therefore, the DE variants we compared were: Rand/1/bin, Rand/2/bin, Best/1/bin, Best/2/bin, and Rand-to-Best/1/bin. For a detailed comparison, we ran three distinguished tests (convergence quality test, convergence speed test, and execution time test).[34][33]

Due to the stochastic nature of the DE algorithm, results can vary with each run. To obtain accurate and reliable results, each test was run independently 100 times, and we calculated the mean, median, and standard deviation of these results. This approach helped mitigate the randomness inherent in the algorithm, providing a more precise evaluation. Additionally, we applied the Friedman test to the results to statistically analyze the performance differences across different parameter settings. The Friedman test is a non-parametric test used to detect differences in treatments across multiple test attempts. In this study, it ranked the performance of different control parameters and variants, computing a mean rank for each one of them based on their results across different benchmark functions. The test also generated a p-value, which indicated the probability that the observed differences in ranks occurred by chance. A low p-value (typically less than 0.05) suggested that the differences in algorithm performance were statistically significant, while a high p-value indicated no significant difference. By applying the Friedman test, we could more rigorously determine which control parameters or DE variant had a statistically significant impact on performance, thus providing stronger evidence for the optimal settings. [35]

In each table, the "Best in" row represented the number of best cases where each control parameter value achieved the best solutions, while the "Friedman Rank" row provided the mean rank for each control parameter value, and the "Friedman Prob" indicated the p-value. The tables highlighted the best solution for each case using bold text, while the highest number of best cases and the best mean rank were highlighted with bold text on a grey background.

We conducted all of the tests of the first and second study using Matlab_2021 on AMD Rayzon 9 4900HS 3.00 GHz processor.

IV.4. Numerical results and discussion:

In this section we will present, compare, and discus the results of our two studies and tests. The results of our first study will be discussed under the title 'Control Parameters Effects' and the second one under 'DE variants comparison'.

IV.4.1. Control Parameters Effects:

A. Problem dimension:

Table 3: Impact of problem dimension on the DE Algorithm Results

Function	Dimension	Mean	Median	STD
f_1	10	**3.32E+00**	2.83E+00	1.83E+00
	30	8.28E+00	8.29E+00	1.67E+00
	50	1.01E+01	1.01E+01	1.12E+00
	100	1.32E+01	1.32E+01	8.42E-01
f_2	10	**1.49E-02**	3.18E-03	4.32E-02
	30	5.20E-01	4.44E-01	3.94E-01
	50	2.32E+00	2.12E+00	1.04E+00
	100	1.26E+01	1.22E+01	2.90E+00
f_3	10	**4.54E+02**	1.83E+02	6.62E+02
	30	4.03E+03	3.64E+03	2.42E+03
	50	1.06E+04	9.80E+03	4.65E+03
	100	3.92E+04	3.69E+04	2.02E+04
f_4	10	**1.14E+00**	1.31E+00	6.54E-01
	30	6.81E+00	6.77E+00	1.33E+00
	50	1.46E+01	1.45E+01	1.59E+00
	100	3.77E+01	3.72E+01	2.75E+00
f_5	10	**-9.91E-01**	-9.96E-01	1.27E-02
	30	-9.02E-01	-9.13E-01	5.90E-02
	50	-7.41E-01	-7.47E-01	8.83E-02
	100	-2.92E-01	-2.95E-01	8.64E-02
f_6	10	**1.49E+00**	6.96E-01	2.12E+00
	30	1.57E+01	1.42E+01	9.93E+00
	50	5.16E+01	4.96E+01	1.96E+01
	100	2.08E+02	2.07E+02	4.38E+01
f_7	10	**8.21E+00**	4.97E+00	8.66E+00
	30	3.47E+01	2.84E+01	2.14E+01
	50	5.98E+01	5.78E+01	1.94E+01
	100	1.30E+02	1.29E+02	2.76E+01
f_8	10	**3.83E-01**	3.45E-01	2.54E-01
	30	1.88E+00	7.63E-01	2.75E+00
	50	1.18E+01	1.17E+01	7.64E+00
	100	5.61E+01	5.50E+01	1.22E+01
f_9	10	**3.86E+05**	8.54E+04	8.80E+05
	30	2.48E+07	1.72E+07	2.78E+07
	50	8.68E+07	5.83E+07	8.23E+07
	100	4.22E+08	4.04E+08	1.97E+08

Function	Dimension	Mean	Median	STD
f_{10}	10	**3.19E-01**	9.52E-02	5.46E-01
	30	3.72E+00	3.30E+00	2.15E+00
	50	9.91E+00	9.17E+00	3.65E+00
	100	4.27E+01	4.14E+01	8.89E+00
f_{11}	10	**1.67E-02**	4.68E-03	3.46E-02
	30	8.60E-01	6.81E-01	6.90E-01
	50	4.82E+00	4.57E+00	2.82E+00
	100	5.14E+01	4.99E+01	1.69E+01
f_{12}	10	**3.75E+00**	3.34E+00	1.84E+00
	30	2.24E+01	2.14E+01	7.09E+00
	50	5.06E+01	5.03E+01	1.06E+01
	100	1.84E+02	1.83E+02	3.01E+01
f_{13}	10	**1.78E+01**	1.25E+01	1.54E+01
	30	1.62E+02	1.57E+02	4.84E+01
	50	3.95E+02	3.87E+02	9.05E+01
	100	1.49E+03	1.45E+03	3.21E+02
f_{14}	10	**1.33E+02**	5.26E+01	1.99E+02
	30	1.80E+03	1.58E+03	1.06E+03
	50	5.14E+03	4.58E+03	2.36E+03
	100	2.43E+04	2.36E+04	5.11E+03
f_{15}	10	**4.71E+00**	1.48E+00	7.69E+00
	30	2.41E+02	2.03E+02	1.53E+02
	50	1.31E+03	1.23E+03	5.26E+02
	100	1.08E+04	1.02E+04	2.53E+03
f_{16}	10	**4.52E-06**	3.55E-14	2.74E-05
	30	2.07E-01	4.18E-06	6.69E-01
	50	3.48E+00	2.61E+00	3.11E+00
	100	2.70E+01	2.77E+01	8.62E+00
f_{17}	10	**2.90E+04**	5.24E+02	1.22E+05
	30	7.72E+06	3.18E+06	1.42E+07
	50	6.78E+07	4.11E+07	1.14E+08
	100	1.15E+09	9.17E+08	8.84E+08
f_{18}	10	**3.01E+00**	1.35E+00	4.90E+00
	30	3.87E+01	3.31E+01	2.55E+01
	50	1.68E+02	1.61E+02	4.35E+01
	100	1.07E+03	1.07E+03	1.07E+02

Our initial test examines the impact of problem dimension on DE algorithm performance. We conduct tests for each of the dimension D = {10, 30, 50, 100} on nineteen scalable benchmark functions, with the population size set to NP = 20, scaling factor F = 0.5, mutation rate CR = 0.5, and iterations GEN = 1000. The effect of the problem dimension is straightforward and predictable, according to the results of our simulations presented in Table 3, we observe a consistent behavior for all functions used, characterized by a deterioration in the results as the

dimension increases. This means that as the complexity of the problem increases, the DE algorithm's performance declines. Higher dimensions make the search space larger and more complex, making it harder for the algorithm to find optimal solutions efficiently. This behavior of the DE algorithm, specifically the Rand/1/bin variant, is consistent across other variants as well. It's also important to note that increasing the parameters requires greater computational resources, resulting in longer execution times.

B. Iterations Number:

Table 4: Impact of Iteration Number on the DE Algorithm Results

Function	Iteration	Mean	Median	STD
f_1	500	4.27E-07	4.62E-12	4.27E-06
	1000	4.44E-15	4.44E-15	0.00E+00
	2000	4.44E-15	4.44E-15	0.00E+00
	5000	**4.44E-15**	4.44E-15	0.00E+00
f_2	500	2.34E-03	2.46E-03	7.63E-04
	1000	1.43E-03	1.43E-03	0.00E+00
	2000	9.03E-05	9.03E-05	0.00E+00
	5000	**0.00E+00**	0.00E+00	0.00E+00
f_3	500	9.02E-12	1.15E-21	9.02E-11
	1000	2.47E-48	2.47E-48	0.00E+00
	2000	2.11E-100	2.11E-100	0.00E+00
	5000	**0.00E+00**	0.00E+00	0.00E+00
f_4	500	5.29E+00	5.35E+00	3.05E-01
	1000	3.32E+00	3.32E+00	0.00E+00
	2000	9.80E-01	9.80E-01	0.00E+00
	5000	**8.73E-01**	8.73E-01	0.00E+00
f_5	500	-1.00E+00	-1.00E+00	0.00E+00
	1000	-1.00E+00	-1.00E+00	0.00E+00
	2000	-1.00E+00	-1.00E+00	0.00E+00
	5000	**-1.00E+00**	-1.00E+00	0.00E+00
f_6	500	1.08E-04	0.00E+00	8.12E-04
	1000	0.00E+00	0.00E+00	0.00E+00
	2000	0.00E+00	0.00E+00	0.00E+00
	5000	**0.00E+00**	0.00E+00	0.00E+00
f_7	500	2.80E-01	2.81E-01	4.13E-02
	1000	2.64E-01	2.64E-01	0.00E+00
	2000	2.38E-01	2.38E-01	0.00E+00
	5000	**1.38E-01**	1.38E-01	0.00E+00
f_8	500	3.90E-01	3.91E-01	5.63E-02
	1000	3.94E-01	3.94E-01	0.00E+00
	2000	3.77E-01	3.77E-01	0.00E+00
	5000	**2.65E-01**	2.65E-01	0.00E+00
f_9	500	7.61E-03	7.51E-19	7.61E-02
	1000	2.50E-44	2.50E-44	0.00E+00
	2000	1.11E-96	1.11E-96	0.00E+00
	5000	**0.00E+00**	0.00E+00	0.00E+00
f_{10}	500	5.12E-24	4.21E-24	4.25E-24
	1000	1.50E-32	1.50E-32	0.00E+00
	2000	1.50E-32	1.50E-32	0.00E+00
	5000	**1.50E-32**	1.50E-32	0.00E+00
f_{11}	500	6.61E-03	6.74E-03	1.83E-03
	1000	2.34E-03	2.34E-03	0.00E+00
	2000	2.16E-03	2.16E-03	0.00E+00
	5000	**2.81E-04**	2.81E-04	0.00E+00
f_{12}	500	2.00E+01	1.99E+01	5.76E+00
	1000	9.29E+00	9.29E+00	0.00E+00
	2000	0.00E+00	0.00E+00	0.00E+00
	5000	**0.00E+00**	0.00E+00	0.00E+00

Function	Iteration	Mean	Median	STD
f_{13}	500	1.71E+01	1.68E+01	5.10E+00
	1000	1.57E+01	1.57E+01	0.00E+00
	2000	1.57E+01	1.57E+01	0.00E+00
	5000	**1.57E+01**	1.57E+01	0.00E+00
f_{14}	500	4.04E-13	2.68E-22	4.04E-12
	1000	1.27E-48	1.27E-48	0.00E+00
	2000	1.05E-100	1.05E-100	0.00E+00
	5000	**5.11E-211**	5.11E-211	0.00E+00
f_{15}	500	1.80E-20	2.35E-23	1.79E-19
	1000	7.01E-49	7.01E-49	0.00E+00
	2000	1.25E-101	1.25E-101	0.00E+00
	5000	**0.00E+00**	0.00E+00	0.00E+00
f_{16}	500	1.19E-14	1.42E-14	9.68E-15
	1000	0.00E+00	0.00E+00	0.00E+00
	2000	0.00E+00	0.00E+00	0.00E+00
	5000	**0.00E+00**	0.00E+00	0.00E+00
f_{17}	500	1.41E+02	1.50E+02	5.06E+01
	1000	8.89E+01	8.89E+01	0.00E+00
	2000	5.38E+00	5.38E+00	0.00E+00
	5000	**3.98E+00**	3.98E+00	0.00E+00
f_{18}	500	1.27E+01	1.25E+01	4.37E+00
	1000	2.37E-02	2.37E-02	0.00E+00
	2000	6.59E-09	6.59E-09	0.00E+00
	5000	**3.49E-61**	3.49E-61	0.00E+00
f_{19}	500	0.00E+00	0.00E+00	0.00E+00
	1000	0.00E+00	0.00E+00	0.00E+00
	2000	0.00E+00	0.00E+00	0.00E+00
	5000	**0.00E+00**	0.00E+00	0.00E+00
f_{20}	500	0.00E+00	0.00E+00	0.00E+00
	1000	0.00E+00	0.00E+00	0.00E+00
	2000	0.00E+00	0.00E+00	0.00E+00
	5000	**0.00E+00**	0.00E+00	0.00E+00
f_{21}	500	3.98E-01	3.98E-01	0.00E+00
	1000	3.98E-01	3.98E-01	0.00E+00
	2000	3.98E-01	3.98E-01	0.00E+00
	5000	**3.98E-01**	3.98E-01	0.00E+00
f_{22}	500	-1.00E+00	-1.00E+00	0.00E+00
	1000	-1.00E+00	-1.00E+00	0.00E+00
	2000	-1.00E+00	-1.00E+00	0.00E+00
	5000	**-1.00E+00**	-1.00E+00	0.00E+00
f_{23}	500	3.00E-64	5.46E-66	1.44E-63
	1000	6.49E-131	6.49E-131	0.00E+00
	2000	3.97E-259	3.97E-259	0.00E+00
	5000	**0.00E+00**	0.00E+00	0.00E+00
f_{24}	500	-1.87E+02	-1.87E+02	2.54E-14
	1000	-1.87E+02	-1.87E+02	0.00E+00
	2000	-1.87E+02	-1.87E+02	0.00E+00
	5000	**-1.87E+02**	-1.87E+02	0.00E+00

The second test examines the effect of the number of iterations on DE algorithm performance. We conduct tests for each of the iteration numbers GEN = {500, 1000, 2000, 5000} on 24 benchmark functions, with the population size set to NP = 20, scaling factor F = 0.5, mutation rate CR = 0.5, and problem dimension D = 20.

The effect of the number of iterations is also straightforward and predictable, looking at the results our simulations presented in Table 4, it is evident in all functions that the solutions improve as we increase the number of iterations. This is because a higher number of iterations provides the algorithm with more opportunities to explore the search space and refine solutions. Consequently, the algorithm can converge more effectively towards optimal or near-optimal solutions.

C. Mutation rate:

The final test focuses on the effect of the crossover rate on DE algorithm performance. We conduct tests for each of the mutation rates CR = {0.1, 0.5, 0.9, 1} on 24 benchmark functions, with the number of iterations set to GEN=1000, scaling factor population size NP = 20, the scaling factor F = 0.5, and problem dimension D=20.

Looking at the results of this test presented in Table 5, we observe that a mutation rate value of 0 produced the best results for the vast majority of benchmark functions. This value achieved the highest number of best cases and the best mean rank of 1.2917.

We can also note that the lower the mutation rate, the better the results we obtained. However, we should note that in this specific case setting the mutation rate to zero is equivalent to setting it to 0.05 because the problem dimension is set to 20 and knowing that in the binomial crossover method at least one component is taken from the mutant vector to ensure that the trial vector does not duplicate the target vector (parent).

Table 5: Impact of mutation rate on the DE Algorithm Results.

Function	Mutation Rate %	Mean	Median	STD	Function	Mutation Rate	Mean	Median	STD
f_1	0	**9.56E-02**	2.40E-11	2.59E-01	f_{13}	0	**1.27E+01**	1.27E+01	7.04E+00
	0.3	3.82E+00	3.62E+00	2.03E+00		0.3	5.09E+01	4.58E+01	2.54E+01
	0.5	6.15E+00	6.05E+00	2.08E+00		0.5	8.45E+01	8.15E+01	3.04E+01
	1	1.75E+01	1.76E+01	1.00E+00		1	8.91E+02	7.94E+02	4.50E+02
f_2	0	**6.34E-06**	2.33E-10	6.25E-05	f_{14}	0	**3.44E-02**	2.03E-22	3.12E-01
	0.3	2.98E-02	7.48E-03	6.84E-02		0.3	1.82E+02	8.37E+01	2.01E+02
	0.5	1.60E-01	7.54E-02	2.37E-01		0.5	7.02E+02	4.66E+02	7.27E+02
	1	1.94E+01	1.94E+01	3.39E+00		1	1.44E+04	1.43E+04	4.50E+03
f_3	0	**6.21E-01**	5.41E-21	6.16E+00	f_{15}	0	**3.57E-02**	1.19E-23	3.02E-01
	0.3	3.15E+02	2.17E+02	3.47E+02		0.3	1.44E+01	8.72E+00	1.94E+01
	0.5	6.95E+03	1.33E+03	5.28E+04		0.5	6.73E+01	4.85E+01	6.15E+01
	1	4.35E+04	3.42E+04	4.69E+04		1	1.30E+03	1.27E+03	4.18E+02
f_4	0	**8.88E-01**	9.12E-01	1.07E-01	f_{16}	0	4.40E-03	2.75E-05	9.57E-03
	0.3	2.42E+00	2.34E+00	9.67E-01		0.3	**1.40E-03**	7.82E-14	6.64E-03
	0.5	3.65E+00	3.62E+00	9.98E-01		0.5	4.40E-02	9.95E-14	1.56E-01
	1	9.17E+00	9.09E+00	2.77E-01		1	4.18E+00	3.47E+00	3.40E+00
f_5	0	**-1.00E+00**	-1.00E+00	3.13E-04	f_{17}	0	**4.06E+04**	2.23E+04	6.92E+04
	0.3	-9.88E-01	-9.93E-01	1.49E-02		0.3	1.42E+05	5.76E+03	6.05E+05
	0.5	-9.58E-01	-9.68E-01	3.51E-02		0.5	1.34E+06	3.03E+05	2.65E+06
	1	-4.39E-01	-4.29E-01	1.15E-01		1	1.23E+08	6.72E+07	1.46E+08
f_6	0	**3.61E-03**	5.35E-07	1.88E-02	f_{18}	0	1.58E+02	1.57E+02	3.04E+01
	0.3	2.24E+00	1.68E+00	1.92E+00		0.3	5.07E+00	3.42E+00	4.68E+00
	0.5	7.15E+00	5.85E+00	4.56E+00		0.5	1.57E+01	1.26E+01	1.45E+01
	1	1.31E+02	1.30E+02	4.24E+01		1	2.20E+07	7.89E+04	6.89E+07
f_7	0	**2.18E-01**	2.17E-01	4.16E-02	f_{19}	0	**2.50E-02**	3.32E-03	1.21E-01
	0.3	7.69E+00	5.71E+00	7.40E+00		0.3	2.56E-02	4.38E-28	1.30E-01
	0.5	2.16E+01	1.69E+01	1.65E+01		0.5	4.84E-02	2.01E-07	1.65E-01
	1	3.83E+02	3.55E+02	1.19E+02		1	2.48E-01	3.53E-02	4.04E-01
f_8	0	**3.66E-01**	3.69E-01	4.80E-02	f_{20}	0	3.12E-02	2.83E-03	1.14E-01
	0.3	4.75E-01	4.07E-01	1.66E-01		0.3	5.08E-02	0.00E+00	3.75E-01
	0.5	7.65E-01	4.42E-01	1.08E+00		0.5	**2.70E-02**	1.06E-18	1.59E-01
	1	3.61E+01	3.50E+01	9.47E+00		1	1.70E+00	6.61E-02	4.05E+00
f_9	0	**4.72E+00**	1.00E-18	4.67E+01	f_{21}	0	**4.13E-01**	3.98E-01	1.07E-01
	0.3	6.52E+05	3.05E+05	9.40E+05		0.3	4.23E-01	3.98E-01	2.31E-01
	0.5	7.12E+06	3.42E+06	1.18E+07		0.5	4.30E-01	3.98E-01	2.92E-01
	1	3.12E+08	2.69E+08	1.68E+08		1	5.39E-01	4.07E-01	3.22E-01
f_{10}	0	**5.10E-03**	1.06E-24	4.57E-02	f_{22}	0	**-9.79E-01**	-1.00E+00	1.41E-01
	0.3	4.12E-01	1.80E-01	5.09E-01		0.3	-9.20E-01	-1.00E+00	2.73E-01
	0.5	1.49E+00	1.15E+00	1.22E+00		0.5	-8.02E-01	-1.00E+00	3.98E-01
	1	3.96E+01	3.76E+01	1.34E+01		1	-3.02E-01	-2.67E-09	4.40E-01
f_{11}	0	**3.33E-02**	3.22E-02	1.11E-02	f_{23}	0	1.10E-02	3.29E-03	2.26E-02
	0.3	6.23E-02	2.87E-02	1.15E-01		0.3	**5.60E-04**	2.23E-51	4.83E-03
	0.5	2.07E-01	1.14E-01	2.61E-01		0.5	8.19E-04	6.68E-27	6.50E-03
	1	6.49E+00	6.03E+00	3.87E+00		1	2.38E-02	2.90E-03	5.78E-02
f_{12}	0	**1.46E+00**	1.07E+00	1.17E+00	f_{24}	0	**-1.87E+02**	-1.87E+02	7.04E-03
	0.3	6.52E+00	6.18E+00	2.43E+00		0.3	**-1.87E+02**	-1.87E+02	4.35E-03
	0.5	1.02E+01	1.01E+01	3.25E+00		0.5	**-1.87E+02**	-1.87E+02	4.40E-01
	1	1.59E+02	1.58E+02	2.15E+01		1	-1.79E+02	-1.87E+02	2.23E+01

Mutation rate	**0**	**0.3**	**0.5**	**1**
Best in	**20**	3	2	0
Friedman rank	**1.2917**	1.9167	2.7917	4
Friedman Prob	4.46E-013			

D. Population size:

Table 6: Results of Population Size Impact on the DE Algorithm

Function	Population	Mean	Median	STD	Function	Population	Mean	Median	STD
f_1	20	6.10E+00	6.08E+00	2.06E+00	f_{13}	20	8.12E+01	8.18E+01	3.20E+01
	30	2.43E+00	2.19E+00	1.44E+00		30	4.42E+01	3.66E+01	2.27E+01
	60	7.73E-02	6.72E-07	2.88E-01		60	1.96E+01	1.75E+01	8.70E+00
	100	**4.44E-15**	4.44E-15	0.00E+00		100	**1.68E+01**	1.68E+01	2.32E+00
f_2	20	1.59E-01	1.02E-01	1.78E-01	f_{14}	20	6.10E+02	4.50E+02	5.47E+02
	30	1.22E-02	1.97E-03	3.15E-02		30	7.75E+01	3.11E+01	1.20E+02
	60	**5.15E-05**	1.82E-13	1.88E-04		60	4.48E-01	6.42E-10	5.15E+00
	100	7.85E-04	6.94E-04	6.06E-04		100	**3.08E-06**	2.13E-48	5.34E-05
f_3	20	2.03E+03	1.63E+03	1.66E+03	f_{15}	20	6.91E+01	5.17E+01	6.01E+01
	30	2.89E+02	1.43E+02	3.93E+02		30	8.28E+00	3.65E+00	1.18E+01
	60	1.47E+00	4.02E-09	1.74E+01		60	1.59E-02	3.57E-11	9.22E-02
	100	**5.15E-05**	7.04E-48	8.91E-04		100	**1.70E-09**	1.90E-49	2.95E-08
f_4	20	3.56E+00	3.62E+00	1.08E+00	f_{16}	20	6.14E-02	9.95E-14	1.88E-01
	30	**2.57E+00**	2.66E+00	9.86E-01		30	3.50E-03	7.11E-14	1.91E-02
	60	3.41E+00	3.53E+00	7.55E-01		60	2.62E-05	2.84E-14	4.40E-04
	100	3.77E+00	3.82E+00	4.66E-01		100	**9.14E-15**	7.11E-15	1.06E-14
f_5	20	-9.60E-01	-9.68E-01	3.21E-02	f_{17}	20	1.35E+06	1.94E+05	5.37E+06
	30	-9.94E-01	-9.98E-01	8.71E-03		30	7.40E+04	2.75E+03	3.00E+05
	60	**-1.00E+00**	-1.00E+00	2.29E-04		60	1.56E+02	3.62E+01	1.09E+03
	100	**-1.00E+00**	-1.00E+00	6.32E-11		100	**6.32E+01**	5.24E+01	5.08E+01
f_6	20	7.00E+00	6.03E+00	4.94E+00	f_{18}	20	1.52E+01	1.17E+01	1.25E+01
	30	1.41E+00	1.11E+00	1.27E+00		30	2.58E+00	1.05E+00	4.36E+00
	60	1.42E-02	3.81E-08	5.37E-02		60	8.59E-02	3.62E-02	2.02E-01
	100	**4.93E-05**	0.00E+00	6.03E-04		100	**4.81E-02**	3.69E-02	4.10E-02
f_7	20	2.14E+01	1.78E+01	1.52E+01	f_{19}	20	6.95E-02	8.56E-14	2.28E-01
	30	4.70E+00	3.27E+00	5.54E+00		30	1.10E-02	0.00E+00	6.87E-02
	60	2.79E-01	2.51E-01	3.06E-01		60	8.17E-05	0.00E+00	8.45E-04
	100	**2.25E-01**	2.27E-01	3.70E-02		100	**0.00E+00**	0.00E+00	0.00E+00
f_8	20	6.98E-01	4.18E-01	1.05E+00	f_{20}	20	4.83E-02	3.35E-16	3.58E-01
	30	4.34E-01	4.03E-01	1.23E-01		30	5.70E-04	0.00E+00	6.56E-03
	60	3.87E-01	3.85E-01	6.29E-02		60	**0.00E+00**	0.00E+00	0.00E+00
	100	**3.69E-01**	3.64E-01	5.71E-02		100	**0.00E+00**	0.00E+00	0.00E+00
f_9	20	8.04E+06	2.89E+06	1.53E+07	f_{21}	20	4.38E-01	3.98E-01	2.76E-01
	30	1.23E+06	2.32E+05	4.01E+06		30	**3.98E-01**	3.98E-01	3.57E-04
	60	3.72E+03	3.53E-03	2.41E+04		60	**3.98E-01**	3.98E-01	8.34E-16
	100	**6.92E+01**	7.24E-45	7.95E+02		100	**3.98E-01**	3.98E-01	8.34E-16
f_{10}	20	1.31E+00	1.01E+00	1.15E+00	f_{22}	20	-8.13E-01	-1.00E+00	3.88E-01
	30	2.65E-01	1.10E-01	3.57E-01		30	-9.63E-01	-1.00E+00	1.88E-01
	60	8.31E-03	2.19E-15	5.86E-02		60	**-1.00E+00**	-1.00E+00	0.00E+00
	100	**5.00E-17**	1.50E-32	6.63E-16		100	**-1.00E+00**	-1.00E+00	0.00E+00
f_{11}	20	2.20E-01	1.27E-01	3.15E-01	f_{23}	20	3.05E-03	2.70E-22	2.99E-02
	30	2.85E-02	1.39E-02	4.47E-02		30	8.54E-05	1.49E-121	6.47E-04
	60	4.20E-03	4.02E-03	1.51E-03		60	7.70E-90	1.23E-128	1.33E-88
	100	**3.16E-03**	3.14E-03	9.52E-04		100	**2.41E-126**	6.06E-130	2.75E-125
f_{12}	20	1.08E+01	1.01E+01	4.18E+00	f_{24}	20	**-1.87E+02**	-1.87E+02	2.22E-02
	30	4.65E+00	4.22E+00	2.15E+00		30	**-1.87E+02**	-1.87E+02	1.04E-05
	60	**2.24E+00**	9.95E-01	3.83E+00		60	**-1.87E+02**	-1.87E+02	6.09E-14
	100	2.95E+00	3.00E-05	4.82E+00		100	**-1.87E+02**	-1.87E+02	3.56E-14

Population	**20**	**30**	**60**	**100**
Best in	1	3	7	**21**
Friedman rank	3.8958	2.8542	1.8750	**1.3750**
Friedman Prob	1.72E-1			

In the third test, we focus on the effect of population size on DE algorithm performance, we conducted the test for each population size NP = {20, 30, 60, 100} on 24 benchmark functions, with iterations set to GEN = 1000, mutation rate CR = 0.5, F = {0.2}, and problem dimension D = 20.

According to the results presented in table 6, we can see that according to both the number of best cases and the Friedman test it is evident that the best population size in this case is 100 as it had the smallest mean rank of 1.3750 and the highest number of best cases. Increasing the population size in the Differential Evolution (DE) algorithm improves solution quality by enhancing diversity and genetic variation, allowing better exploration and exploitation of the search space.

However, excessively large populations can increase computational cost without proportional gains in solution quality. Additionally, when choosing the population size, we need to be careful in choosing the scaling factor knowing that its effect is very sensitive to the scaling factor. To explore this interaction, we will later investigate the relationship between the population and the scaling factor.

E. Scaling factor:

Our fourth test focuses on the effect of scaling factor on the DE algorithm performance. We conduct tests for each of the scaling factor F = {0.1, 0.5, 0.9, 1} on 24 benchmark functions, with the number of iterations set to GEN=1000, scaling factor population size NP = 20, mutation rate CR = 0.5, and problem dimension D=20.

The results of this test presented in Table 6 supports our conclusion from this scaling factor comparison. After analyzing these results, we found that a scaling factor value of 0.5 produced the best results for most benchmark functions.

Table 7: Impact of scaling factor on the DE Algorithm Results.

Function	Scaling Factor	Mean	Median	STD
f_1	0.1	1.08E+01	1.09E+01	1.45E+00
	0.3	2.72E+00	2.32E+00	2.04E+00
	0.5	**1.17E-12**	9.48E-13	9.21E-13
	0.9	5.11E-01	4.23E-01	2.91E-01
	1	3.82E+00	3.76E+00	4.77E-01
f_2	0.1	9.85E-01	8.35E-01	6.52E-01
	0.3	1.73E-02	2.47E-03	5.26E-02
	0.5	**7.15E-04**	4.24E-06	1.51E-03
	0.9	8.27E+00	8.25E+00	1.71E+00
	1	9.68E+00	9.78E+00	1.63E+00
f_3	0.1	1.51E+06	5.86E+03	5.99E+06
	0.3	3.47E+02	1.58E+02	5.40E+02
	0.5	**5.69E-23**	2.11E-23	1.39E-22
	0.9	6.39E-01	5.35E-01	4.83E-01
	1	3.90E+01	3.35E+01	2.06E+01
f_4	0.1	5.27E+00	5.40E+00	9.80E-01
	0.3	**3.66E+00**	3.68E+00	1.14E+00
	0.5	5.83E+00	6.18E+00	9.38E-01
	0.9	6.61E+00	6.73E+00	6.69E-01
	1	4.97E+00	5.01E+00	8.54E-01
f_5	0.1	-8.65E-01	-8.77E-01	6.18E-02
	0.3	-9.93E-01	-9.97E-01	1.17E-02
	0.5	**-1.00E+00**	-1.00E+00	7.42E-14
	0.9	**-1.00E+00**	-1.00E+00	1.29E-05
	1	-9.98E-01	-9.98E-01	7.26E-04
f_6	0.1	2.44E+01	2.30E+01	1.26E+01
	0.3	1.70E+00	1.02E+00	2.00E+00
	0.5	**1.58E-03**	0.00E+00	4.90E-03
	0.9	8.12E-01	8.17E-01	9.39E-02
	1	1.28E+00	1.26E+00	1.42E-01
f_7	0.1	7.07E+01	6.68E+01	2.74E+01
	0.3	4.16E+00	2.77E+00	4.31E+00
	0.5	**3.89E-01**	3.99E-01	6.71E-02
	0.9	7.52E-01	7.54E-01	1.11E-01
	1	2.68E+00	2.56E+00	8.33E-01
f_8	0.1	4.69E+00	4.37E+00	4.04E+00
	0.3	4.41E-01	3.87E-01	1.33E-01
	0.5	**4.02E-01**	3.69E-01	1.13E-01
	0.9	4.18E-01	4.11E-01	8.92E-02
	1	4.77E-01	4.60E-01	1.23E-01
f_9	0.1	5.13E+07	2.77E+07	5.56E+07
	0.3	1.38E+06	5.46E+05	2.63E+06
	0.5	**6.58E+02**	9.55E-21	4.53E+03
	0.9	3.29E+04	4.64E+01	1.95E+05
	1	9.09E+05	1.94E+05	2.13E+06
f_{10}	0.1	5.10E+00	4.83E+00	2.18E+00
	0.3	4.52E-01	1.72E-01	5.96E-01
	0.5	**2.64E-02**	4.39E-25	1.83E-01
	0.9	3.04E-01	2.77E-01	1.82E-01
	1	3.60E+00	3.49E+00	1.45E+00
f_{11}	0.1	8.30E-01	6.64E-01	6.56E-01
	0.3	4.85E-02	1.68E-02	7.28E-02
	0.5	**9.15E-03**	8.96E-03	3.23E-03
	0.9	8.33E-02	8.39E-02	2.43E-02
	1	1.80E-01	1.74E-01	6.19E-02
f_{12}	0.1	2.27E+01	2.17E+01	6.87E+00
	0.3	**7.64E+00**	6.11E+00	5.25E+00
	0.5	5.04E+01	5.01E+01	7.24E+00
	0.9	7.68E+01	7.71E+01	1.00E+01
	1	5.77E+01	5.72E+01	9.31E+00

Function	Scaling Factor	Mean	Median	STD
f_{13}	0.1	1.49E+02	1.39E+02	5.64E+01
	0.3	3.98E+01	3.05E+01	2.12E+01
	0.5	**1.64E+01**	1.63E+01	2.01E+00
	0.9	2.79E+01	2.50E+01	1.03E+01
	1	1.00E+02	9.32E+01	3.50E+01
f_{14}	0.1	2.25E+03	2.05E+03	1.13E+03
	0.3	1.23E+02	5.44E+01	1.92E+02
	0.5	**2.52E-23**	5.06E-24	8.50E-23
	0.9	4.27E-01	3.72E-01	2.30E-01
	1	3.08E+01	2.76E+01	1.50E+01
f_{15}	0.1	2.24E+02	2.23E+02	9.22E+01
	0.3	1.05E+01	3.40E+00	1.49E+01
	0.5	**4.50E-24**	7.11E-25	3.32E-23
	0.9	2.98E-02	2.74E-02	1.60E-02
	1	3.87E+00	1.88E+00	1.39E+01
f_{16}	0.1	7.91E-01	2.84E-01	1.13E+00
	0.3	2.89E-04	6.75E-14	2.89E-03
	0.5	4.18E-14	4.26E-14	2.15E-14
	0.9	1.29E-14	1.42E-14	1.23E-14
	1	**8.67E-15**	0.00E+00	1.11E-14
f_{17}	0.1	7.26E+06	2.98E+06	1.09E+07
	0.3	1.17E+05	2.18E+03	5.79E+05
	0.5	**2.06E+02**	2.22E+02	6.34E+01
	0.9	3.64E+02	3.61E+02	2.11E+01
	1	1.91E+04	9.25E+03	3.20E+04
f_{18}	0.1	4.14E+01	3.58E+01	2.37E+01
	0.3	**2.85E+00**	9.60E-01	5.54E+00
	0.5	1.31E+01	1.19E+01	6.22E+00
	0.9	1.53E+02	1.54E+02	2.51E+01
	1	1.97E+02	1.98E+02	3.55E+01
f_{19}	0.1	1.54E-01	7.75E-03	4.14E-01
	0.3	2.75E-02	0.00E+00	1.33E-01
	0.5	**0.00E+00**	0.00E+00	0.00E+00
	0.9	**0.00E+00**	0.00E+00	0.00E+00
	1	6.02E-32	0.00E+00	4.23E-31
f_{20}	0.1	7.15E-01	1.47E-03	1.92E+00
	0.3	2.76E-02	0.00E+00	2.67E-01
	0.5	**0.00E+00**	0.00E+00	0.00E+00
	0.9	**0.00E+00**	0.00E+00	0.00E+00
	1	9.47E-32	0.00E+00	4.09E-31
f_{21}	0.1	4.37E-01	3.98E-01	2.40E-01
	0.3	4.09E-01	3.98E-01	6.68E-02
	0.5	4.24E-01	3.98E-01	2.32E-01
	0.9	**4.01E-01**	3.98E-01	3.03E-02
	1	4.06E-01	3.98E-01	3.76E-02
f_{22}	0.1	-2.98E-01	-8.08E-05	4.36E-01
	0.3	-9.60E-01	-1.00E+00	1.97E-01
	0.5	**-1.00E+00**	-1.00E+00	0.00E+00
	0.9	**-1.00E+00**	-1.00E+00	0.00E+00
	1	-5.84E-01	-9.76E-01	4.77E-01
f_{23}	0.1	3.04E-02	1.81E-04	8.51E-02
	0.3	3.20E-05	1.09E-113	3.20E-04
	0.5	**8.77E-91**	5.31E-96	8.49E-90
	0.9	1.83E-60	2.05E-62	6.28E-60
	1	5.72E-33	0.00E+00	2.54E-32
f_{24}	0.1	**-1.87E+02**	-1.87E+02	1.54E+00
	0.3	**-1.87E+02**	-1.87E+02	1.70E-02
	0.5	**-1.87E+02**	-1.87E+02	1.77E-02
	0.9	**-1.87E+02**	-1.87E+02	2.15E-02
	1	**-1.86E+02**	-1.87E+02	1.38E+00

Scaling factor	**0.1**	**0.3**	**0.5**	**0.9**	**1**
Best in	1	4	**19**	6	2
Friedman rank	4.5208	3.1458	**1.6042**	2.3542	3.3750
Friedman prob	1.09E-09				

F. F and NP relationship:

In order to investigate the nature of the relationship between the population size and the scaling factor, we compared the results of three values of the scaling factor F = {0.2, 0.5, 0.9} for each of the following population {20, 30, 60, 100} and for 24 functions. The results of the comparison are presented in table 8.

When the scaling factor was set to 0.2, which promotes conservative exploration, increasing the population size to 100 provided significant improvements in the quality of solutions. This outcome indicates that with a lower scaling factor, broader exploration is necessary, and a larger population ensures sufficient diversity to avoid premature convergence and thoroughly search the solution space. A smaller population size in this context would limit diversity, reducing the algorithm's ability to find optimal solutions.

At a moderate scaling factor of 0.5, the algorithm achieves a balance between exploration and exploitation. Here, a population size of 60 was found to be optimal. This balance allows the algorithm to make steady progress without overwhelming the search process with either excessive exploration or premature exploitation. Increasing the population size beyond this point may introduce unnecessary redundancy, while a smaller population size may not provide enough diversity to explore effectively.

When the scaling factor was set to 0.9, emphasizing aggressive exploration, a smaller population size of 20 was optimal. In this scenario, the large differential variations push the search into new areas aggressively, and a smaller population allows for quicker convergence once promising regions are identified. Increasing the population size in this case led to inefficiencies and redundancy, as the high scaling factor already provides ample exploration. A larger population would slow down the convergence process, resulting in less efficient refinement of solutions.

Table 8: F and NP relationship results

		F=0.2	F=0.5	F=0.9
Function	**NP**	**Mean**		
f_1	20	6.10E+00	6.21E-04	**4.62E-01**
	30	2.43E+00	**6.84E-12**	8.12E-01
	60	7.73E-02	4.31E-11	1.24E+00
	100	**4.44E-15**	8.02E-11	1.39E+00
f_2	20	1.59E-01	**5.77E-04**	**8.30E+00**
	30	1.22E-02	5.55E-03	8.68E+00
	60	**5.15E-05**	1.00E-02	9.02E+00
	100	7.85E-04	1.08E-02	9.03E+00
f_3	20	2.03E+03	2.45E-06	**6.16E-01**
	30	2.89E+02	**1.54E-21**	1.30E+00
	60	1.47E+00	4.78E-20	2.05E+00
	100	**5.15E-05**	1.67E-19	2.25E+00
f_4	20	3.56E+00	**5.84E+00**	**6.48E+00**
	30	**2.57E+00**	6.25E+00	7.03E+00
	60	3.41E+00	6.51E+00	7.21E+00
	100	3.77E+00	6.57E+00	7.25E+00
f_5	20	-9.60E-01	**-1.00E+00**	**-1.00E+00**
	30	-9.94E-01	**-1.00E+00**	**-1.00E+00**
	60	**-1.00E+00**	**-1.00E+00**	**-1.00E+00**
	100	**-1.00E+00**	**-1.00E+00**	**-1.00E+00**
f_6	20	7.00E+00	1.57E-03	**8.09E-01**
	30	1.41E+00	2.51E-04	9.08E-01
	60	1.42E-02	7.40E-06	9.33E-01
	100	**4.93E-05**	**5.12E-12**	9.59E-01
f_7	20	2.14E+01	3.89E-01	7.75E-01
	30	4.70E+00	3.81E-01	7.49E-01
	60	2.79E-01	3.66E-01	7.38E-01
	100	**2.25E-01**	**3.45E-01**	**7.01E-01**
f_8	20	6.98E-01	3.88E-01	4.13E-01
	30	4.34E-01	3.48E-01	3.80E-01
	60	3.87E-01	2.88E-01	3.58E-01
	100	**3.69E-01**	**2.66E-01**	**3.21E-01**
f_9	20	8.04E+06	6.26E+03	2.86E+04
	30	1.23E+06	**6.11E-19**	**8.83E+01**
	60	3.72E+03	1.59E-17	1.40E+02
	100	**6.92E+01**	5.38E-17	1.59E+02
f_{10}	20	1.31E+00	2.56E-02	**3.39E-01**
	30	2.65E-01	**4.12E-23**	4.71E-01
	60	8.31E-03	1.45E-21	6.13E-01
	100	**5.00E-17**	5.14E-21	6.56E-01
f_{11}	20	2.20E-01	9.96E-03	8.76E-02
	30	2.85E-02	9.22E-03	8.83E-02
	60	4.20E-03	8.51E-03	8.04E-02
	100	**3.16E-03**	**7.88E-03**	**7.39E-02**
f_{12}	20	1.08E+01	**5.11E+01**	7.98E+01
	30	4.65E+00	5.42E+01	8.01E+01
	60	**2.24E+00**	5.39E+01	7.94E+01
	100	2.95E+00	5.18E+01	**7.74E+01**

		F=0.2	F=0.5	F=0.9
Function	**NP**	**Mean**		
f_{13}	20	8.12E+01	1.63E+01	**2.78E+01**
	30	4.42E+01	1.52E+01	2.90E+01
	60	1.96E+01	1.35E+01	3.39E+01
	100	**1.68E+01**	**1.26E+01**	3.60E+01
f_{14}	20	6.10E+02	7.08E-05	**4.17E-01**
	30	7.75E+01	**5.38E-22**	8.27E-01
	60	4.48E-01	1.55E-20	1.35E+00
	100	**3.08E-06**	5.51E-20	1.47E+00
f_{15}	20	6.91E+01	1.76E-03	**2.62E-02**
	30	8.28E+00	**3.89E-23**	5.16E-02
	60	1.59E-02	1.39E-21	8.35E-02
	100	**1.70E-09**	4.78E-21	9.33E-02
f_{16}	20	6.14E-02	4.51E-14	1.49E-14
	30	3.50E-03	2.38E-14	3.27E-15
	60	2.62E-05	3.18E-15	**0.00E+00**
	100	**9.14E-15**	**2.13E-16**	**0.00E+00**
f_{17}	20	1.35E+06	2.03E+02	**3.66E+02**
	30	7.40E+04	2.02E+02	3.86E+02
	60	1.56E+02	**1.99E+02**	4.05E+02
	100	**6.32E+01**	**1.99E+02**	4.05E+02
f_{18}	20	1.52E+01	**1.26E+01**	1.50E+02
	30	2.58E+00	1.70E+01	1.51E+02
	60	8.59E-02	2.01E+01	1.47E+02
	100	**4.81E-02**	2.07E+01	**1.40E+02**
f_{19}	20	6.95E-02	3.33E-03	**0.00E+00**
	30	1.10E-02	**0.00E+00**	**0.00E+00**
	60	8.17E-05	**0.00E+00**	**0.00E+00**
	100	**0.00E+00**	**0.00E+00**	**0.00E+00**
f_{20}	20	4.83E-02	5.31E-31	**0.00E+00**
	30	5.70E-04	**0.00E+00**	**0.00E+00**
	60	**0.00E+00**	**0.00E+00**	**0.00E+00**
	100	**0.00E+00**	**0.00E+00**	**0.00E+00**
f_{21}	20	4.38E-01	4.29E-01	4.23E-01
	30	**3.98E-01**	3.99E-01	**3.98E-01**
	60	**3.98E-01**	**3.98E-01**	**3.98E-01**
	100	**3.98E-01**	**3.98E-01**	**3.98E-01**
f_{22}	20	-8.13E-01	**-1.00E+00**	**-1.00E+00**
	30	-9.63E-01	**-1.00E+00**	**-1.00E+00**
	60	**-1.00E+00**	**-1.00E+00**	**-1.00E+00**
	100	**-1.00E+00**	**-1.00E+00**	**-1.00E+00**
f_{23}	20	3.05E-03	3.75E-83	3.55E-99
	30	8.54E-05	1.94E-156	3.94E-99
	60	7.70E-90	6.22E-159	5.50E-101
	100	**2.41E-126**	**3.37E-160**	**1.58E-101**
f_{24}	20	**-1.87E+02**	**-1.87E+02**	**-1.87E+02**
	30	**-1.87E+02**	**-1.87E+02**	**-1.87E+02**
	60	**-1.87E+02**	**-1.87E+02**	**-1.87E+02**
	100	**-1.87E+02**	**-1.87E+02**	**-1.87E+02**

Scaling factor	**0.2**	**0.5**	**0.9**
Best Population size	100	60	20
Friedman rank	2.0938	**1.7396**	2.1667
Friedman Prob	0.0013		

IV.4.2. DE variants Comparison:

The results of our crossover methods comparison presented in Table 9 shows that the binomial method (bin) performed better in most cases. Therefore, we chose it as our crossover method in

the DE variants comparison study, comparing the following DE variants: Rand/1/bin, Rand/2/bin, Best/1/bin, Best/2/bin, and Rand-to-best/1/bin.

Table 9: Crossover methods comparison (binomial vs exponential)

Fun	dim	Rand/1/bin			Rand/1/exp		
		Mean	Median	STD	Mean	Median	STD
f_1	10	**4.12E-15**	4.44E-15	1.02E-15	4.33E-15	4.44E-15	6.09E-16
	30	**1.51E-14**	1.51E-14	4.34E-15	1.06E-05	1.05E-05	2.44E-06
	50	**4.74E-07**	4.69E-07	1.19E-07	7.95E-03	7.94E-03	1.10E-03
f_2	10	**1.65E-17**	2.18E-58	1.65E-16	9.81E-13	1.42E-14	4.51E-12
	30	1.73E-02	1.72E-02	2.43E-03	**6.20E-04**	5.97E-04	1.87E-04
	50	5.05E-02	5.06E-02	2.52E-03	**2.09E-02**	2.10E-02	2.08E-03
f_3	10	**4.70E-98**	9.75E-99	1.59E-97	5.56E-38	2.49E-38	1.18E-37
	30	**3.98E-27**	2.48E-27	5.54E-27	1.69E-09	1.61E-09	6.99E-10
	50	**7.23E-12**	6.27E-12	3.93E-12	1.13E-03	1.07E-03	3.37E-04
f_4	10	2.58E-01	2.62E-01	1.86E-01	**1.91E-01**	1.93E-01	9.17E-02
	30	1.10E+01	1.10E+01	4.88E-01	**1.95E+00**	1.96E+00	1.34E-01
	50	2.17E+01	2.17E+01	3.57E-01	**7.94E+00**	8.06E+00	5.97E-01
f_5	10	**-1.00E+00**	-1.00E+00	0.00E+00	**-1.00E+00**	-1.00E+00	0.00E+00
	30	**-1.00E+00**	-1.00E+00	3.16E-17	**-1.00E+00**	-1.00E+00	1.42E-14
	50	**-1.00E+00**	-1.00E+00	2.62E-16	**-1.00E+00**	-1.00E+00	7.03E-09
f_6	10	1.23E-04	0.00E+00	1.23E-03	**2.44E-17**	0.00E+00	2.33E-16
	30	7.40E-05	0.00E+00	7.40E-04	**2.17E-05**	8.88E-08	7.52E-05
	50	**1.19E-11**	6.90E-12	2.13E-11	4.24E-03	2.12E-03	5.15E-03
f_7	10	**1.65E-01**	1.64E-01	3.19E-02	2.20E-01	2.15E-01	3.88E-02
	30	4.39E-01	4.46E-01	5.21E-02	**4.09E-01**	4.15E-01	5.25E-02
	50	6.32E-01	6.39E-01	6.75E-02	**4.82E-01**	4.90E-01	4.58E-02
f_8	10	**1.71E-01**	1.66E-01	4.29E-02	2.43E-01	2.36E-01	5.54E-02
	30	3.60E-01	3.18E-01	1.31E-01	**3.58E-01**	3.59E-01	3.44E-02
	50	4.50E-01	3.66E-01	2.17E-01	**3.96E-01**	3.96E-01	2.75E-02
f_9	10	**9.33E-96**	2.26E-96	1.75E-95	1.47E-35	7.82E-36	2.23E-35
	30	**1.70E-24**	1.14E-24	1.74E-24	4.29E-06	3.45E-06	2.59E-06
	50	**2.70E-09**	2.34E-09	1.61E-09	9.82E+00	8.83E+00	4.32E+00
f_{10}	10	**1.50E-32**	1.50E-32	8.25E-48	**1.50E-32**	1.50E-32	8.25E-48
	30	**4.22E-28**	2.49E-28	5.41E-28	6.82E-12	6.24E-12	3.02E-12
	50	**4.91E-12**	3.55E-12	3.88E-12	6.65E-06	6.45E-06	1.69E-06
f_{11}	10	**1.07E-03**	1.05E-03	4.30E-04	2.81E-03	2.63E-03	1.11E-03
	30	**1.05E-02**	1.05E-02	2.55E-03	5.04E-02	5.15E-02	1.22E-02
	50	**3.62E-02**	3.64E-02	7.37E-03	1.85E-01	1.87E-01	3.42E-02
f_{12}	10	**0.00E+00**	0.00E+00	0.00E+00	**0.00E+00**	0.00E+00	0.00E+00
	30	1.18E+02	1.17E+02	8.22E+00	**2.07E-06**	1.63E-06	1.39E-06
	50	3.01E+02	3.03E+02	1.31E+01	**2.33E+00**	2.13E+00	1.14E+00
f_{13}	10	**2.69E+00**	2.75E+00	8.31E-01	3.51E+00	3.48E+00	7.67E-01
	30	**2.36E+01**	2.38E+01	6.95E-01	2.50E+01	2.50E+01	7.84E-01
	50	**4.45E+01**	4.46E+01	8.38E-01	4.54E+01	4.55E+01	7.29E-01
f_{14}	10	**8.30E-99**	1.27E-99	2.96E-98	5.74E-39	3.19E-39	1.02E-38
	30	**2.25E-27**	1.69E-27	2.17E-27	7.26E-10	6.78E-10	3.30E-10
	50	**5.01E-12**	4.29E-12	4.02E-12	6.28E-04	6.04E-04	1.81E-04
f_{15}	10	**2.26E-100**	5.92E-101	5.64E-100	2.26E-40	1.27E-40	3.18E-40
	30	**1.98E-28**	1.42E-28	1.78E-28	9.12E-11	8.23E-11	4.25E-11
	50	**9.59E-13**	8.02E-13	5.44E-13	1.38E-04	1.40E-04	3.20E-05
f_{16}	10	4.23E-15	0.00E+00	7.22E-15	**1.63E-15**	0.00E+00	4.12E-15
	30	2.18E-14	2.13E-14	1.29E-14	**3.41E-15**	0.00E+00	6.43E-15
	50	6.85E-07	1.14E-13	6.85E-06	**4.69E-15**	0.00E+00	7.01E-15
f_{17}	10	1.61E+01	1.74E+01	1.05E+01	**7.86E-02**	6.82E-12	3.58E-01
	30	4.32E+02	4.21E+02	3.78E+01	**4.93E+01**	7.88E+00	8.53E+01
	50	1.86E+03	1.86E+03	6.77E+01	**1.64E+03**	1.74E+03	3.08E+02
f_{18}	10	**6.10E-24**	1.93E-24	1.06E-23	1.50E-01	1.23E-01	1.11E-01
	30	**6.10E+01**	5.92E+01	1.31E+01	1.91E+02	1.97E+02	2.74E+01
	50	**4.13E+02**	4.09E+02	3.75E+01	5.12E+02	5.14E+02	4.23E+01
f_{19}	2	**0.00E+00**	0.00E+00	0.00E+00	4.09E-04	1.45E-04	7.51E-04
f_{20}	2	**0.00E+00**	0.00E+00	0.00E+00	9.60E-17	1.72E-18	4.11E-16
f_{21}	2	**3.98E-01**	3.98E-01	0.00E+00	**3.98E-01**	3.98E-01	5.03E-10
f_{22}	2	**-1.00E+00**	-1.00E+00	0.00E+00	**-1.00E+00**	-1.00E+00	1.15E-10
f_{23}	2	**1.31E-318**	4.90E-324	0.00E+00	4.40E-05	2.36E-05	5.48E-05
f_{24}	2	-1.87E+02	-1.87E+02	3.18E-14	**-1.87E+02**	-1.87E+02	2.54E-14
Friedman rank		**1.3833**			1.6167		
Best in		**40**			24		
Friedman Prob		0.0522					

A. *Convergence quality test:*

Table 10: Comparative results of the convergence quality test.

fun	Dim	Rand/1		Rand/2		Best/1		Best/2		Rand-to-Best/2	
		Mean	STD	Mean	STD	Mean	STD	Mean	STD	Mean	STD
f_1	10	**3.87E-15**	1.31E-15	4.33E-15	6.09E-16	1.25E-01	3.83E-01	4.94E-15	1.24E-15	1.16E-02	1.16E-02
	30	1.70E-14	5.83E-15	5.35E-04	1.40E-04	2.06E+00	1.16E+00	**1.46E-14**	3.91E-15	9.51E-01	9.51E-01
	50	**4.83E-07**	1.32E-07	3.52E+00	1.90E-01	4.25E+00	1.46E+00	1.32E-06	7.18E-07	3.32E+00	3.32E+00
f_2	10	**3.97E-23**	3.97E-22	7.61E-04	3.02E-04	1.38E-15	1.97E-15	3.11E-17	2.06E-16	1.72E-17	1.72E-17
	30	1.77E-02	2.17E-03	1.13E+01	1.65E+00	**2.25E-14**	4.17E-14	1.90E-01	1.39E+00	1.82E-04	1.82E-04
	50	5.08E-02	2.56E-03	4.41E+01	2.68E+00	**1.76E-10**	1.30E-09	1.01E+01	1.02E+01	5.17E-02	5.17E-02
f_3	10	4.11E-98	1.03E-97	4.08E-57	8.70E-57	**5.76E-276**	0.00E+00	3.51E-128	1.51E-127	1.81E-243	1.81E-243
	30	4.33E-27	7.14E-27	3.42E-06	1.79E-06	**5.11E-99**	1.79E-98	6.73E-30	9.56E-30	3.25E+00	3.25E+00
	50	**7.65E-12**	5.23E-12	6.47E+01	1.54E+01	1.00E+02	1.00E+03	3.57E-11	3.91E-11	2.87E+02	2.87E+02
f_4	10	**2.93E-01**	1.86E-01	1.43E+00	2.60E-01	1.59E+00	7.68E-01	1.10E+00	6.16E-01	7.07E-01	7.07E-01
	30	1.09E+01	5.55E-01	1.23E+01	2.19E-01	8.94E+00	1.35E+00	1.12E+01	4.12E-01	**8.01E+00**	8.01E+00
	50	2.16E+01	3.26E-01	2.23E+01	1.91E-01	**1.73E+01**	1.38E+00	2.16E+01	3.45E-01	1.93E+01	1.93E+01
f_5	10	**-1.00E+00**	0.00E+00	**-1.00E+00**	0.00E+00	**-1.00E+00**	8.57E-17	**-1.00E+00**	4.73E-17	**-1.00E+00**	-1.00E+00
	30	**-1.00E+00**	3.35E-17	**-1.00E+00**	5.75E-11	**-1.00E+00**	8.46E-16	**-1.00E+00**	6.31E-17	**-1.00E+00**	-1.00E+00
	50	**-1.00E+00**	2.62E-16	-9.97E-01	6.96E-04	**-1.00E+00**	5.54E-14	**-1.00E+00**	1.31E-15	-9.98E-01	-9.98E-01
f_6	10	**0.00E+00**	0.00E+00	9.37E-02	3.48E-02	6.41E-02	3.69E-02	1.97E-02	1.82E-02	2.73E-02	2.73E-02
	30	**3.20E-04**	1.59E-03	3.75E-02	5.38E-02	3.17E-02	9.71E-02	2.07E-03	4.94E-03	5.68E-02	5.68E-02
	50	**1.30E-11**	4.55E-11	1.48E+00	1.13E-01	6.09E-02	1.44E-01	1.06E-03	2.96E-03	1.29E+00	1.29E+00
f_7	10	1.60E-01	2.85E-02	2.06E-01	3.85E-02	1.47E-01	4.17E-02	1.66E-01	3.56E-02	**1.15E-01**	1.15E-01
	30	4.45E-01	4.74E-02	6.13E-01	6.91E-02	4.86E-01	1.33E-01	4.78E-01	7.46E-02	**3.98E-01**	3.98E-01
	50	6.36E-01	5.83E-02	2.48E+00	6.93E-01	**5.94E-01**	1.30E-01	6.80E-01	8.96E-02	1.62E+00	1.62E+00
f_8	10	1.63E-01	4.36E-02	1.69E-01	3.38E-02	2.00E-01	8.17E-02	**1.19E-01**	3.71E-02	2.61E-01	2.61E-01
	30	3.77E-01	1.51E-01	**3.53E-01**	7.57E-02	4.97E-01	2.24E-01	4.98E-01	2.46E-01	4.56E-01	4.56E-01
	50	**4.92E-01**	2.58E-01	5.28E-01	2.06E-01	6.01E-01	2.79E-01	6.34E-01	3.33E-01	5.67E-01	5.67E-01
f_9	10	1.40E-95	4.18E-95	4.28E-55	1.18E-54	**1.13E-273**	0.00E+00	3.33E-125	2.07E-124	4.03E-238	4.03E-238
	30	**1.50E-24**	1.33E-24	4.09E-04	2.03E-04	4.50E+04	1.66E+05	4.18E+02	4.18E+03	1.12E+04	1.12E+04
	50	**2.78E-09**	1.54E-09	4.94E+03	1.18E+03	3.03E+06	2.56E+07	2.95E+03	2.95E+04	4.92E+05	4.92E+05
f_{10}	10	**1.50E-32**	1.93E-47	**1.50E-32**	8.25E-48	9.70E-02	1.88E-01	4.54E-03	4.54E-02	8.95E-04	8.95E-04
	30	**4.79E-28**	6.31E-28	1.34E-05	1.30E-05	4.12E+00	2.59E+00	2.39E-01	4.40E-01	7.46E-01	7.46E-01
	50	**5.69E-12**	5.72E-12	2.07E+01	5.15E+00	1.20E+01	5.33E+00	1.02E+00	1.22E+00	2.98E+00	2.98E+00
f_{11}	10	1.09E-03	4.22E-04	1.89E-03	7.21E-04	6.14E-04	3.90E-04	9.04E-04	4.00E-04	**3.41E-04**	3.41E-04
	30	1.14E-02	2.71E-03	4.35E-02	9.72E-03	9.48E-03	4.26E-03	1.15E-02	3.03E-03	**7.87E-03**	7.87E-03
	50	3.59E-02	7.74E-03	4.47E-01	8.33E-02	6.12E-02	3.01E-02	4.62E-02	**1.08E-02**	6.22E-02	6.22E-02
f_{12}	10	**0.00E+00**	0.00E+00	4.26E-01	9.54E-01	4.09E+00	2.51E+00	9.95E-02	3.32E-01	7.76E-01	7.76E-01
	30	1.16E+02	9.32E+00	1.55E+02	8.95E+00	3.75E+01	1.17E+01	1.33E+02	1.04E+01	**1.55E+01**	1.55E+01
	50	3.01E+02	1.40E+01	3.99E+02	1.63E+01	8.83E+01	2.13E+01	3.40E+02	1.67E+01	**3.86E+01**	3.86E+01
f_{13}	10	2.57E+00	8.40E-01	**7.51E-03**	4.28E-03	8.37E-01	1.63E+00	3.59E-01	1.15E+00	5.19E+00	5.19E+00
	30	2.36E+01	6.80E-01	2.43E+01	2.78E-01	**9.61E+00**	3.75E+00	1.68E+01	6.04E+00	3.00E+01	3.00E+01
	50	4.43E+01	8.02E-01	3.60E+02	7.27E+01	4.39E+01	2.87E+01	**4.29E+01**	1.12E+01	8.53E+01	8.53E+01
f_{14}	10	7.79E-99	3.50E-98	7.72E-58	1.70E-57	**8.57E-278**	0.00E+00	1.39E-128	5.11E-128	4.12E-245	4.12E-245
	30	2.25E-27	2.81E-27	2.38E-06	1.25E-06	**3.85E-99**	1.80E-98	2.68E-30	3.67E-30	3.73E-02	3.73E-02
	50	5.66E-12	3.89E-12	5.26E+01	1.47E+01	**3.46E-48**	1.44E-47	2.13E-11	2.39E-11	4.27E+01	4.27E+01
f_{15}	10	9.94E-100	7.37E-99	3.92E-59	9.24E-59	**1.92E-277**	0.00E+00	5.84E-130	3.43E-129	1.47E-246	1.47E-246
	30	2.44E-28	2.29E-28	2.11E-07	9.29E-08	**1.96E-99**	1.29E-98	3.95E-31	6.30E-31	1.01E-02	1.01E-02
	50	**9.83E-13**	6.47E-13	7.81E+00	1.90E+00	2.00E+00	1.41E+01	4.80E-12	5.93E-12	1.24E+01	1.24E+01
f_{16}	10	3.45E-15	7.08E-15	**7.46E-16**	3.00E-15	1.48E-03	1.48E-02	4.54E-14	1.89E-14	5.30E-14	5.30E-14
	30	2.38E-14	1.45E-14	**8.46E-15**	8.79E-15	2.31E-02	9.03E-02	1.25E-13	2.56E-14	8.69E-02	8.69E-02
	50	3.40E-04	3.40E-03	9.44E-02	1.35E-01	2.61E-01	6.72E-01	**2.68E-13**	4.11E-14	9.14E+00	9.14E+00
f_{17}	10	1.54E+01	1.00E+01	3.37E+01	4.95E+00	1.99E+01	1.41E+01	2.34E+01	1.09E+01	**1.50E+01**	1.50E+01
	30	4.33E+02	3.02E+01	6.89E+02	2.41E+01	**4.02E+02**	1.29E+02	5.38E+02	7.37E+01	4.36E+02	4.36E+02
	50	**1.86E+03**	5.96E+01	8.26E+06	5.66E+06	1.48E+03	5.33E+02	2.07E+03	2.28E+02	1.29E+04	1.29E+04
f_{18}	10	7.28E-24	1.73E-23	9.19E-12	1.73E-11	**2.19E-89**	1.44E-88	7.02E-33	4.13E-32	2.80E-74	2.80E-74
	30	6.10E+01	1.26E+01	1.87E+02	2.60E+01	1.60E+00	6.32E+00	3.56E+01	1.24E+01	**1.48E-03**	1.48E-03
	50	4.18E+02	4.58E+01	6.53E+02	5.33E+01	6.70E+01	4.09E+01	3.91E+02	5.69E+01	**1.10E+01**	1.10E+01
f_{19}	2	**0.00E+00**	0.00E+00	**0.00E+00**	0.00E+00	5.33E-02	1.95E-01	2.29E-02	1.31E-01	2.29E-02	2.29E-02
f_{20}	2	**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
f_{21}	2	**3.98E-01**	1.06E-15	**3.98E-01**	0.00E+00	4.67E-01	3.96E-01	4.21E-01	2.31E-01	**3.98E-01**	3.98E-01
f_{22}	2	**-1.00E+00**	0.00E+00	**-1.00E+00**	0.00E+00	-9.40E-01	2.39E-01	-9.50E-01	2.19E-01	**-1.00E+00**	-1.00E+00
f_{23}	2	1.93E-198	0.00E+00	2.60E-155	1.72E-154	**0.00E+00**	0.00E+00	1.92E-238	0.00E+00	1.61E-291	1.61E-291
f_{24}	2	**-1.87E+02**	4.54E-14	**-1.87E+02**	2.30E-14	**-1.87E+02**	3.78E-14	**-1.87E+02**	3.05E-14	**-1.87E+02**	-1.87E+02
Friedman rank		**2.3250**		3.8500		2.8667		2.8833		3.0750	
Best in		**25**		12		21		8		16	
Friedman Prob		2.20E-06									

The first test of our second study evaluates the convergence quality of each variant by assessing their solutions within a limited number of iterations (fixed-cost solution results), setting GEN = 2000. The population size was set to NP = 40, the scaling factor to F = 0.5, and the mutation rate to CR = 0.5. Each test is conducted for three dimension values, D = {10, 30, 50}, across scalable functions. To identify the best variant in terms of the convergence quality, we compare the quality of solutions produced by each variant.

The results presented in Table 12 shows that rand/1 performed the best, achieving the top solution in 25 out of 60 cases and having the best mean rank. It was followed closely by best/1, with 21 best cases and the second-best mean rank. Best/2 had the fewest best cases (8), even though it didn't have the worst mean rank, while rand/2 had the worst mean rank despite having 12 best cases. Based on these results, rand/1 appears to be the optimal choice for scenarios with limited iterations and constrained computing resources and time.

B. Convergence speed test:

The second test measures the convergence speed by determining the number of iterations required to achieve an acceptable solution (fixed-target cost results). The acceptable solution (the fixed target) is set to 1E-16 while the maximum number of iterations is set to 10000, as setting it to a bigger number is unpractical. If one of the variants algorithms converged within this number of iterations, the average number of iterations required to reach the acceptable solution was recorded. If, for a given function, no algorithm could converge to the solution, the convergence data for that function was omitted.

The test also calculates the success rate, it is calculated by counting the number of successful runs (those that reached the target within 10,000 iterations) and dividing that by the total number of runs (100 runs in this case). For any algorithm that did not reach an acceptable solution within 10,000 iterations in one of those 100 runs, the number of iterations for that run (the convergence speed) was taken as 10,000 when calculating the mean, and it was considered a failure when calculating the success rate.

In this test, we didn’t analyze the results using the Fridman test because of their nature, which are not compatible with the Friedman test, knowing that in the convergence speed test, we set a maximum limit of 10,000 iterations, making it impossible to determine the exact number of iterations required to reach an acceptable solution if it exceeded this limit. This lack of precise iteration counts prevents accurate ranking in the Friedman test.

The population size was set to NP = 40, the scaling factor to F = 0.5, and the mutation rate to CR = 0.5. Each test is conducted for three dimension values, D = {10, 30, 50}, across 20 functions that have 0 as its minimum value.

The results presented in Table 13 indicate that the best/1 variant had the superior performance, achieving the best results in 16 out of 42 cases. This was followed by rand/1 and best/2, which had the best results in 9 and 8 cases, respectively. Conversely, rand/2 and rand-to-best exhibited the poorest performance, each excelling in only 2 cases. Based on these results, the best/1 variant seems to be the most suitable choice for scenarios where there are unlimited iterations and sufficient computing resources and time.

Table 11: Comparative results of the convergence speed test.

fun	dim	Rand/1		Rand/2		Best/1		Best/2		Rand-to-Best/2	
		GEN	Success Rate	GEN	Success rate	GEN	Success rate	GEN	Success rate	GEN	Success Rate
f_2	10	1284.66	100%	8318.27	67%	7660.6	24%	**836.11**	98%	1071.78	92%
	30	9311.61	38%	10000	0%	10000	0%	**4611.42**	76%	10000	0%
	50	10000	0%	10000	0%	10000	0%	**9079.41**	29%	10000	0%
f_3	10	408.33	100%	676.27	100%	**152.92**	100%	317.63	100%	172.49	100%
	30	1333.53	100%	4014.68	100%	**503.21**	99%	1211.82	100%	10000	0%
	50	2590.8	100%	0000	0%	**972.54**	98%	2674.67	100%	10000	0%
f_4	10	**8430.97**	23%	10000	0%	10000	0%	9940.78	1%	10000	0%
	30	10000	0%	10000	0%	10000	0%	10000	0%	10000	0%
	50	10000	0%	10000	0%	10000	0%	10000	0%	10000	0%
f_6	10	**965.74**	100%	3743.1	100%	9901.7	1%	6267.23	42%	8829.07	12%
	30	**1337.24**	100%	4669.78	100%	9808.05	2%	2559.47	85%	10000	0%
	50	**2515.63**	100%	10000	0%	10000	0%	3793.79	84%	10000	0%
f_9	10	452.14	100%	743.84	100%	465.83	97%	349.96	100%	**194.75**	100%
	30	1496.5	100%	4426.03	100%	1916.06	85%	**1385.58**	100%	10000	0%
	50	**2907.18**	100%	10000	0%	3764.41	69%	3073.39	100%	10000	0%
f_{10}	10	359.53	100%	608.56	100%	2400.97	77%	381.12	99%	**147.63**	100%
	30	**1273.08**	100%	4103.09	100%	10000	0%	3765.64	71%	10000	0%
	50	**2553.26**	100%	10000	0%	10000	0%	7991.54	28%	10000	0%
f_{12}	10	**1041.97**	100%	2345.72	100%	9806.24	2%	2029.31	89%	6052.88	41%
	30	10000	0%	10000	0%	10000	0%	10000	0%	10000	0%
	50	10000	0%	10000	0%	10000	0%	10000	0%	10000	0%
f_{13}	10	10000	0%	6738.73	100%	**3170.5**	81%	3483.83	94%	10000	0%
	30	10000	0%	10000	0%	**7091.35**	78%	9995.08	1%	10000	0%
	50	10000	0%	10000	0%	**9920.56**	7%	10000	0%	10000	0%
f_{14}	10	390.94	100%	650.57	100%	**142.78**	100%	301.09	100%	161.16	100%
	30	1313.92	100%	3989.14	100%	**392.88**	100%	1202.84	100%	10000	0%
	50	2563.67	100%	10000	0%	**771.63**	100%	2639.73	100%	10000	0%
f_{15}	10	364.54	100%	607.03	100%	**133.09**	100%	282.29	100%	150.49	100%
	30	1258.76	100%	3790.53	100%	**379.89**	100%	1147.13	100%	10000	0%
	50	2472.18	100%	10000	0%	**839.47**	99%	2553.35	100%	10000	0%
f_{16}	10	3028.18	70%	**841.63**	92%	10000	0%	10000	0%	10000	0%
	30	8979.23	11%	**6029.06**	46%	10000	0%	10000	0%	10000	0%
	50	**9964.84**	2%	10000	0%	10000	0%	10000	0%	10000	0%
f_{17}	10	10000	0%	10000	0%	9903.05	1%	**9805.79**	2%	10000	0%
	30	10000	0%	10000	0%	10000	0%	10000	0%	10000	0%
	50	10000	0%	10000	0%	10000	0%	10000	0%	10000	0%
f_{18}	10	1441.76	100%	2612.27	100%	**421.34**	100%	1069.01	100%	505.47	100%
	30	10000	0%	10000	0%	**4510.35**	96%	10000	0%	10000	0%
	50	10000	0%	10000	0%	10000	0%	10000	0%	10000	0%
f_{19}	2	161.43	100%	210.39	100%	673.93	94%	**130.79**	100%	500.38	96%
f_{20}	2	142.43	100%	178.23	100%	**74.03**	93%	120.45	98%	96.75	98%
f_{23}	2	147.97	100%	190.55	100%	**74.93**	89%	122.19	97%	98.92	97%
Best in		9		2		**16**		8		2	

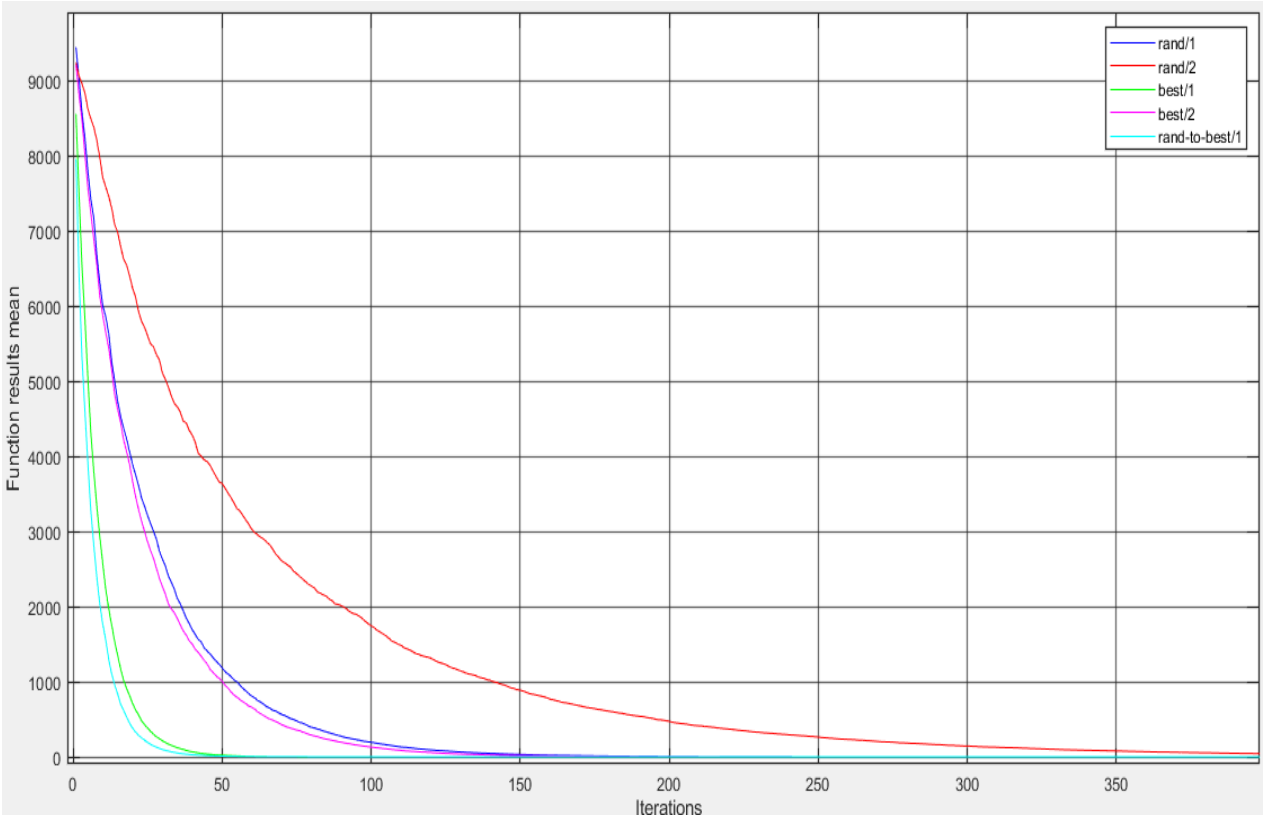

Figure 23: Convergence speed for function 15 with D=30

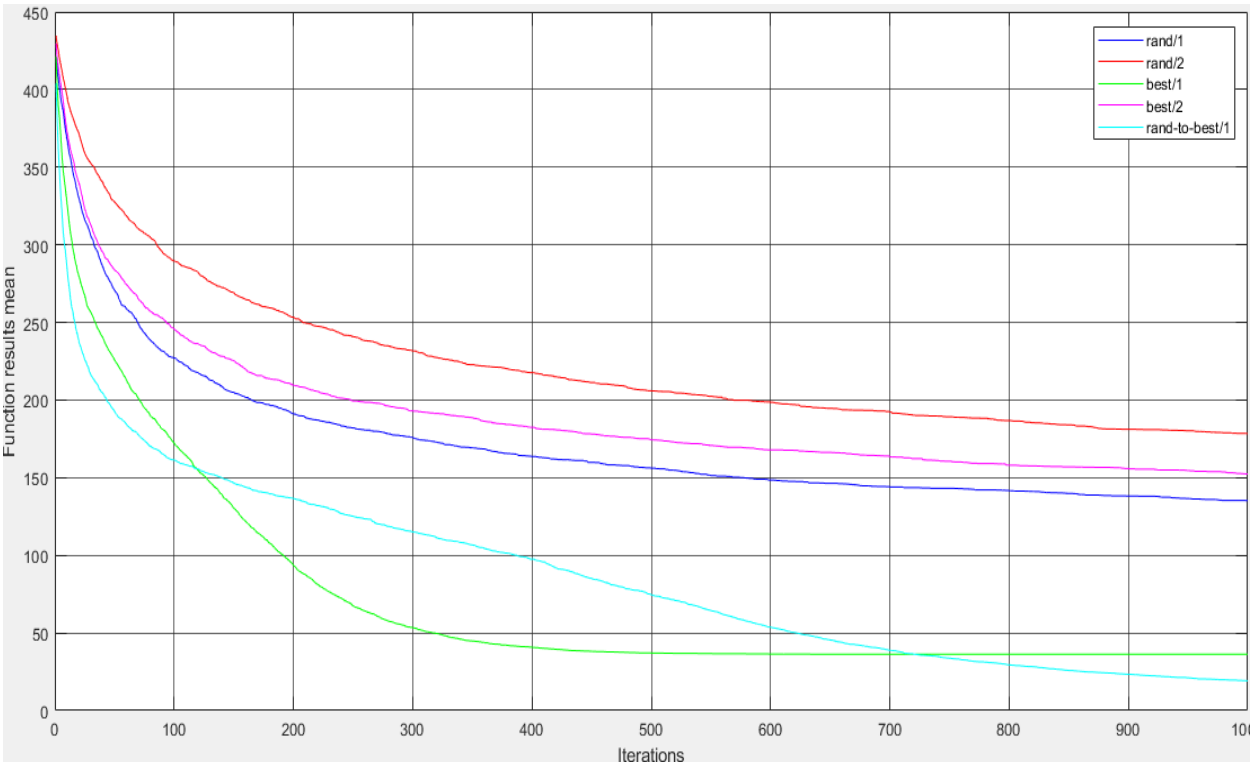

Figure 24: Convergence speed for function 12 with D=30

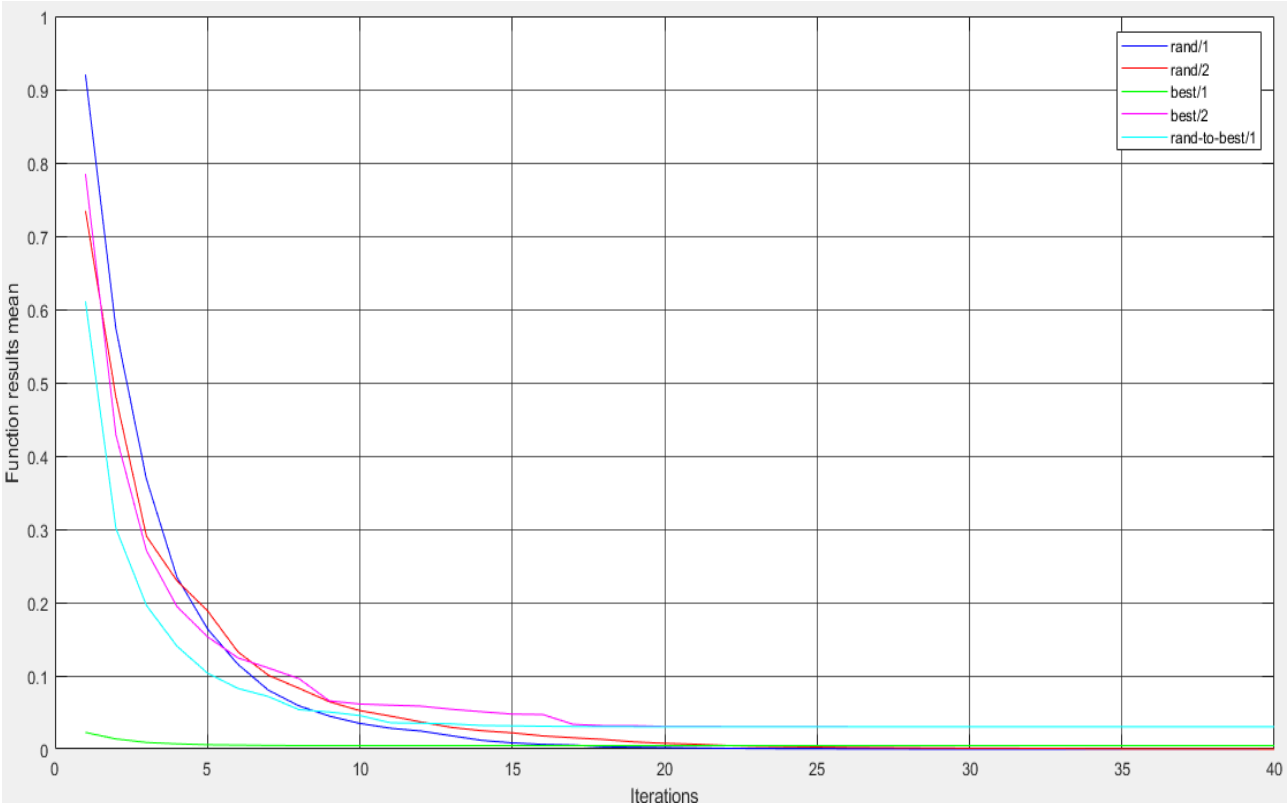

Figure 25: Convergence speed for function 19 with D=30

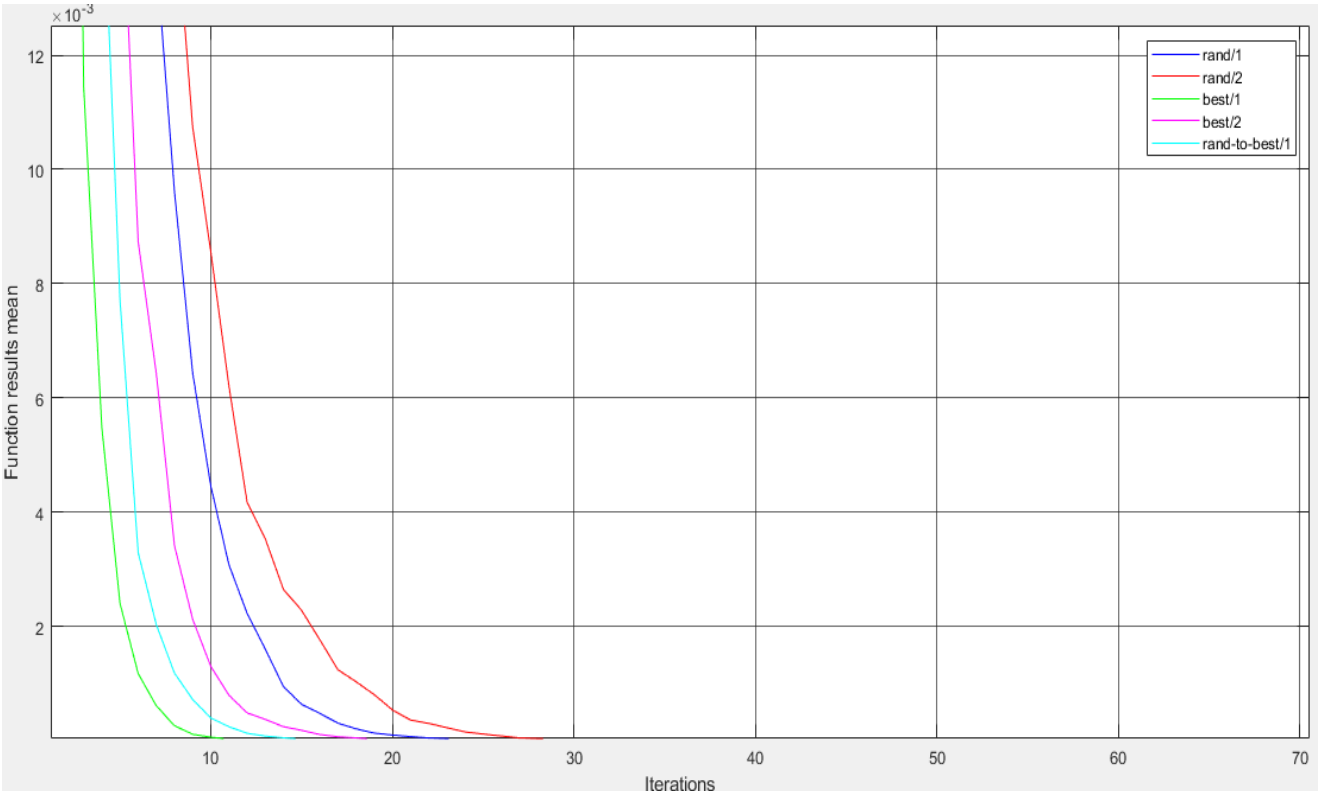

Figure 26: Convergence speed for function 23 with D=30

Figures 22 - 25 shows the convergence speed results of the different DE variants in four cases. To obtain clear graphs, we plotted the results only for relatively small numbers of iterations. In Figure 22 and 23, the variant rand-to-best/1 starts as the fastest; however, for fig 22, the best/1 variant rapidly reaches and surpasses it around the iteration 200.

On the other hand, looking at Figure 19 and 20, we can see that the best/1 is the fastest variant. Based on the Figures, we can say that both rand-to-best/1 and best/1 seems to be the best choice for low number of iterations, as they achieve solution that could be considered as acceptable in certain cases with very low number of iterations, making it suitable for scenarios characterized by extremely limited iterations and severely constrained computing resources and time. Nevertheless, compared to rand-to-best/1, the variants best/1 and rand/1 achieve significantly better results for higher numbers of iteration.

C. Execution time test:

Finally, our third test measures the actual computation time taken to reach a solution within a predefined number of iterations (fixed-cost solve time). The population size is set to NP = 40, the scaling factor is F = 0.5, and the mutation rate is CR = 0.5. Each test is conducted for three dimension values, D = {10, 30, 50}, across six functions ($f1$, $f8$, $f13$, $f16$, $f17$, $f24$).

According to the obtained results presented in Table 12 and their Friedman mean ranks, rand/1 was the fastest variant, followed by rand/2, best/1, rand-to-best/1, and finally best/2. As anticipated, rand/1 generally outperforms other variants in terms of speed due to its simplicity, providing relatively acceptable solutions quickly.

However, in certain cases, particularly with more complex problems like functions 16 and 17 or when the dimension is 50, we can see that other variants can achieve faster convergence. We can also notice that when an algorithm performs well in execution time, particularly when a problem is hard, it usually does not perform well in the convergence speed test, meaning when the problem is hard enough and the algorithm is fast it usually has a slow speed of convergences. That could be explained by the structure of the DE algorithm knowing that in each new generation when a better solution is found it should replace the solution of the previous generation (parent) which could be time consuming when the convergence speed is high.

Table 12: Execution time test results (s)

fun	Dim	Execution Time				
		Rand/1	**Rand/2**	**Best/1**	**Best/2**	**Rand-to-Best/1**
f_1	10	**0.332606**	0.334926	0.400627	0.404486	0.400921
	30	**0.391334**	0.410332	0.465159	0.461608	0.466243
	50	**0.459688**	0.514662	0.541747	0.535121	0.538722
f_8	10	**0.307482**	0.313011	0.377498	0.378889	0.375186
	30	**0.363052**	0.371398	0.434124	0.437545	0.435993
	50	**0.417367**	0.426464	0.490887	0.492515	0.490565
f_{13}	10	**0.375986**	0.377794	0.447519	0.445256	0.453637
	30	**0.427909**	0.431745	0.502209	0.499884	0.507229
	50	0.488349	**0.488083**	0.558508	0.555085	0.562871
f_{16}	10	5.572309	5.587099	5.567081	**5.559244**	5.569061
	30	15.21645	15.15512	15.15683	15.18159	**15.02709**
	50	24.5761	24.59797	24.73138	24.71708	**24.40799**
f_{17}	10	**0.480796**	0.49327	0.494481	0.550922	0.508549
	30	1.645233	1.989942	**1.532258**	1.865268	1.574411
	50	4.560634	5.162347	**3.560737**	4.828732	4.118343
f_{24}	2	**0.287685**	0.292097	0.361062	0.361654	0.362348
Friedman rank		**1.8125**	2.5625	3.3125	3.8125	3.5000
Best in		**10**	1	2	1	2
Friedman Prob		0.0022				

IV.5. Conclusion:

In this chapter, we conducted a comprehensive experimental study to investigate the impact of control parameters and various DE algorithm variants on the performance of the Differential Evolution (DE) algorithm. Our study utilized several benchmark functions, analyzed through statistical methods to ensure robust conclusions.

Our findings of the first study revealed that increasing the problem dimension deteriorates the DE algorithm's performance due to the larger and more complex search space. Conversely, increasing the number of iterations consistently improved performance by providing more opportunities for the algorithm to refine its solutions. The optimal population size was found to depend on the scaling factor. For a scaling factor of 0.2, a larger population size of 100 was optimal, providing necessary diversity for broader exploration. With a scaling factor of 0.5, a population size of 60 achieved the best balance between exploration and exploitation. When the scaling factor was set to 0.9, a smaller population size of 20 was optimal, allowing for quicker convergence given the aggressive exploration. A moderate scaling factor of 0.5 consistently produced the best results across various benchmark functions, balancing exploration and exploitation effectively. A lower mutation rate, particularly a value of 0, produced the best results across the majority of benchmark functions, suggesting that less aggressive mutation promotes better convergence in this context.

In the second study, we found that the binomial crossover method (bin) generally performed the best, leading us to use it in subsequent comparisons. The convergence speed tests highlighted that both Rand-to-best/1/bin and best/1 were initially the fastest for low iteration numbers, making it suitable for scenarios characterized by extremely limited iterations and severely constrained computing resources and time. Conversely, for scenarios with ample resources and no time constraints, the best/1 variant also exhibited the best convergence speed. However, in terms of convergence quality within moderate and limited number of iterations, the rand/1 variant demonstrated superior performance, making it suitable for scenarios with moderate and constrained computational resources and time. Execution time tests revealed that Rand/1/bin was the fastest variant, especially advantageous for scenarios requiring quick, relatively acceptable solutions. However, the performance varied with problem complexity, and faster execution times did not correlate with faster convergence speeds.

Overall, our study provides a detailed understanding of how different control parameters and DE variants affect the algorithm's performance. The insights gained can guide the selection of optimal settings for DE algorithms in various optimization scenarios, enhancing their effectiveness and efficiency in solving complex problems.

GENERAl CONCLUSION

General Conclusion

This thesis has investigated the impact of control parameters and variants of the Differential Evolution (DE) algorithm on its performance across various optimization tasks. Chapter I presented a foundational exploration of optimization problems, delineating between complete and approximate methods while chapter II provided an overview of metaheuristic algorithms, emphasizing their application in diverse domains such as renewable energy and mechanical engineering. It laid the groundwork for a focused study on DE in Chapter III, detailing its mechanisms and variants.

Chapter IV presented a comprehensive experimental study evaluating the influence of control parameters and DE variants on algorithm performance. Through rigorous analysis of benchmark functions and statistical methods, we found that the performance of the Differential Evolution (DE) algorithm is significantly influenced by control parameters and algorithm variants. Increasing the problem dimension generally worsened the algorithm's performance due to a more complex search space, while increasing the number of iterations consistently improved it. The optimal population size varied with the scaling factor: a larger population size was beneficial for lower scaling factors, while a smaller size was optimal for higher scaling factors. A moderate scaling factor of 0.5 consistently produced the best results, and a lower mutation rate was found to promote better convergence.

Moreover, the binomial crossover method proved to be highly effective, while the Rand-to-best/1/bin and best/1 variants showed the fastest initial convergence in short iteration cycles. Additionally, the rand/1 variant demonstrated superior performance during moderate iteration scenarios while the best/1 variant excelled in high iteration scenarios. Execution time tests revealed that the Rand/1/bin variant was the fastest, particularly useful for scenarios requiring quick, relatively acceptable solutions. Overall, our findings offer valuable insights into selecting optimal settings for DE algorithms to enhance their performance in various optimization tasks.

In summary, this thesis contributes to the field of optimization by thoroughly examining DE algorithmic parameters and variants. Future research can extend these insights by exploring hybrid approaches, adapting DE for specific applications, and investigating self-adaptive versions of DE. Furthermore, there is a need for further studies to implement DE algorithms in optimizing MPPT systems and to compare these implementations with their original algorithms. Additionally, a more detailed comparison between the DE variants, including an analysis of more variants and the impact of control parameters on them, is essential.

BIBLIOGRAPHICAL REFERENCES

References

[1] D. A. Perez Ruiz, *Modern Optimization with R*, vol. 70, no. Book Review 3. London, 2014. doi: 10.18637/jss.v070.b03.

[2] R. Kumar, "Optimization: algorithms and applications," *Choice Rev. Online*, vol. 53, no. 08, Apr. 2016, doi: 10.5860/CHOICE.195857.

[3] H. Poincar, *L'invention mathématique*1908 .. [Online]. Available: https://adsabs.harvard.edu/full/1908BSAFR..22..389P

[4] Sons and J. Wiley, *EVOLUTIONARY OPTIMIZATION ALGORITHMS Biologically-Inspired and Population-Based Approaches to Computer Intelligence*. Canada: John Wiley & Sons, 2013.

[5] J. A. Snyman and D. N. Wilke, *Practical Mathematical Optimization*, vol. 97. in Applied Optimization, vol. 97. New York: Springer-Verlag, 2005. doi: 10.1007/b105200.

[6] El-Ghazali Talbi, *METAHEURISTICS FROM DESIGN TO IMPLEMENTATION*. 2009. [Online]. Available: http://www.wiley.com/go/permission.

[7] G. K. Ananthasuresh, "Classification of Optimization Problems and the Place of Calculus of Variations in it," n.d.

[8] B. Mebarka and A. Y. Nour, "Etude de l'influence des paramètres d'optimisation sur les algorithmes méta-heuristiques," *Confrontation*, 2022.

[9] S. Ben Mena, "Introduction aux méthodes multicritères d'aide à la décision," *Biotechnol. Agron. Soc. Environ.*, vol. 4, no. 2, 2000, [Online]. Available: https://popups.uliege.be/1780-4507/index.php?id=15338

[10] C. Solnon, S. A. Ilog, and S. Antipolis, "Contributions à la résolution pratique de problèmes combinatoires — des fourmis et des graphes —," 2005, [Online]. Available: title=%7BContributions %7B%5C%60a%7D la r%7B%5C'e%7Dsolution pratique de probl%7B%5C%60e%7Dmes combinatoires%7D,

[11] Hiriart Urruty and Jean Baptiste, "Les mathématiques du mieux faire. Volume 1. Premiers pas en optimisation*2007*"، [Online]. Available: https://hal.science/hal-04142154/

[12] M. BIERLAIRE, "Introduction à l'optimisation différentiable*2006*"، Accessed: May 11, 2024. [Online]. Available: https://books.google.com/books?hl=ar&lr=&id=HK55T5x2bygC&oi=fnd&pg=PA447&d

q=M.+Bierlaire,+Introduction+à+l%27optimisation+différentiable:+PPUR+presses+polytechniques,+2006&ots=6LhnndXSW7&sig=FMhnGtriVxr5UCADFT5dgPS88rk

[13] C. Solnon, "Contributions à la résolution pratique de problèmes combinatoires*2005* "، Accessed: May 11, 2024. [Online]. Available: https://hal.science/hal-01470067/

[14] L. Boyd, Stephen P and Vandenberghe, *Convex Optimization*. Cambridge university press, 2004. [Online]. Available: https://web.stanford.edu/~boyd/cvxbook/

[15] S. G. Johnson, "A Brief Overview of Optimization Problems,", 2019.

[16] G. Berthiau and P. Siarry, "État de l'art des méthodes 'd'optimisation globale,'" *Rairo Oper. Res.*, vol. 35, no. 2001, p. 365, 2002, doi: 10.1051/ro.

[17] S. Borak and W. Karl, " *Matheuristics*",2009

[18] G. Nemhauser and L. Wolsey, *Integer and Combinatorial Optimization*. John Wiley & Sons, 1988. doi: 10.1002/9781118627372.

[19] J. Mira and J. R.Alvarez, *Artificial Intelligence and Knowledge Engineering Applications: A Bioinspired Approach*, vol. 3562. in Lecture Notes in Computer Science, vol. 3562. Berlin, Heidelberg: Springer Berlin Heidelberg, 2005. doi: 10.1007/b137296.

[20] B. Belin, "Conception interactive d'environnements urbains durables à base de résolution de contraintes,", 2014, [Online]. Available: https://hal.science/tel-01095433/

[21] K. L. Du and M. N. S. Swamy, *Search and optimization by metaheuristics: Techniques and algorithms inspired by nature*. 2016. doi: 10.1007/978-3-319-41192-7.

[22] H. Hachimi, "HYBRIDATIONS D'ALGORITHMES MÉTAHEURISTIQUES EN OPTIMISATIONGLOBALE ETLEURS APPLICATIONS," Mohammed V - Agdal, Rabat École Mohammadia d'Ingénieurs, 2013.

[23] O. AITSAHED, "FUZZY PREDICTIVE CONTROL USING META-HEURISTIC ALGORITHMS," Universite de blida 1, 2015.

[24] C. A. C. Coello, G. B. Lamont, and D. A. Van Veldhuizen, *Evolutionary Algorithms for Solving Multi-Objective Problems*. in Genetic and Evolutionary Computation Series. Boston, MA: Springer US, 2007. doi: 10.1007/978-0-387-36797-2.

[25] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to algorithms, 4 Edition*. 2022. [Online]. Available: http://lccn.loc.gov/2021037260

[26] A. Qing and C. K. Lee, *Differential Evolution in Electromagnetics*. 2010.

[27] L. Skanderová, *Differential Evolution*, vol. 5. in Springer Optimization and Its Applications,

vol. 5. Boston, MA: Springer US, 2006. doi: 10.1007/978-0-387-36896-2.

[28] F. VITALIY, “DIFFERENTIAL EVOLUTION In Search of Solutions,” *J. AWWA*, vol. 96, no. 11, , Nov. 2006, doi: 2006929851.

[29] K. Price, R. M. Storn, and J. A. Lampinen, “Differential Evolution Algorithm,” in *Computational Intelligence-based Optimization Algorithms*, 2006 Springer Science & Business Media, Ed., Boca Raton: CRC Press, 2006, p. 539. doi: 10.1201/9781003424765-8.

[30] C. Lin, A. Qing, and Q. Feng, “A comparative study of crossover in differential evolution,” *J. Heuristics*, vol. 17, no. 6, , Dec. 2011, doi: 10.1007/s10732-010-9151-1.

[31] P. Meister, *Adaptive differential evolution: a robust approach to multimodal problem optimization*. 2009. [Online]. Available: http://www.springerlink.com/index/68K17776343H67G2.pdf%5Cnhttp://books.google.com/books?hl=en&lr=&id=yTAtVQev0uMC&oi=fnd&pg=PR1&dq=Adaptive+differential+evolution:+A+robust+approach+to+multimodal+problem+optimization&ots=Gb0VOieVUR&sig=jkSnVSSaGV479OwtAR0

[32] A. Qing, *DIFFERENTIAL EVOLUTION Fundamentals and Applications in Electrical Engineering*, vol. 206. 2009. doi: 10.3254/ENFI210034.

[33] V. Beiranvand, W. Hare, and Y. Lucet, “Best practices for comparing optimization algorithms,” *Optim. Eng.*, vol. 18, no. 4, 2017, doi: 10.1007/s11081-017-9366-1.

[34] E. Mezura-Montes, J. Velázquez-Reyes, and C. A. Coello Coello, “A comparative study of differential evolution variants for global optimization,” in *Proceedings of the 8th annual conference on Genetic and evolutionary computation*, New York, NY, USA: ACM, Jul. 2006. doi: 10.1145/1143997.1144086.

[35] “Friedman’s test - MATLAB friedman.” https://www.mathworks.com/help/stats/friedman.html#bubrqay-3.