République algérienne démocratique et populaire
Ministère de l'enseignement supérieur et de la recherche scientifique

Centre Universitaire Abdelhafid Boussouf Mila

Institut de Mathématiques et Informatique

Département Informatique

MÉMOIRE DE MASTER

Pour obtenir le diplôme de Master en Informatique

Option : Sciences et Technologies de l'Information et de la Communication (STIC)

Boosting the performance of Differential Evolution Metaheuristic

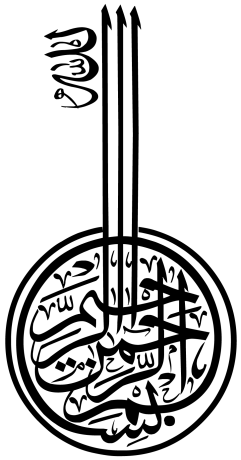
---

Réalisé par : KHEMICI Dallal
TAMEN Habet Ellah

Soutenue devant le jury:

President:	Dr. BENHAMMADA Sadek	Centre Universitaire de Mila – Abdelhafid Boussouf
Encadrant:	Dr. KHALFI Souheila	Centre Universitaire de Mila – Abdelhafid Boussouf
Examinateur	Mr. LALOUCI Ali	Centre Universitaire de Mila – Abdelhafid Boussouf

Année universitaire 2023–2024



Acknowledgments

Praise be to Almighty God, who has given us strength and courage to successfully complete this work, which, in life, requires first and foremost the blessings of Allah and, secondly, the help and support of many people.

We would like to sincerely thank our supervisor, Dr.Khalfi Souheila, for the trust she has placed in us in supervising this work. Her advice, guidance, great availability, and methodological comments throughout this project were essential. Her support allowed us to precisely define the subject and successfully complete this work.

We also thank the jury members for the interest they have shown in evaluating our work and, we hope that this thesis meets their expectations.

Finally, We would like to acknowledge our parents, who have encouraged and supported us both morally and materially throughout our university studies, helping us to achieve our intellectual goals.

Dedication

To my dear parents,
To my brother and my sister,
To my dear grandmother,
I dedicate this humble work,

HIBA

To my dear parents,
To my sisters,
To my niece Maram, and nephews Youcef and Sami,
I dedicate this humble work,

DALAL

Last but not least, we want to thank ourselves for believing in us and doing all this hard work.

ملخص

تُعتبر التحسين أداة حيوية في المجتمع العلمي، حيث تدفع التقدم وتحل المشكلات المعقدة عبر مختلف المجالات. من بين تقنيات التحسين المختلفة، اكتسبت الميتاهيورستيك مكانة بارزة بفضل مرونتها وفعاليتها. يُستخدم خوارزمية التطور التفاضلي DE ، وهي خوارزمية ميتاهيورستيك رائدة، على نطاق واسع بسبب كفاءتها وصلابتها في التعامل مع تحديات التحسين المتنوعة. تم تطبيق DE بنجاح في العديد من التطبيقات الواقعية، بما في ذلك تصميم الهندسة، والتعلم الآلي، والنمذجة المالية. على الرغم من قوتها، تواجه DE العديد من التحديات مثل موازنة الاستكشاف والاستغلال والحفاظ على الأداء في المساحات ذات الأبعاد العالية. تُعزى أداء DE بشكل كبير إلى المعايير والمشغلين المستخدمين. استنادًا إلى ذلك، نجمع بين أفكار من نهج مختلفة موجودة لتعزيز قدرات DE . يتضمن ذلك تعديل حجم السكان الثابت الأساسي واستخدام استراتيجيات مشغلين متنوعة مع تقنيات التحكم في المعايير. من خلال القيام بذلك، نسعى إلى تحسين كفاءة DE عبر سيناريوهات تحسين متنوعة. من خلال تجارب مكثفة باستخدام بيئة قياس الأداء BBOB ، تظهر النهج الجديدة نتائج واعدة.

الكلمات المفتاحية: التحسين، التطور التفاضلي، التحكم في المعلمات

Abstract

Optimization is a critical tool in the scientific community, driving advancements and solving complex problems across various domains. Among the various optimization techniques, metaheuristics have gained prominence for their flexibility and effectiveness. Differential Evolution (DE), a leading metaheuristic algorithm, is widely used due to its efficiency and robustness in handling diverse optimization challenges. DE has been successfully applied in numerous real-world applications, including engineering design, machine learning, and financial modeling. Despite its strengths, DE faces many challenges such as balancing exploration and exploitation and maintaining performance in high-dimensional spaces. The performance of DE is largely attributed to the parameters and operators used. Motivated by this, we combine ideas from different existing approaches to enhance DE's capabilities. This includes modifying the basic fixed population size and employing various operator strategies with parameter control techniques. By doing this, we aim to improve the efficiency of DE across diverse optimization scenarios. Through extensive experimentation using the BBOB benchmarking environment, the new approach demonstrates promising results.

Keywords: Optimization, Differential Evolution (DE), Parameter Control.

Résumé

L'optimisation est un outil essentiel dans la communauté scientifique, favorisant les avancées et résolvant des problèmes complexes dans divers domaines. Parmi les différentes techniques d'optimisation, les métaheuristiques ont gagné en importance grâce à leur flexibilité et leur efficacité. L'évolution différentielle (DE), un algorithme métaheuristique de premier plan, est largement utilisée en raison de son efficacité et de sa robustesse à traiter divers défis d'optimisation. La DE a été appliquée avec succès dans de nombreuses applications réelles, y compris la conception d'ingénierie, l'apprentissage automatique et la modélisation financière. Malgré ses forces, la DE fait face à de nombreux défis, tels que l'équilibre entre exploration et exploitation et le maintien des performances dans des espaces de haute dimension. La performance de la DE est en grande partie attribuée aux paramètres et aux opérateurs utilisés. Motivés par cela, nous combinons des idées de différentes approches existantes pour améliorer les capacités de la DE. Cela inclut la modification de la taille de la population fixe de base et l'utilisation de diverses stratégies d'opérateurs avec des techniques de contrôle des paramètres. En faisant cela, nous visons à améliorer l'efficacité de la DE dans divers scénarios d'optimisation. Grâce à des expérimentations approfondies utilisant l'environnement de benchmarking BBOB, la nouvelle approche démontre des résultats prometteurs.

Mots clés : Optimisation, Évolution Différentielle (DE), Contrôle des Paramètres.

Contents

List of Figures

List of Tables

General Introduction

Optimization is becoming increasingly vital in the scientific community, driving advancements and solving complex problems and The more complex the difficulties, the more crucial it becomes. Various methods are employed to tackle optimization problems including exact algorithms which systematically explore the solution space to guarantee optimal solutions. In contrast, approximate methods such as heuristics and metaheuristics do not guarantee optimal solutions but are often more practical for large or complex problems. Heuristic approaches apply rules of thumb to quickly find good-enough solutions by focusing on local searches and exploiting problem-specific knowledge. Metaheuristics on the other hand, employ more flexible and advanced techniques, inspired by natural processes like evolution or swarm behavior, to explore and exploit the search space efficiently. Our focus lies on these metaheuristic methods, aiming to leverage their robust mechanisms to find high-quality solutions in diverse optimization scenarios.

As a new branch of evolutionary algorithms (EA), the Differential Evolution (DE) algorithm shares a similar structure with EA, including three important evolutionary operators: mutation, crossover, and selection. The performance of DE primarily relies on these operators and their associated parameter settings. DE has attracted great attention from scientific researchers due to its success in solving various numerical and real-world problems, ease of use, speed, and robustness. Despite its proven performance in optimization, DE suffers from an imbalance between the exploration of the search space and the exploitation of promising regions. Additionally, its performance decreases with increasing dimensions, Further limitations are discussed in [22]. To address these limitations, several variants have been proposed. Some focus on fine-tuning the parameters of DE, while others propose novel crossover or mutation strategies to optimize its performance.

Motivated by the limitations of DE, we have proposed a variant of this algorithm that divides the parent population into three sub-populations based on fitness values. Each sub-population employs a different mutation strategy, responsible for either exploitation

or exploration. Additionally, we incorporate an adaptive control of parameters F and CR in each sub-population.

To present our work, we have organized it as follows:

Chapter 1 discusses the definitions of optimization problems and various resolution methods, including exact and approximate methods. It also covers the classification of resolution methods, single-solution metaheuristics like Simulated Annealing and Tabu Search, and population-based metaheuristics such as Genetic Algorithms, Differential Evolution, and Particle Swarm Optimization. Additionally, it includes a critical review of some metaheuristics based on metaphor.

Chapter 2 delves into Differential Evolution (DE) variants, discussing various adaptations of the DE algorithm. It covers DE with adaptation of F and Cr parameters, adaptive population size control, new or improved mutation strategies, new or improved crossover strategies, and hybrid DE algorithms. Additionally, it explores the applications of DE.

In *Chapter 3*, we present the main contributions of our work. We begin by discussing the motivations that led us to study the parameters and operators of the Differential Evolution (DE) algorithm. Following this, we introduce the variants of DE that we have developed. We then present the experimental results of the proposed approaches, tested and validated using COCO platform.

Finally, our manuscript concludes with a general summary that recapitulates our work and the results obtained, while also suggesting potential future directions for research.

Chapter 1

Optimization and Metaheuristics

Introduction

Optimization is essential in many fields, providing a framework for addressing complex tasks by modeling them as problems with defined objectives, such as minimizing cost or maximizing quality. Problems are categorized into discrete and continuous domains, allowing for the selection of appropriate algorithms. Continuous optimization, which deals with solutions that vary across a range of values, is particularly important for real-world issues like parameter tuning in machine learning [60] and optimal resource allocation in engineering [5]. Thus, the subsequent sections primarily focus on continuous optimization.

This chapter covers optimization problem definitions and classifications, resolution methods (both exact and approximate), and metaheuristic approaches. We will analyze single-solution strategies like Simulated Annealing (SA) and Tabu Search (TS), as well as population-based techniques like Genetic Algorithms (GA), Differential Evolution (DE), and Particle Swarm Optimization (PSO). Finally, we will critically examine metaphor-based metaheuristics from the perspective of experts.

1.1 Definitions of an optimization problem

Optimization is the process of finding the best solution or achieving the optimal outcome from a set of feasible alternatives. In the context of single-objective continuous optimization problems:

$$\min f(x), \quad \text{s.t.} \quad x \in S$$

where $\mathbf{x} = (x_1, x_2, \ldots, x_D)$ represents a candidate solution to the D-dimensional optimization problem defined by the objective function $f(\cdot)$, which we aim to minimize (or maximize). The search space S is defined by the upper bound vector $\mathbf{a} = (a_1, a_2, \ldots, a_D)$

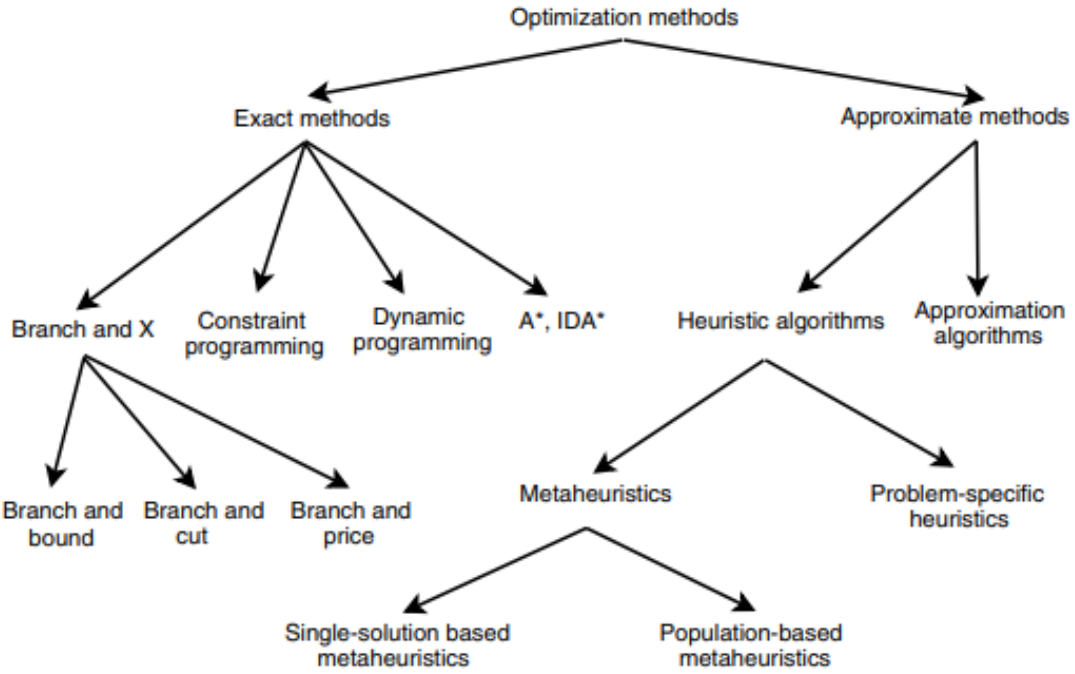


Figure 1.1: Classical optimization methods [71].

and the lower bound vector $\mathbf{b} = (b_1, b_2, \ldots, b_D)$, such that $a_i \leq x_i \leq b_i$ for all $i \in \{1, 2, \ldots, D\}$. These constraints, also known as bound constraints, give rise to the term continuous optimization with bound constraints.

1.2 Resolution methods

In optimization, resolution methods refer to a range of strategies and algorithms used to identify the best solutions to challenging problems. These methods are often classified into two categories: exact methods and approximate methods. Each method within these categories has its algorithm to guide the optimization process, as illustrated in figure 1.1:

1.2.1 Exact Methods

In the context of optimization, exact methods refer to algorithms or approaches designed to find the best solution to a given optimization problem within a finite amount of time. Typically, these techniques involve exhaustive search across the entire space of feasible solutions or direct computation using mathematical formulas to determine the optimal solution. They stand in contrast to heuristic or approximation approaches, which are often faster and more scalable for complex or large-scale problems, although they may not always guarantee optimality [33].

1.2.2 Approximate Methods

Approximate methods, such as heuristics and simulations, often provide solutions to optimization problems that are potentially suboptimal. These methods use computational techniques to find feasible solutions within a reasonable timeframe [33]. Within the category of approximate methods, two subclasses of algorithms can be distinguished: approximation algorithms and heuristic algorithms.

1.2.2.1 Heuristics

When tackling large-scale problems, heuristics perform well by providing adequate performance at reasonable costs across various problem types, although they usually do not guarantee optimal or near-optimal solutions [71].

1.2.2.2 Metaheuristics

Researchers have defined metaheuristics in various ways. Here, we will present some of these definitions:

According to Talbi [70], the term "heuristic" originates from the ancient Greek word "heuriskein," which means the art of discovering new strategies or rules to solve problems. The suffix "meta", also a Greek word, means "upper level methodology".

In 2003, Blum and Roli [7] defined a metaheuristic as "a high-level problem-independent algorithmic framework that provides a set of guidelines or strategies to develop heuristic optimization algorithms". The objective is to effectively search the space in order to identify nearly ideal solutions.

Metaheuristics are defined by Michel Gendreau [50] as "strategies that guide the search process by learning from the generated solutions so far". Stated differently, they utilize the data gathered throughout the search process to concentrate their efforts on more promising areas within the solution space.

In general, a metaheuristic method is particularly relevant for solving search and optimization problems. This approach utilizes one or more heuristics, inheriting three key properties [4]: (i) it aims to find a near-optimal solution rather than the exact optimal solution, (ii) it typically lacks a rigorous proof of convergence to the optimal solution, and (iii) it is usually computationally faster than exhaustive search.

1.2.2.3 Classification for Metaheuristic Optimization Methods

[4] states that there are multiple criteria that can be used to classify optimization methods. Below, we discuss a few popular standards for grouping these algorithms:

1. Categorizing metaheuristics according to *the domain that they mimic*: For metaheuristics, catch-all phrases like "nature-inspired" and "bio-inspired" are frequently employed. They can be further divided into three subcategories: algorithms based on physical phenomena, swarm intelligence, and evolution.
2. Another often used criterion for categorizing metaheuristic methods is *the number of initial solutions* that are changed in successive rounds. Single-solution metaheuristics begin with a single starting solution that is iteratively changed. It should be noted that while several solutions may be used throughout the modification process, each subsequent iteration will only employ one solution. Multiple starting solutions are used to begin optimization in population-based metaheuristics. Many solutions are updated in each iteration, and some of them move on to the next.
3. *Deterministic* versus *stochastic* methods: Deterministic approaches depart from the random initial solution(s) in a predetermined direction. Probabilistic jumps from the present solution(s) to the next are possible using stochastic methods, also referred to as discontinuous methods.
4. *Greedy* versus *non-greedy* methods: Greedy algorithms typically start their search near the current solution solution and switch to a superior one as soon as it is discovered. This behavior often leads to a local optimum. Non-greedy approaches either offer a mechanism to escape from a local optimum or delay changing the solution(s) for a few iterations.

1.3 Single-Solution Metaheuristics

In optimization, a class of techniques known as single-solution metaheuristics is devoted to iteratively refining a single solution.

1.3.1 Simulated Annealing Algorithm

Kirkpatrick et al. [45] were the first to propose simulated annealing (SA) as a method for solving discrete optimization problems. SA has also been applied to continuous optimization problems. While the theoretical aspects of continuous simulated annealing have been explored by fewer researchers compared to its discrete counterpart, significant research on continuous simulated annealing was conducted by Dekkers and Aarts [29], who not only provided a theoretical convergence proof but also developed an algorithm for solving continuous optimization problems.

Most notably, there are similarities between the discrete simulated annealing process and the continuous SA algorithm, but also differences. When comparing discrete and

continuous optimization problems, SA uses distinct definitions for neighbors of solutions in the design space. A small variation in the configuration defines the neighborhood of a solution for combinatorial optimization problems (e.g., a 2-change neighborhood for the traveling salesman problem). However, in continuous optimization problems, the neighborhood of a solution is defined by distances in the design space, which makes it easier to consider than in combinatorial problems. Determining the neighborhood range for generating the next point is crucial for continuous optimization problems [39]. If the neighborhood range is set at a fixed range, each problem should have its own range established.[51]

The advantages and disadvantages of SA are well summarized in [39]. Among its notable disadvantages are the substantial time required to find the optimum solution and the difficulty in determining the appropriate cooling schedule. Despite these challenges, SA is theoretically sound, efficient, and simple to implement [36]. By accepting moves that deteriorate the objective function value in the pursuit of finding a global optimum, simulated annealing offers a method to escape local optima, which is its primary characteristic.

The pseudo code 1.1 outlines the key steps of the Simulated Annealing algorithm.

Algorithme 1.1 Simulated Annealing Algorithm.

Initialize current solution S_{current}
Initialize temperature T
Initialize cooling rate α
Initialize stopping criteria
while $T >$ stopping criteria **do**
    Generate a new solution S_{new} by making a small change to S_{current}
    Calculate energy difference $\Delta E = \text{energy}(S_{\text{new}}) - \text{energy}(S_{\text{current}})$
    if $\Delta E < 0$ or $e^{-\Delta E/T} >$ random number between 0 and 1 **then**
        Accept S_{new} as the current solution: $S_{\text{current}} \leftarrow S_{\text{new}}$
    end if
    Reduce temperature: $T \leftarrow \alpha \times T$
end while
return S_{current}

1.3.1.1 Variants of Simulated Annealing Algorithm

Over time, numerous variants and improvements to the basic Simulated Annealing (SA) algorithm have been introduced to enhance its efficacy in tackling diverse optimization challenges. [36] provides a comprehensive overview of SA and its enhanced variants, including Fast Simulated Annealing (FSA), Sequential Monte Carlo simulated annealing (SMC-SA), and a newly proposed scheme called Curious Simulated Annealing (CSA).

1.3.2 Tabu Search Algorithm

A metaheuristic called Tabu Search (TS) was first created by Glover [34]. It has been effectively used to solve a number of combinatorial optimization issues. On the other hand, its application to the global minimization of functions dependent on continuous variables is the subject of very few works. Let us briefly review the adaptation of Glover's concept to the continuous case in [65]. The algorithm, called *CTS*, begins with an initial solution s that is chosen at random. This present solution, s, generates a collection of neighbors, s', from it. The neighbors of the current solution, which are part of a later established "tabu list", are systematically removed in order to prevent the risk of a cycle arising. For every generated solution s', the objective function to be minimized is assessed, and even if the best neighbor of s is worse than s, it becomes the new current solution. S is added to a circular list of tabu solutions following each 'move'; when the list is full, the initial solution entered is removed. Then, a fresh "iteration" is carried out, where the prior process is repeated beginning from the new current position,until some stopping condition is reached. Usually, the algorithm terminates without improving the value of the objectif function after a predetermined number of rounds. The main steps of the Tabu Search algorithm are described in the pseudo code 1.2.

Algorithme 1.2 Tabu Search Algorithm.

0: Choose an initial solution curr
1: N(curr) ← Find a subset of f(curr), neighbors of curr that are not in the tabu list
2: Find the best one (next) in set N(curr)
3: If $f(\text{next}) < f(\text{curr})$ then set curr = next ▷ positive move
4: If $f(\text{next}) > f(\text{curr})$ ▷ negative move
5: Update tabu list, curr ← next
6: Go to second step, till the stopping condition.

1.3.2.1 Variants of Tabu Search Algorithm

TS has several variants to tackle complex optimization problems. Jaeggi et al [41] developed Multi-Objective Tabu Search (MOTS) and Pareto Ranked Multi-Objective Tabu Search (PRMOTS) for continuous multi-objective optimization. Additionally, Chelouah and Siarry [13] introduced a hybrid method combining Continuous Tabu Search (CTS) with the Nelder–Mead Simplex algorithm to enhance global optimization of multiminima (multimodal functions) functions. These adaptations showcase the versatility of TS in optimizing various challenges.

1.4 Population-Based Metaheuristics

These algorithms operate on a population of candidate solutions, drawing inspiration from (1) *evolutionary biology* principles such as selection, mutation, and crossover; (2) *swarm intelligence* behaviors observed in flocks of birds, schools of fish, and colonies of ants; and (3) *other natural phenomena* like the immune system's response to pathogens. Let us now explore a few notable examples:

1.4.1 Genetic Algorithm

Genetic algorithms (GAs) are based on the natural reproduction and evolution of living organisms. Populations change over many generations according to "survival of the fittest" and natural selection theories. Holland [42] developed the basic ideas of GAs in detail, and they have been successful in resolving a wide range of combinatorial optimization issues.

Holland's concept is modified for the continuous case as follow: An initial population of n "individuals" is used by the algorithm at the beginning, each individual is made up of real coordinates that are correspondingly linked to the variables of the current objective function. This population uses the reproduction operators, which are motivated by genetics, children are produced from parents. The best individuals are chosen to constitute the new population. One wants to gradually enrich the population with the most productive individuals by repeating this process. The "crossover," which involves transferring certain coordinates between two individuals, and the "mutation," which involves creating a new coordinate at a specific location for one individual, are the typical methods of reproduction [12]. This process is illustrated in pseudocode 1.3.

Algorithme 1.3 Classical Genetic Algorithm.

1: Initialize necessary parameters;
2: Initialize a population of n individuals;
3: Evaluate the n individuals;
4: **while** The stopping condition is not satisfied: **do**
5: Use the selection operator to select K individuals;
6: Apply the crossover operator on the selected K individuals with probability P_c;
7: Apply the mutation operator on the individuals with probability P_m;
8: Use the evaluation operator to evaluate the offspring;
9: Use the selection operator to replace some parent individuals with offspring individuals;
10: **end while**
11: Return the best solution(s);

Genetic Algorithms (GAs) utilize various operators during the search process, includ-

ing encoding schemes, crossover, mutation, and selection. In the following, we will focus exclusively on the mutation and crossover operators.

1.4.1.1 Genetic Algorithm Crossover Operators

Crossover operators generate offspring by combining the genetic information of two or more parents. Here are some of the well-known crossover operators:

- **Heuristics Crossover ($\mathcal{HX}$):** It was introduced in [75]. This operator creates a biased offspring from a pair of parents by favoring the superior one. Specifically, an offspring $\mathbf{y} = (y_1, y_2, \ldots, y_n)$ is formed from parents $\mathbf{x^{(1)}} = (x_1^{(1)}, x_2^{(1)}, \ldots, x_D^{(1)})$ and $\mathbf{x^{(2)}} = (x_1^{(2)}, x_2^{(2)}, \ldots, x_D^{(2)})$. Therefore, each offspring element y_i is defined as:

$$y_i = u \cdot (x_i^{(2)} - x_i^{(1)}) + x_i^{(2)} \tag{1.1}$$

  where u represents a uniformly distributed random number within the interval $[0, 1]$, and the parent $x^{(2)}$ has a fitness value that is not worse than that of parent $x^{(1)}$.

- **Arithmetical Crossover:** This operator performs a linear combination of genetic information from two chromosomes (parent solutions) to produce offspring with new genetic traits[49]. It is expressed mathematically as :

$$y_i^{(1)} = \lambda x_i^{(2)} + (1 - \lambda) x_i^{(1)} \tag{1.2}$$

$$y_i^{(2)} = \lambda x_i^{(1)} + (1 - \lambda) x_i^{(2)} \tag{1.3}$$

  where $x_i^{(1)}$ and $x_i^{(2)}$ represent the i-th design variables of parents from the previous generation, and $y_i^{(1)}$ and $y_i^{(2)}$ denote the i-th design variables of the childs in the current generation. λ is random number within the range $[0, 1]$.

- **Laplace Crossover ($\mathcal{LX}$):** The Laplace crossover, introduced by the authors of [28] for real coded genetic algorithms (RGAs), utilizes the Laplace distribution (also known as the double exponential distribution). The Laplace distribution's probability density function (pfd) is defined as:

$$f(x) = \frac{1}{2b} \exp\left(-\frac{|x - a|}{b}\right), \quad -\infty < x < \infty \tag{1.4}$$

  and its cumulative distribution function (CDF) is given by:

$$F(x) = \begin{cases} \frac{1}{2} \exp\left(\frac{|x-a|}{b}\right) & \text{if } x \leq a; \\ 1 - \frac{1}{2} \exp\left(-\frac{|x-a|}{b}\right) & \text{if } x > a; \end{cases} \tag{1.5}$$

  where $a \in \mathbb{R}$ is the location parameter and $b > 0$ is the scale parameter.

In the Laplace crossover ($\mathcal{LX}$), two offspring are generated from a pair of parents $\mathbf{x^{(1)}} = (x_1^{(1)}, x_2^{(1)}, \ldots, x_n^{(1)})$ and $\mathbf{x^{(2)}} = (x_1^{(2)}, x_2^{(2)}, \ldots, x_n^{(2)})$ as follows: Initially, a uniform random number u is generated within the interval $[0, 1]$. Using the inverse of the Laplace distribution's cumulative distribution function, a Laplace-distributed random number β is computed:

$$\beta = \begin{cases} a - b\log_e(u) & \text{if } u \leq \frac{1}{2}; \\ a + b\log_e(u) & \text{if } u > \frac{1}{2}. \end{cases} \tag{1.6}$$

The offspring are then calculated using the following equations:

$$y_i^{(1)} = x_i^{(1)} + \beta \cdot |x_i^{(1)} - x_i^{(2)}| \tag{1.7}$$

$$y_i^{(2)} = x_i^{(2)} + \beta \cdot |x_i^{(1)} - x_i^{(2)}| \tag{1.8}$$

For smaller values of β, the offspring are expected to be generated close to the parents, whereas larger values of β lead to offspring that are from the parent solutions.

1.4.1.2 Genetic Algorithm Mutation Operators

Mutation is an operator that maintains genetic diversity from one population to the next. The well-known mutation operators are:

- **Non-uniform mutation:** Maintaining the same level of random mutations all over the evolutionary process could result in an ineffective search. In fact, while bigger mutations might be required early on to promote exploration, smaller mutations might be helpful later on to promote exploitation and improve the search. The Non-Uniform Mutation (NUM) operator for real-coded GA has been proposed in [48] to achieve this adaptive behavior. This operator is defined:
  Let $\mathbf{x}_t = x_1, \ldots, x_k, \ldots, x_D \in \mathbb{R}^D$ denote the real-coded vector representing the current solution at the t-th iteration (where t is the iteration counter), with x_k being the element selected randomly for variation through this mutation.
  NUM generates the offspring solution: $\mathbf{x}_{t+1} = \{x_1, \ldots, x'_k, \ldots, x_D\}$, where:

$$x'_k = \begin{cases} x_k + \Delta(t, U_k - x_k) & \text{if } \eta = 1; \\ x_k - \Delta(t, x_k - L_k) & \text{if } \eta = -1; \end{cases} \tag{1.9}$$

  The variable η is a random digit that takes either the value -1 or 1, and L_k and U_k are the lower and upper bounds of the variable x_k, respectively. The function $\Delta(t, y)$ given below takes value in the interval $[0, y]$. $\Delta(t, y) = y(1 - u^{1-\frac{t}{T}})^b$ where u is a uniformly distributed random number in the interval $[0, 1]$, T is the maximum

number of iterations, and b is a parameter, determining the strength of the mutation operator.

- **Direction-based Exponential Mutation:** Direction-based exponential mutation (DEM), introduced in [21], creates shifted solutions by using exponential functions and the directional information of the variables. It mostly consists of two parts: A. Obtaining directional information of the variables: The mutation operator uses variable directional information to guide algorithms to reach globally optimal solutions. It calculates average values in each generation, compares them with previous ones, and considers positive or negative directional information, except for the first and second generations.
  B. Generating mutated solutions using directional information and exponential functions: The mutation operator applies variable-wise independently, biasing mutated solutions to the search direction. A probability value of mutation P_m, directional probability P_d, is introduced to ensure the accuracy of the obtained information.
  - When directional information is positive
    A mutated solution $\mathbf{Y}_m$ is generated from a parent solution $\mathbf{Y}_p$ using equation (1.10) if a randomly generated number $\mathbf{r}_1$ (ranging from 0 to 1) is less than or equal to P_d. Otherwise, $\mathbf{Y}_m$ is derived from equation (1.2).

$$Y_m = Y_p + B1 \cdot (Y_u - Y_p) \tag{1.10}$$

$$Y_m = Y_p - B2 \cdot (Y_p - Y_L) \tag{1.11}$$

  - When directional information is negative
    The mutated solution $\mathbf{Y}_m$ is computed using equation (1.3) if $r_1 \leq P_d$. Otherwise, if $r_1 > P_d$, $\mathbf{Y}_m$ is obtained using equation (1.4).

$$Y_m = Y_p - B1 \cdot (Y_p - Y_L) \tag{1.12}$$

$$Y_m = Y_p + B2 \cdot (Y_u - Y_p) \tag{1.13}$$

    where Y_u and Y_L are the upper and lower limits of the variable,respectively.B1 and B2 represent two different perturbation values for mutation, calculated as follows:

$$b1 = e^{r^2} \cdot e^{(r - \frac{2}{r})} \tag{1.14}$$

$$b1 = e^{(r - r^2)} \cdot e^{(r - \frac{2}{r})} \tag{1.15}$$

- **Polynomial Mutation:** The polynomial mutation operator proposed in [27] perturbs a solution in the region of a parent using a polynomial probability distribution. The

mutation operator modifies the probability distribution to the left and right of a variable value so that no value is produced beyond the given range $[a, b]$. The mutant solution y_i for a given variable is constructed for a random integer u created within $[0, 1]$ for a given parent solution $x_i \in [a, b]$, as follows:

$$y_i = \begin{cases} x_i + \overline{\delta}_L \cdot (x_i - a_i), & \text{for } u \leq 0.5, \\ x_i - \overline{\delta}_R \cdot (b_i - x_i), & \text{for } u > 0.5. \end{cases} \tag{1.16}$$

Then, either of the two parameters ($\overline{\delta}_L$ or $\overline{\delta}_R$) is calculated, as follows:

$$\overline{\delta}_L = (2u)^{\frac{1}{1+\eta_m}}, \quad \text{for } u \leq 0.5 \tag{1.17}$$

$$\overline{\delta}_R = 1 - (2(1-u))^{\frac{1}{1+\eta_m}}, \quad \text{for } u > 0.5 \tag{1.18}$$

1.4.1.3 Genetic Algorithm Variants

Researchers have proposed numerous variants of Genetic Algorithms (GAs). These variants can be broadly categorized into five main types: real and binary coded GAs, multi-objective GAs, parallel GAs, chaotic GAs, and hybrid GAs. For more details about these categories and the approaches they encompass, refer to [43].

1.4.2 Differential Evolution Algorithm

Price and Storn [68] introduced Differential Evolution (DE) as an evolutionary algorithm tailored for optimizing continuous domain problems. DE evolves dynamically by adjusting its search strategy: initially, it introduces significant perturbations due to the large distances between parent populations. As evolution continues, populations converge, and perturbations decrease adaptively. This leads to focused exploitation in mature stages and extensive exploration in the early phases. DE's distinctive selection process creates each offspring by mutating and recombining current population members, directly comparing it with its parent [59]. For a deeper understanding of DE's principles, operators, and variants, refer to Chapter 02.

1.4.3 Particle Swarm Optimization Algorithm

In [44] Drs. Russell C. Eberhart and James Kennedy developed the Particle Swarm Optimization (PSO) technique as an approach inspired by how schools of fish and flocks of birds behave. In this algorithm each particle in a search space represents a possible solution. The location and velocity vector of the i^{th} particle in dimensional space can

be written as: $x_i = (x_{i1}, x_{i2}, \ldots, x_{iD})$ and $v_i = (v_{i1}, v_{i2}, \ldots, v_{iD})$, respectively. After the particles are randomly initialized the position and velocity of the particle indexed as i^{th} are updated in accordance with the following equations:

$$v_i(t+1) = wv_i(t) + c_1 r_1(p_i - x_i(t)) + c_2 r_2(p_g - x_i(t)) \tag{1.19}$$

$$x_i(t+1) = x_i(t) + v_i(t+1) \tag{1.20}$$

where w, the inertia's weight, can be utilized to regulate how much the old speed influences the current one. The weights of p_i and p_g are determined by two constants, c_1 and c_2. Whereas p_g indicates the best prior location of all the particles in the current generation, and p_i represents the best previous position of each particle. As a result, two independently produced random variables with uniform distributions in the interval $[0, 1]$ are represented by r_1 and r_2. To illustrate the previously indicated description, Algorithm 1.4 presents the pseudo-code.

Algorithme 1.4 Pseudo-code of Particle Swarm Optimization Algorithm.

1. Initialization
1.1. Find the initial population by clustering method
1.2. Calculate the objective function of initial solutions (f)
1.3. Set initial velocity vector equal to zeros
1.4. Set $P_b = f$, $G_b = \min(f)$, (G_b is Gbest and P_b is Pbest)
2. Repeat the following steps until the stopping condition is met
a) Update G_b and P_b:
for $i = 1 : PS$ (Population Size) **do**
    if $f_i < P_b(i)$ **then**
        $P_b(i) = f_i$
    end if
end for
$b = \min(f)$, (b is the best solution of the current generation)
if $b < G_b$ **then**
    $G_b = b$
end if
b) Generate nest population by using (2) and (3)
c) Checking the feasibility of generated solutions (f)
3. Report the best solution

1.4.4 Metaphor-based metaheuristics

With the rapid increase in metaphor-based metaheuristic algorithms in recent years, the prevalence of algorithms that are markedly similar yet differently named has become a common issue. This raises a critical question: can an optimization technique truly be

deemed 'novel' if its search characteristics are only slightly modified or nearly identical to those of existing methods? To address this question, a critique of metaphor-based metaheuristics was provided in [2] by optimization experts, who presented several arguments and points:

1. Useless metaphor: The use of metaphors to inspire numerous metaheuristics remains highly debated. A comprehensive compilation of "novel" algorithms (source: https://github.com/fcampelo/EC-Bestiary) indicates that developing metaheuristics which only partially mimics a real-world process may not merit their inclusion in the scientific literature. Furthermore, the mathematical model and the implementation of the algorithm often fail to accurately reflect the intended metaphor.
2. Lack of novelty: Evidence suggests that the rationale for introducing new metaphor-based metaheuristics based on innovative behavioral concepts is often unfounded. It is increasingly common to find that concepts claimed as novel in these metaheuristics have already been proposed in earlier publications, sometimes years earlier. What typically differentiates these "novel" metaheuristics is their adoption of non-standard terminology, which leads to misunderstandings in the literature, complicates understanding of existing metaheuristics, and hinders theoretical and experimental comparisons.
3. Poor experimental validation and comparison: Biased computational experiments can yield misleading results, often comparing the performance of "novel" metaheuristics run on newer processors against those on older ones, and/or limit experimentation, creating an inaccurate picture of performance . Additionally, rather than testing new metaheuristics against the best-performing algorithms on standard benchmarks, comparisons are frequently made with algorithms, which are far from the state-of-the-art.

1.5 Conclusion

This chapter has provided the foundation for understanding what optimization is. We have discussed the difference between exact methods and approximate methods used to solve these problems. Additionally, we have explored single-solution metaheuristics and population-based metaheuristics in detail, emphasizing their importance in addressing various optimization challenges.

In the next chapter, we will explore various versions of the Differential Evolution (DE) algorithm in detail and, clarifying their specific applications.

Chapter 2

Differential Evolution Variants

2.1 Introduction

With all its nuances, novelties, and abstract computational structure, the differential evolution algorithm takes the idea of evolutionary computation to the next level. Arguably one of the most formidable meta-heuristic algorithms for tackling complex real-world problems, DE has undergone significant development since the publication of the first comprehensive survey article by Das and Suganthan in 2011. Over the past 13 years, numerous advancements have emerged across various facets of the algorithm, culminating in an impressive state of research and application. Unlike several other evolutionary computation techniques, basic DE stands out for its simplicity, requiring only a few lines of code in any standard programming language for implementation. Moreover, the canonical DE boasts minimal control parameters—precisely three: the scale factor, the crossover rate, and the population size—rendering it easily accessible to practitioners.

In this chapter, we explore various DE variants and their practical uses in solving optimization problems. We will look into the details of some of these variants, including adaptations of parameters, new mutation and crossover strategies, and hybrid methods.

2.2 Principle of Differential Evolution

Price and Storn [68] introduced Differential Evolution (DE) as a kind of evolutionary algorithm for solving optimization problems across a continuous domain. DE's key principle is to modify the search strategy as it grows. Due to the parent populations' great distances from one another, there are significant perturbations early in the evolution. The population converges to a small region when the evolutionary process reaches maturity, and the perturbations reduce to an adaptive level. Consequently, the evolutionary algorithm performs a local exploitation in the mature stage of the search and a global exploratory

search in the early phases of the evolutionary process. In Differential Evolution (DE), the process of selecting individuals for the next generation is unique compared to other evolutionary algorithms. each offspring (child) is generated by mutating and recombining the current population members. After creating an offspring, it is directly compared to its corresponding parent (the specific individual from which it was generated). This means the offspring does not compete with the entire population but only with its direct predecessor (the parent) [59]. The basic DE algorithm is presented in Pseudo-code 2.1.

Algorithme 2.1 Basic Differential Evolution (DE) Algorithm.

$t \leftarrow 1$
Initialize $P_t = \{x_{1,t}, ..., x_{N,t}\}$ randomly
while The termination criteria are not met **do**
    for $i \in \{1, ..., N\}$ **do**
        Generate the mutant vector $v_{i,t}$ using a mutation strategy with F
        Generate the trial vector $u_{i,t}$ by crossing $x_{i,t}$ and $v_{i,t}$ using a crossover method with CR
    end for
    for $i \in \{1, ..., N\}$ **do**
        if $f(u_{i,t}) \leq f(x_{i,t})$ **then**
            $x_{i,t+1} \leftarrow u_{i,t}$
        else
            $x_{i,t+1} \leftarrow x_{i,t}$
        end if
    end for
    $t \leftarrow t + 1$
end while

2.3 Differential Evolution operators

Various types of crossovers and mutations operators are employed in Differential Evolution (DE), each playing a crucial role in the optimization process to to effectively explore and exploit the search space. Among them, the $DE/rand/1$ mutation and binomial crossover are widely recognized as two of the most commonly used strategies.

- $DE/rand/1$ **Mutation**: The differential evolution algorithm's mutation strategy is represented by the notation $DE/x/y$ where x stands for the reference vector in the mutation operation, y for the number of differential vectors in the mutation operation, and DE for differential evolution algorithm. The most popular method of mutation is to take two individuals at random from the population, measure the differences in their vectors, and then use another random individual to do vector

creation. The following is the obtained mutation individual V_i:

$$V_i^{G+1} = X_{r1}^G + F \cdot (X_{r2}^G - X_{r3}^G) \tag{2.1}$$

where r1, r2, and r3 are randomly generated integers ranging from 1 to NP, and $r1 \neq r2 \neq r3 \neq i$; G represents the current generation number; and F denotes the scaling factor.

- **Binomial Crossover (uniform)**: Following the mutation phase, a crossover operation is performed to produce a trial vector for each pair of the target $X_{i,G}$ and the matching mutant vector $V_{i,G}$:

$$U_{i,G} = (u_{i,G}^1, u_{i,G}^2, ..., u_{i,G}^D)$$

In the basic version, DE employs the binomial (uniform) crossover defined as follows:

$$u_{i,G}^j = \begin{cases} v_{i,G}^j, & \text{if } (\text{rand}_j \leq CR) \text{ or } (j = j_{\text{rand}}) \\ x_{i,G}^j, & \text{otherwise} \end{cases} \tag{2.2}$$

where the crossover rate CR, a user-specified constant in the interval [0, 1], determines the percentage of parameter values that are replicated from the mutant vector, and $j = 1, 2, ..., D$. An integer j_{rand} is selected at random from the set [1, D]. If $rand_j[0, 1] \leq CR$ or $j = j_{rand}$, then the binomial crossover operator duplicates the j^th parameter of the mutant vector $V_{i,G}$ to the corresponding element in the trial vector $U_{i,G}$. If not, it is taken from the matching target vector $X_{i,G}$.

- **Exponential Crossover**: The exponential crossover was created to resemble the one-point and two-point crossover variants found in genetic algorithms. As a result, the trial vector includes a sequence of consecutive components (in a circular manner) taken from the mutant vector[78]. The structure of the trial vector can be described as shown in Equation where j is random number between (1,D) and D is the dimension

.

$$U_{i,j}^t = \begin{cases} V_{i,j}^t, & \text{where} j = 1 + j \bmod D \text{if rand}[0,1] < CR \text{ or } j = D - 1; \\ x_{i,t}^j, & \text{otherwise,} \end{cases} \tag{2.3}$$

- **Selection**: The target vector X_i^G and the trial vector $U_{i,j}^{G+1}$ are compared using the greedy selection approach in DE, and the vector with the better fitness value is chosen.

$$X_i^{G+1} = \begin{cases} U_i^{G+1}, & f(U_i^{G+1}) \leq f(X_i^G) \\ X_i^G, & \text{otherwise} \end{cases} \tag{2.4}$$

where $f(.)$ stands for the fitness value.

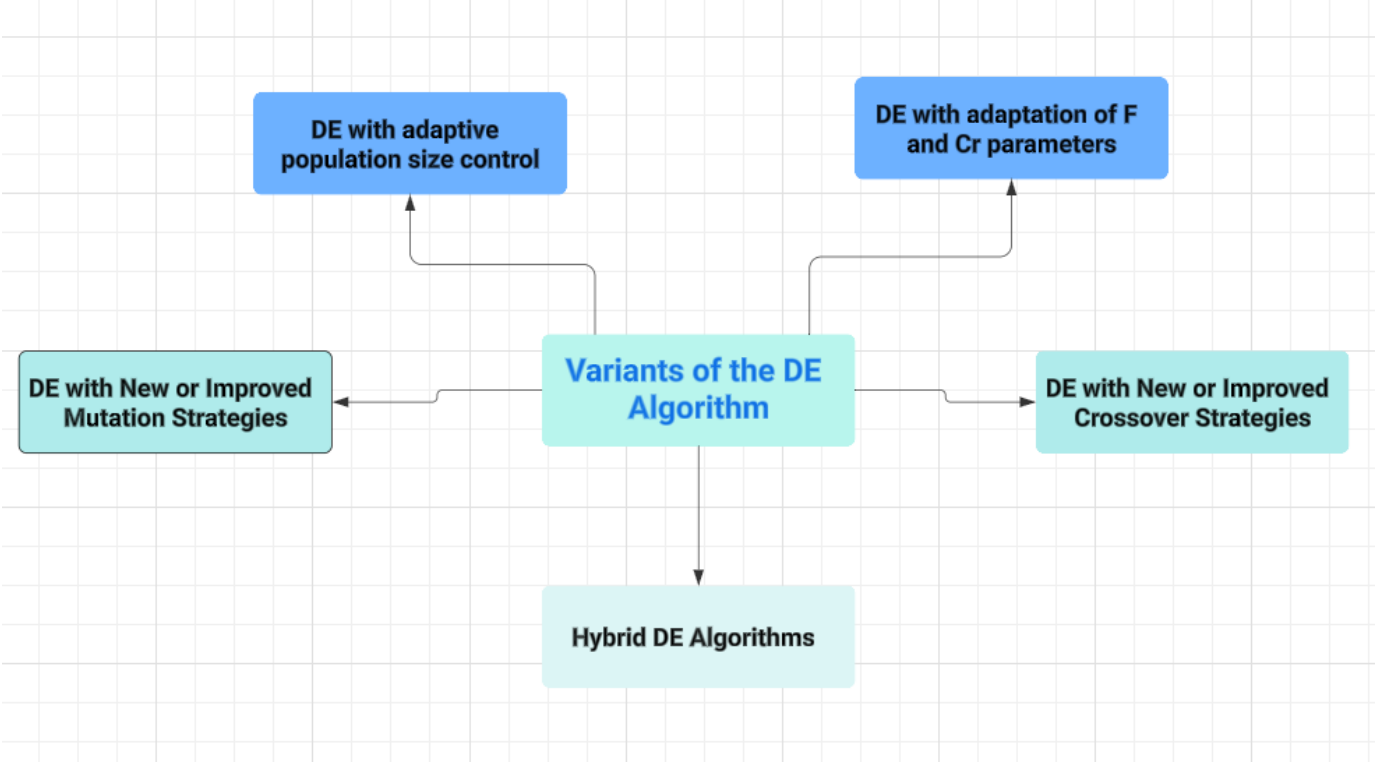


Figure 2.1: Variants of DE algorithm.

2.4 Differential Evolution Variants

2.4.1 Differential Evolution with Adaptation of F and CR Parameters

A DE variation with an adaptive parameter control mechanism was reported in [30]. The trial vector is generated using the $DE/rand/1/bin$ strategy (two vectors are randomly selected from the population, while the base is selected from the solutions in the range of top 10% to 40% performing individualsand the authors defined two sets of allowable values for F and Cr. For every combination of F and Cr values in the set, a counter is first assigned and then reset to zero. A trial vector is created by randomly selecting a set of control parameters; if that combination is chosen to go to the next generation, the counter associated with that combination is incremented by one. The top half of the performing combinations are chosen at each generational break, their counters are reset to zero and the remaining combinations are deleted. The iteration count and combination counters are reset to zero, signaling a restart, if the number of generations surpasses a certain threshold.

The authors in [63] enhanced the previously described work by Elsayed et al. [30] by dynamically identifying and utilizing the optimal set of DE control parameters, namely NP, F, and Cr. The value of DE is reduced by the algorithm following a recurring sequence of generations. While the remaining solutions are archived, the superior solutions

are retained in the smaller population. The DE corresponding to the best improved population is chosen as the current population size when the least value of DE is attained. The average fitness improvement by the populations with varying numbers of members is then determined. The maximum number of people needed from the archive is added to the population. The mutation scheme used by Elsayed et al. [30] remains unchanged.

Das et al [25] proposed a straightforward DE strategy to solve large-scale optimization problems where the values of the F and Cr are switched uniformly randomly between two extreme corners of their feasible ranges for different population members. This approach avoids the need for complex parameter adaptation schemes or the addition of additional local search methods. Additionally, every member of the population is modified using either the $DE/best/1$ scheme or the $DE/rand/1$ method. The population member is exposed to the same mutation technique that produced the previous successful update at the same population index. On the high-dimensional functions generated under the CEC (Congress on Evolutionary Computation) 2008 and 2010 competition on large-scale global optimization, this technique demonstrated very competitive performance. The findings show that solving benchmark functions can benefit greatly from combining the low value (=0.5) with the large and unusual value of F (= 2). The work of Das et al.[25] indicates that much of the valuable information about F and Cr values can remain attached to the borders of their feasible regions, in contrast to the standard DE algorithms that sample F values from the interval of (0.4, 1) and Cr values from (0, 1).

2.4.2 Differential Evolution with Adaptive Population Size Control

Three options for parameter adaptation were proposed together with a new population reduction technique by Brest and Maučec [9]. The authors refer to these techniques as jDEbin (self-adaptive DE/rand/1/bin), jDEexp (self-adaptive DE/rand/1/exp), and jDEbest (self-adaptive DE/best/1/bin). They all make use of the fundamental jDE (Brest et al., [8]) type randomization of F and Cr. One of these three approaches is used in every algorithm iteration. The population reduction mechanism essentially lowers the size of DE's population on a predetermined timeline. The computational budged is separated into periods more precisely. Each phase begins with a halving of the population. Each individual is permitted to compete with another individual who follows the same offspring generation strategy as the previous one in the suggested population reduction process. By doing this, the DE seems to gradually narrow down the search and avoid the undesirable stagnation effect.

Yang et al. [77] suggested a technique to modify the same based on population diversity. Using the pair-wise Euclidean distance between each individual, their method may determine whether there is insufficient diversity in the population. Once the right

time has been found, the population is regenerated to increase diversity and reduce the likelihood of stagnation. However, because Euclidean distances must be calculated multiple times, this method can be somewhat costly for high-dimensional problems.

2.4.3 DE with New or Improved Mutation Strategies

In order to reduce the constraint violation of new solutions produced during the searching process, Hamza et al. [37] suggested a new mutation technique that was influenced by the constraint consensus (CC) method and integrated into DE. The CC technique actively participated in the mutation process to lower the amount of new solutions that broke the predefined constraints, in contrast to the majority of previous works that only took the constraint violation issue into consideration during the ranking and selection processes of solutions. During the search phase, the CC-based mutation technique was notably limited to specific unfeasible solutions in order to retain population variety while reducing runtime. According to simulation tests, the CC-based mutation technique can drastically reduce runtime up to 44.7% and 10.6% for problems with dimensional sizes of 10 and 30, respectively, while improving DE in terms of solution quality. However, because of the quick loss of population variety in the initialization phase, which determines the possible search area of specified optimization issues, the CC-based mutation technique is not effective in handling multimodal problems.

According to Gong and Cai (2013) [35], a mutation may be more successful if the base vector and one of the difference vector's terminal vectors are chosen at random, while the third is chosen based on fitness. Nevertheless, a vector will only have a probability of being chosen as a base or a terminal in order to reduce the process's greed. The authors suggested that the chance of a vector being chosen in a mutation can be computed based on its fitness rank in the population, rather than using randomized or proximity-based approaches. Thus, the suggested approach requires less computing power and is quicker. It is necessary to sort the population in ascending order of fitness values, with the best individuals at the top of the list. The rank value R_i for the i^{th} solution is thus defined as NP - i. Then, we define $p_i = \frac{R_i}{NP}$ as the probability of choosing the i^{th} solution. The first two vectors are picked in proportion to their likelihood of selection when three are chosen for the DE/rand/1 method.

By referring to their individual cumulative scores, Sun et al. [69] changed standard mutation schemes and presented a novel Gaussian mutation in their proposed Gaussian Mutation and Dynamic Parameter Adjustment Differential Evolution (GPDE) to develop new mutant vectors in a collaborative and adaptive manner. The scaling factor needed to achieve the right balancing of exploration and exploration strengths was generated using a periodic function. The fluctuant crossover rate derived from the Gaussian function was

utilized to further improve the population diversity of GPDE and guarantee its strong performance when handling intricate challenges.

2.4.4 DE with New or Improved Crossover Strategies

A variance-based DE method with an optional crossover (VDEO) was suggested by Alswaitti et al. [1] to handle data clustering optimization problems with better convergence speed and solution quality. Initially, VDEO used a single-solution representation strategy to get around the drawbacks of population-based solution strategies for formulating and initializing the clustering optimization issues. To properly balance exploration and exploitation searches, a vector-based mutation factor estimation and a switchable scheme comprising of two mutation methods were then integrated into VDEO. The suggested method sped up the algorithm's convergence by comparing the fitness of the mutant and trial solutions before moving on to the selection stage, taking into account the possibility that a mutant solution would be more fit than its trial counterpart.

An epistatic arithmetic crossover operator, or eXEDE (Epistatic Arithmetic Crossover Operator based on Cartesian Graph Product in Ensemble Differential Evolution), was introduced by Fister et al.[32] into an ensemble DE variant. The impact of epistatic genes in the context of evolutionary computation was taken into consideration by the epistatic arithmetic operators, in contrast to the standard arithmetic crossover, by expressing the epistatic as a graph product of two linear graphs represented by the candidate solutions that are involved in the crossover process.

In order to address large-scale continuous optimization challenges, Mohamed et al. [52] introduced a DE version called as adaptive novel DE (ANDE). In order to strike the right balance between improved convergence speed and effective population variety preservation, a new adaptation strategy was added to ANDE. This scheme allows each solution's crossover rate to be adjusted based on its historical searching experience.

2.4.5 Hybrid DE Algorithms

Hybridization is another popular method used to improve DE's search performance by combining the strengths of search operators from other computational intelligence algorithms. Here are some examples of this approach:

2.4.5.1 Differential Evolution with Artificial Neural Networks

Mason et al. [47] introduced a hybrid approach termed multiobjective ANN (Artificial Neural Network) with DE to lower the system's cost and emissions. It is capable of

solving DEED (Dynamic Economic Emission Dispatch) problems and approximating them simultaneously.

A hybrid surrogate model of DE and ANN was created by Saporetti et al. [62] to automatically classify petrophysical data and enhance reservoir characterization methods in the oil sector.

2.4.5.2 Differential Evolution with Particle Swarm Optimization

To optimize the PID (Proportional-Integral-Derivative) controller's parameter settings, Moharam et al. [53] introduced a hybrid algorithm called aging leader and challengers PSO and DE (ALC-PSODE). ALC-PSODE's adoption of the aging leader and challenger principles was helpful in resolving PSO and DE's premature convergence problems, resulting in a PID controller with reliable performance.

Song et al. [67] improved the design of a 3D wind turbine system using PSO and DE to maximize output power and reduce costs.

2.4.5.3 Differential Evolution with with Genetic Algorithm

Hybridizing GA with DE as hGADE, Trivedi et al. [73] solved the unit commitment scheduling problem. To further improve hGADE performance, a heuristic was added to the population initialization approach.

To solve cloud computing applications, Li et al. [46] presented a hybridization of GA and DE as part of a multi-objective optimization algorithm.

2.5 The Applications of Differential Evolution

Differential Evolution (DE) finds extensive application across various domains, owing to its robustness and versatility in solving complex optimization problems, as presented in Table 2.1.

Domains of Application	**Problems**	**Algorithms Name**	**References**
Pattern Recognition and Image Processing	Image Watermarking	DE/rand/1/bin	[3]
		DE/target-to-best/1/bin	[3]
	Image registration and enhancement	DE/rand/1/bin	[31]
		DE with chaotic local search	[17]
	Data clustering	DE/rand/1/bin	[58]

Domains of Application	**Problems**	**Algorithms Name**	**References**
		DE with neighborhood-based mutation	[24]
		DE with random scale factor and time-varying crossover rate	[23]
Control Systems and Robotics	Controller design and tuning	Self adaptive DE	[55]
		DE/rand/1 with arithmetic crossover	[40]
		DE/rand/1/bin with random scale factor and time-varying C_r	[6]
	Optimal control problems	DE/rand/1/bin and DE/best/2/bin	[18]
		modified DE with adjustable control weight gradient methods	[19]
	System identification	Conventional DE/rand/1/bin	[14] [11]
Bioinformatics	Gene regulatory networks	DE with adaptive local search	[56]
		hybrid of DE and PSO	[76]
	Protein folding	DE/rand/1/bin	[66]
	Bioprocess optimization	DE/rand/1/bin	[54]
Electrical Power Systems	Economic dispatch	DE/rand/1/bin	[57]
		variable scaling hybrid DE	[15]
	Capacitor placement	Hybrid of ant system and DE	[16]
	Optimal power flow	DE/target-to-best/1/bin	[10]
		DE with random localization	[64]

Table 2.1: Classification of DE Applications [26].

2.6 Conclusion

This chapter has examined various versions of the Differential Evolution (DE) algorithm. We have looked closely at each variant, discussing how parameters can be adapted, new mutation and crossover strategies, and how hybrid approaches are used. Additionally, we have explored where these different DE variants are applied, showing the specific problems they can solve.

In the next chapter, we will present an enhanced version of the DE algorithm.

Chapter 3

Contribution: Boosting Differential Evolution Performance

Introduction

The Differential Evolution (DE) algorithm remains a novel and ever-evolving method, which researchers find interesting due to its potential for further improvement. In this chapter, we aim to enhance the algorithm's performance.We combines different strategies from various variants to leverage the advantages of each, focusing on different aspects of DE such as its operators and parameter setting. We will present the motivations behind this approach, provide a detailed explanation of the techniques used and discuss of the obtained results.

3.1 Motivation

Most variants of the DE algorithm evolve a single population using specific DE operators. However, as nature has shown, the concept of work specialization—whereby a group is divided into multiple smaller groups, each assigned a specific task based on its capabilities—can enhance working efficiency, akin to the division of tasks among different types of bees in a colony. Worker bees perform specialized roles such as foraging for nectar and pollen, nursing larvae, and defending the hive. This division of labor significantly boosts the overall efficiency and survival of the colony. Drawing inspiration from this phenomenon, [20] divides the population into three sub-populations according to their fitness values. Each sub-population is assigned specific tasks tailored to the capabilities of its members. The primary reason for this division is to apply different mutation strategies to each sub-population according to their characteristics. By doing so, we ensure a proper balance between exploration (searching for new solutions) and exploitation (refining existing solutions). In our approach, we have adopted a similar strategy.

On the other hand, since the performance of DE is highly sensitive to its associated

control parameters, numerous Parameter Control Methods (PCMs) have been proposed for adjusting the scale factor and crossover rate to enhance DE performance. Based on experiments conducted in [72], which extracted 24 PCMs for the scale factor and crossover rate from their original complex DE algorithms and provided an in-depth benchmarking study of them, we observed that generating F and CR from COBIDE (noting that this method was proposed in DE variant called COvariance matrix learning and BImodal distribution parameter setting (COBIDE) [74]) yielded superior results. Consequently, we integrate COBIDE into our proposed variant as an integral component, enabling our variant to dynamically adjust F and CR.

3.2 Description of the proposal

The following is a description of our proposal, termed the Cauchy-Partitioned Differential Evolution algorithm (CPDE), which features parameter control based on the Cauchy distribution.

After all the individuals are sorted in ascending order based on their fitness values, the entire population is divided into three sub-populations, referred to as inferior, medium, and superior sub-populations, as stated in the motivation. The rationale behind this division is as follows. The inferior sub-population, which includes individuals with better fitness values that may be close to the global optimum, is responsible for exploitation. A tiny perturbation is performed in the local area of the inferior sub-population to find a better local or global optimum point. The medium sub-population, consisting of individuals with moderate fitness values, balances between exploration and exploitation by performing moderate perturbations. This helps to maintain diversity while refining potential solutions. The superior sub-population, composed of individuals with worse fitness values that may be far from the global optimum, undertakes the task of exploration. These individuals generally undergo larger perturbations to escape the local region and approach a more promising area.

Heterogeneous mutation strategies are used to achieve the task of each sub-population. Specifically, the worst individuals, constituting the superior sub-population, employ the $DE/rand/1$ strategy (see Equation 3.1) to enhance exploration by making large perturbations. Although this mutation has a slow convergence rate, it has a significant exploration potential to avoid premature as indicated in [61].

$$v_i = x_{r1} + F \cdot (x_{r2} - x_{r3}) \tag{3.1}$$

The best individuals, in the inferior sub-population, adopt the $DE/best/1$ strategy (see Equation 3.2) which has proven to have a quick convergence rate and excellent perfor-

mance. It employ the best solution found so far to conduct additional research.

$$v_i = x_{\text{best}} + F \cdot (x_{\text{r1}} - x_{\text{r2}}) \tag{3.2}$$

The medium sub-population uses the $DE/rand-to-best/1$ strategy (see Equation 3.3), This mutation combines elements from both the $DE/rand/1$ and $DE/best/1$ strategies. By doing so, it uses the difference between randomly selected individuals to introduce variability (exploration) while also incorporating information from the best solution found so far to refine the search (exploitation).

$$v_i = x_i + F \cdot (x_{Best} - x_i) + F \cdot (x_{r1} - x_{r2}) \tag{3.3}$$

As such, with the heterogeneous mutation schemes used, our variant is expected to strike to attain a favorable equilibrium between exploitation and exploration in order to be able to achieve good optimization performance.

After each sub-population generates its own mutant vector, we apply a repair strategy to ensure the solutions remain within bounds. This strategy involves checking each component of the vector: if a component is less than the lower bound, it is adjusted to the lower bound; if it exceeds the upper bound, it is adjusted to the upper bound.

Next, the mutant vector is combined with a target solution (parent vector) using binomial crossover (Note that we experimented with both binomial and exponential crossover to create two variants.) to produce a trial solution. This process generates offspring that inherit beneficial traits from both parents, potentially leading to improved solutions (see Chapter1 for more details).

At the end of the evolutionary process, a selection operator is applied to determine which individuals survive to the next generation. In our proposed variant, we use greedy selection, which involves comparing the objective function values of two competing solutions (the target vector and the trial vector). If the trial solution improves upon the target solution, it is accepted; otherwise, the target vector is retained.

As for the parameter settings, a Bimodal Distribution is incorporated into the control parameters of the mutation and crossover operators, following the same approach used in COBIDE. We first randomly generate F and CR values from a Bimodal Distribution composed of two Cauchy Distributions (see Equations 3.4,3.5). Subsequently, in each iteration, the algorithm updates its parameters such that successful pairs of F and CR values are inherited by each individual in the next iteration. Conversely, unsuccessful pairs of F and CR parameter pairs are replaced with newly generated values.

$$F_{i,t} = \begin{cases} \text{randC}(0.65, 0.1) & \text{if randU}(0,1) < 0.5 \\ \text{randC}(1.0, 0.1) & \text{otherwise} \end{cases} \tag{3.4}$$

$$CR_{i,t} = \begin{cases} \text{randC}(0.1, 0.1) & \text{if randU}(0,1) < 0.5 \\ \text{randC}(0.95, 0.1) & \text{otherwise} \end{cases} \tag{3.5}$$

Regarding the population size N, it is determined by $N = 5{*}dimension$ if $dimension >= 5$ else 20, as used in [72]. For problems with a dimension of 5 or more, the population size is set to 5 times the dimension. This scaling ensures that as the problem becomes more complex (with increasing dimensions), the population size increases proportionally. A larger population helps maintain diversity and improves the chances of finding a global optimum in high-dimensional spaces. For problems with dimensions less than 5, the population size is fixed at 20. This ensures that even for lower-dimensional problems, the population size is sufficiently large for effective exploration and exploitation of the search space.

The principle described above is illustrated in pseudo-code 3.1.

3.3 Description of the platform used in the experiments

3.3.1 Benchmark functions

An essential task in optimization research is identifying the strengths and weaknesses of existing algorithms and comparing their performance, which in turn helps improve the design of new algorithms. This task has been significantly automated in recent years with the development of tools like the Comparing Continuous Optimizers platform (COCO). COCO is an open-source platform designed for comparing continuous optimizers in a black-box setting. It aims to automate the tedious and repetitive task of benchmarking numerical optimization algorithms to the greatest extent possible. The platform and its underlying methodology allow benchmarking of both deterministic and stochastic solvers for single and multi-objective optimization within the same framework. Additionally, COCO provides a growing archive of comparative benchmarking datasets to the scientific community [38].

A testbed or benchmark suite consists of a collection of problems across various dimensions, typically ranging from 1000 to 5000 problems (number of dimensions × number of functions × number of instances). The COCO framework provides several test suites:

1. **BBOB**: This suite includes 24 functions with 15 instances for each function across dimensions 2, 3, 5, 10, 20, and 40. The functions are categorized into five subgroups:

Algorithme 3.1 Cauchy-Partitioned Differential Evolution algorithm (CPDE)

Initialize:
Define dimension D and population size N.
Initialize population $P = \{x_1, \ldots, x_N\}$ randomly within bounds.
Initialize fitness values $f(x_i)$ for each x_i in P.
Initialize function evaluation counter func_eval $\leftarrow N$.
while termination criteria are not met **do**
    Sort population P by fitness values $f(x_i)$.
    Divide population into sub-populations: inferior, medium, superior.
    Generate adaptive parameters F and CR for each sub-population.
    for each individual x_i in P **do**
        Mutation:
        if x_i belongs to the inferior sub-population **then**
            $v_i \leftarrow$ best1_mutation_strategy$(P, \text{best_individual}, F_i)$.
        else if x_i belongs to the medium sub-population **then**
            $v_i \leftarrow$ rand_to_best1_mutation_strategy$(P, \text{best_individual}, i, F_i)$.
        else if x_i belongs to the superior sub-population **then**
            $v_i \leftarrow$ rand1_mutation_strategy(P, i, F_i).
        end if
        Repair:
        Apply repair strategy to v_i.
        Crossover:
        $u_i \leftarrow$ binomialCrossover(x_i, v_i, CR_i).
        Evaluation:
        Compute $f(u_i)$.
        Selection:
        if $f(u_i) \leq f(x_i)$ **then**
            Replace x_i with u_i.
        end if
        Increment function evaluation counter:
        func_eval $\leftarrow$ func_eval $+ 1$.
    end for
    Update adaptive parameters:
    Update F and CR based on success rates.
end while
Return best solution and fitness:
best_solution $\leftarrow$ argmin $f(x_i)$.
best_fitness $\leftarrow \min\ f(x_i)$.
End of Algorithm

separable (f1 to f5), moderate (f6 to f9), ill-conditioned (f10 to f14), multi-modal weakly structured (f15 to f19), and multi-modal with global structure (f20 to f24).

2. **BBOB-largescale**: This suite also includes 24 functions categorized in the same way as the BBOB suite, but specifically designed for larger dimensions: 20, 40, 80, 160, 320, and 640.
3. **BBOB-mixint**: This suite comprises 24 mixed-integer single-objective functions across dimensions 5, 10, 20, 40, 80, and 160. It follows the same subgroup categorization as the BBOB suite.

In our study, we benchmark our proposed variants as a solvers using the BBOB suite of benchmark functions

3.3.2 Symbols, Constants, and Parameters

- f_{opt}: Optimal function value, defined individually for each benchmark function.
- Δf: Precision to reach, representing the difference to the smallest possible function value f_{opt}.
- $f_{target} = f_{opt} + \Delta f$: Target function value to achieve. The final, smallest considered target function value is $f_{target} = f_{opt} + 10^{-8}$, but larger values for f_{target} are also evaluated.
- $N_{trial} = 15$: Number of trials for each single setup, i.e., each function and dimensionality. A different function instance is used in each trial. Performance is evaluated over all N_{trial} trials.
- $D = 2, 3, 5, 10, 20, 40$: Search space dimensionalities used for all functions. Dimensionality 40 is optional and can be omitted according to the to the benchmark criteria.

3.3.3 PostProcessing

In this step, COCO collects runtime data to generate various HTML figures and comparison tables. Post-processed data from over 300 officially supported algorithm datasets is available on the COCO platform for various test suites.
Postprocessing is a straightforward process that takes several minutes to generate results. The COCOPP Python package needs to be installed first. After installation, execute the required commands from a Python shell.

3.3.4 Algorithms used in comparison

The following algorithms have been used in comparison, along with their parameters.

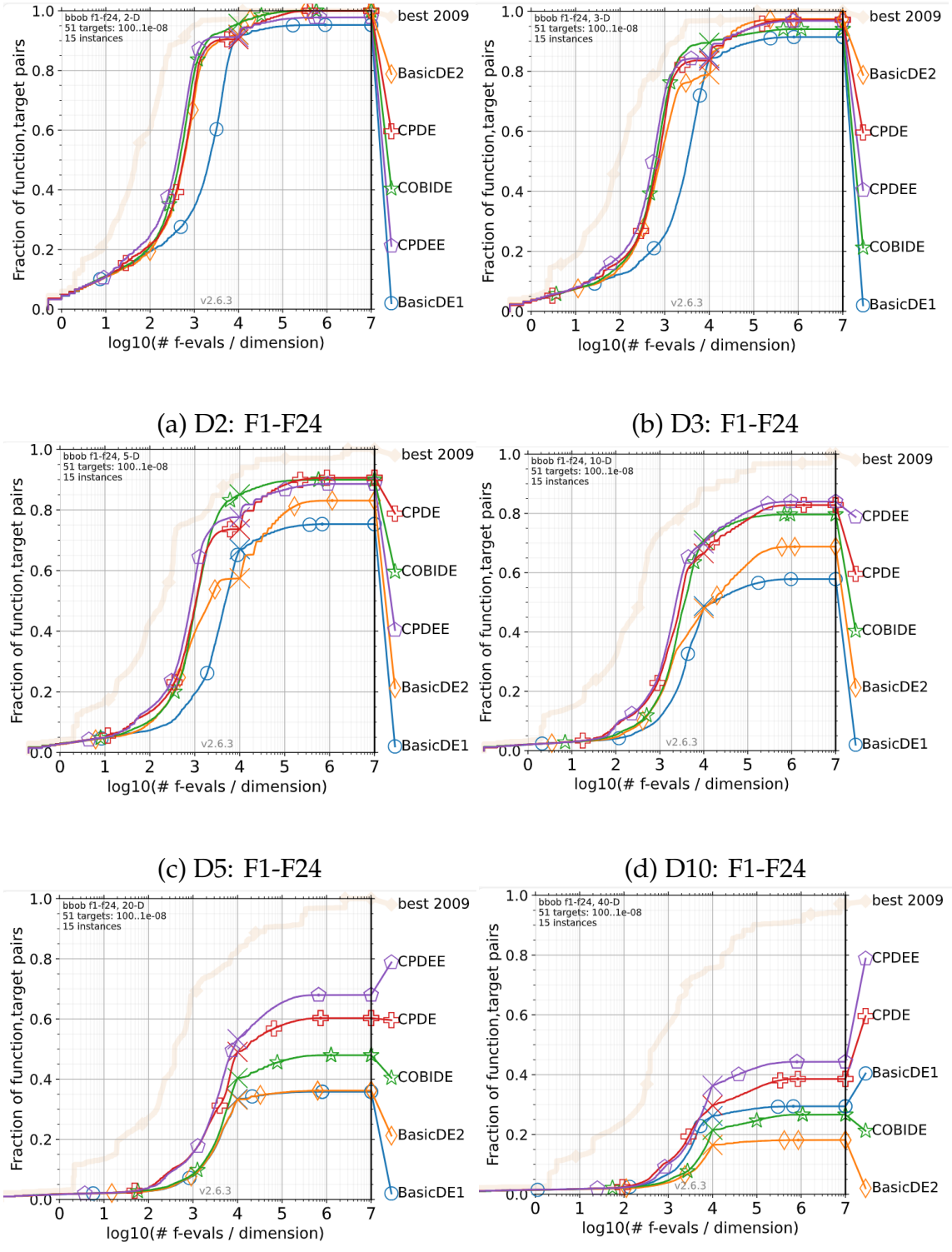
1. BasicDE1: This is a straightforward implementation of the Differential Evolution (DE) algorithm with fixed parameters: Population Size: 100, Scaling Factor (F): 0.5, and Crossover Rate (CR): 0.9. The values of F and CR are based on recommendations from Storn (1997) [68]. BasicDE1 utilizes the DE/rand/1 mutation strategy and binomial crossover.
2. BasicDE2: This variant shares the same parameters as BasicDE1 but dynamically adjusts the population size based on the problem dimension: Population Size: 5×dimension 5×dimension if the dimension is 5 or greater; otherwise, it is set to 2.
3. COBIDE: Based on BasicDE2 with an incorporated parameter control method of F and CR [74].
4. CPDE: Our proposed variant that incorporates binomial crossover. It partitions the population into three sub-populations, each utilizing a specific mutation strategy. CPDE also integrates COBIDE for dynamically generating F and CR values. The population size is determined as in COBIDE.
5. CPDEE: Another proposed variant similar to CPDE but uses exponential crossover instead of binomial crossover.

Note that for BasicDE1 and BasicDE2 algorithms, we implemented them from scratch using the COCO framework, generating the dataset ourselves. In contrast, for COBIDE, we utilized an existing dataset that we found.

3.4 Experimental Results and discussion

In the following, we will interpret and discuss the experimental results of our study, summarized by Figure 3.1 and the table **??**, across dimensions 2, 3, 5, 10, 20, and 40. All algorithms in comparison use the same budget of $FEs = 10000 * D$. The graphs in the figure display the empirical cumulative distribution functions (ECDF) of predicted running times (ERT), which are expressed as function evaluations divided by dimension. The 'Best 2009' line represents the best ERT (Expected Running Time) recorded since BBOB 2009.

The performance resulting from the comparison of the BasicDE1, BasicDE2, and CPDE and COBIDE algorithms in dimensions 2 and 3 is illustrated in Figures 3.1 (a) and (b). We observe that our variant shows performance levels that are closely comparable with all other algorithms. While CPDE and BasicDE2 consistently outperform BasicDE1. This can be attributed to the fact that CPDE and BasicDE2 utilize a smaller population size (20 individuals) compared to BasicDE1, which employs a larger fixed population size of



(a) D2: F1-F24

(b) D3: F1-F24

(c) D5: F1-F24

(d) D10: F1-F24

(e) D20: F1-F24

(f) D40: F1-F24

Figure 3.1: Empirical cumulative distribution functions (ECDFs) of the Expected Running Time (ERT) for the algorithms BasicDE1, BasicDE2, COBIDE, CPDE, and CPDEE for dimensions 2, 3, 5, 10, 20, 40

100 individuals. This highlights the significant effect of smaller populations in enhancing the efficiency of exploration and exploitation within the search space. CPDE and BasicDE2 capitalize on this strategic advantage, enabling them to converge more rapidly and achieve higher success rates earlier in the optimization process compared to BasicDE1. Additionally, our variant appears to have similar performance to COBIDE.

In (c), which compares the performance across dimension 5, it is notable that all three algorithms experience a deterioration in performance. This can be attributed to the overall increase in problem complexity with higher dimensions. Despite this challenge, CPDE typically outperforms BasicDEs by solving approximately 75% of the problems. The difference between CPDE and COBIDE in this dimension is consistent with the previous dimension.

From the results presented in (d) and (e) for dimensions 10 and 20, we observe a significant impact on the performance of our variant CPDE compared to BasicDEs. In dimension 10, CPDE solves over 65% of the problems, while BasicDEs solve 45%. Similarly, in dimension 20, CPDE solves 50% of the problems, whereas BasicDEs solve 33% .This failure of BasicDEs can be attributed to its lack of adaptive parameters.

In the other hand, both CPDE and COBIDE exhibit similar trends in their performance in dimension 10, but with some notable differences. At the budget of 10^4 function evaluations, COBIDE shows a slight advantage over CPDE. However, as the budget increases, CPDE begins to outperform COBIDE.

In dimension 20, the difference between their performances becomes more pronounced. CPDE demonstrates approximately a 10% better performance compared to COBIDE within the same budget range. Furthermore, CPDE maintains a higher fraction of function/target pairs achieved across almost the entire range of function evaluations compared to its competitors, highlighting its superior effectiveness in this higher dimensions.

In (f) that shows comparaison for dimension 40, the CPDE algorithm clearly outperforms both BasicDE1 and BasicDE2. The ECDF curve for CPDE rises significantly faster. The primary reason for CPDE's superior performance is its adaptive mechanisms, including the COBI parameter control method and diverse mutation strategies, which enhance its ability to explore and exploit the search space efficiently. In contrast, BasicDE1 and BasicDE2 suffer from less effective parameter control and mutation strategies, resulting in reduced efficiency and slower progress.

We also noticed that BasicDE1, with its fixed population size, outperforms BasicDE2 despite having a smaller population size. This observation confirms that increasing the population size does not necessarily lead to better performance and can, in fact, diminish efficiency.

It was also observed that, even as the dimension increases, our variant maintains its supe-

riority over COBIDE. Where CPDE solves 30% of the problems, compared to only 20% by COBIDE. The split strategy appears to assist CPDE stay with better diversity and exploring the search space more effectively specially in higher dimensions.

This part compares CPDE and CPDEE. Both algorithms exhibit similar performance across dimensions 2 to 10. However, a significant distinction emerges in dimensions 20 and 40, where CPDEE demonstrates superior performance over CPDE. The remarkable performance boost of CPDEE in higher dimensions can be attributed to the fundamental difference between binomial and exponential crossover. While binomial crossover randomly selects components from the mutant vector, exponential crossover arranges these components into one or two contiguous subsequences. This unique characteristic of exponential crossover seems to play a crucial role in enhancing the effectiveness of differential evolution, especially in larger dimensions.

In the context of function subgroups, we observed that our proposed variants, CPDE and CPDEE, excel across most functions. They demonstrate notable performance in moderate and ill-conditioned functions in dimension 20, as well as in separable functions in dimension 40, Specially for CPDEE.(see figures 3.2,3.3)

Conclusion

In this chapter, we presented a comparison of the BasicDE algorithms, COBIDE, and our proposed variants (CPDE and CPDEE) on a set of test BBOB functions. We began by introducing the motivation behind our variants, followed by a detailed description, including the pseudo-code. We then defined the COCO platform used for benchmarking and analyzed the results. These results demonstrate the superior performance of our variants, especially in higher dimensions, compared to their competitors. If we had results for the large-scale BBOB, we might have drawn even better conclusions regarding higher dimensions (>40). One needs to realize that the increased complexity in higher dimensions poses significant challenges.

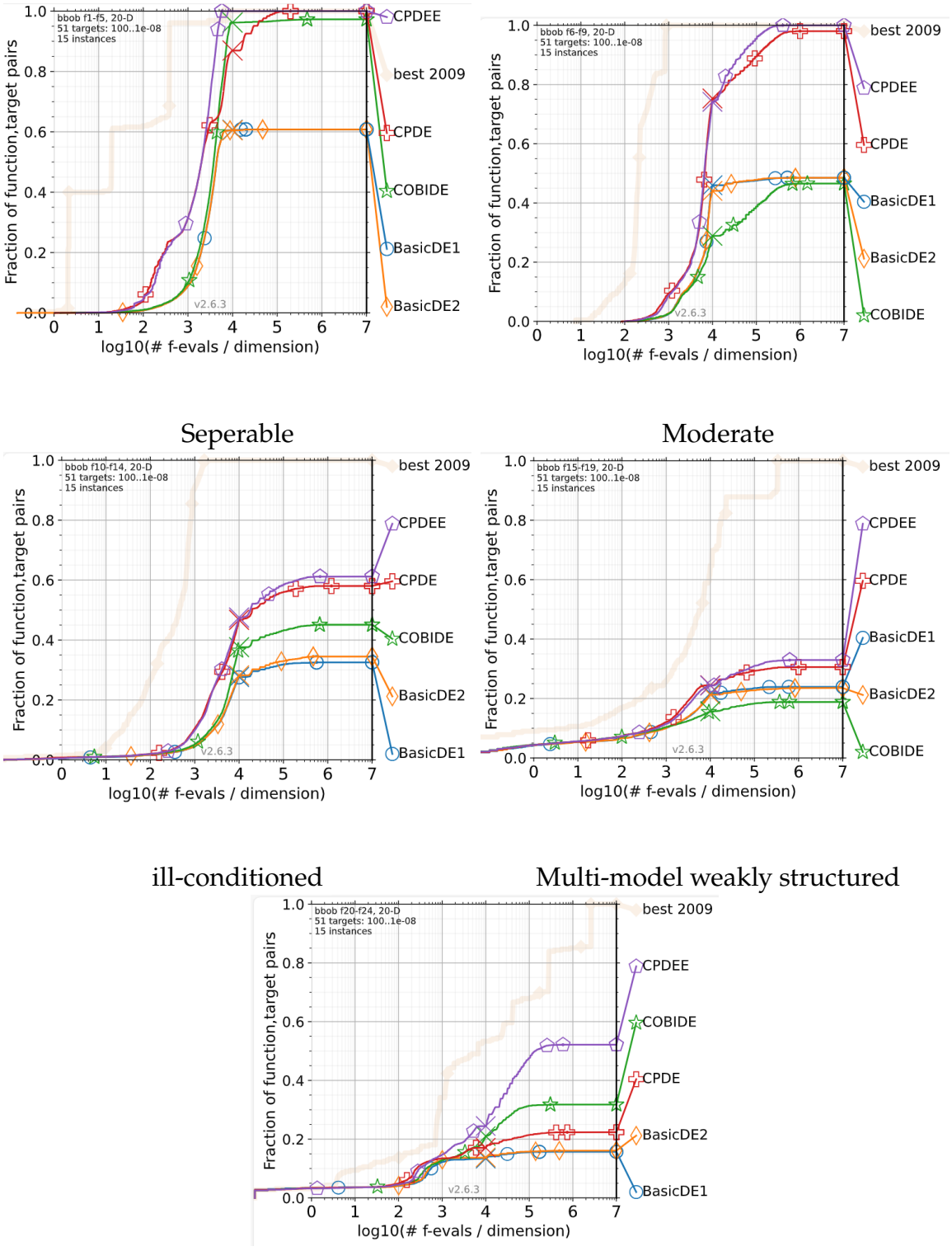


Figure 3.2: Empirical cumulative distribution functions (ECDFs) of the Expected Running Time (ERT) for the algorithms for each subgroup in dimensions 20

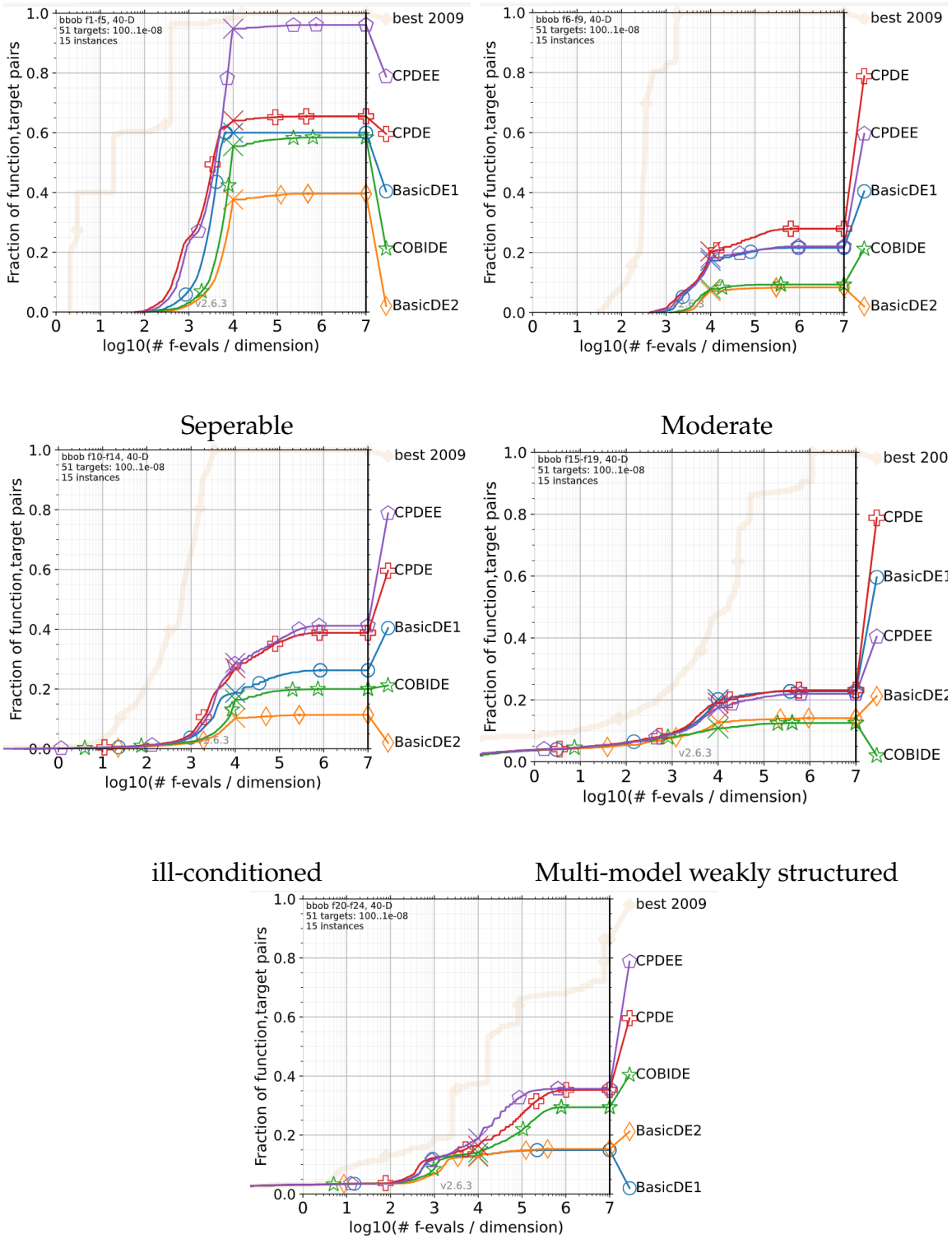


Figure 3.3: Empirical cumulative distribution functions (ECDFs) of the Expected Running Time (ERT) for the algorithms for each subgroup in dimensions 40

General Conclusion

The optimization problems are and will remain difficult regardless of the algorithmic methods developed to solve them. It is clear that it is unrealistic to imagine that the DE algorithm, or any other single algorithm, could be universally effective for all types of problems. Our work focuses on specifically improving the Differential Evolution (DE) algorithm.

We introduced a new variant called CPDE, which classifies individuals into three distinct sub-populations, each utilizing unique DE mutation strategies tailored to their capabilities. We chose to integrate simple, well-known mutation strategies (DE/rand/1, DE/best/1, and DE/rand-to-best/1) and dynamically adjusted parameters using the PCM-COBI parameter setting used in COBIDE.

Conducting research inevitably presents various challenges, and our project was no exception. We encountered several significant obstacles throughout our work:

- One of the main difficulties was the limited access to articles and resources, especially the most recent publications. This limitation constrained our ability to integrate the latest findings and advancements into our research. Plus, it's worth noting that not all metaheuristics codes are readily accessible online.
- Another challenging aspect is writing the master's thesis in English, given that the majority of resources and tutorials are also in English. This is particularly challenging due to our educational background, which is primarily in French.
- We encountered difficulties in installing the cocoex package, a well-known issue among users of the COCO platform in Python. This problem has been commonly discussed in the issues section on GitHub.
- The lack of powerful equipment capable of providing a suitable environment for running our codes posed another challenge. Unfortunately, we couldn't use Google Colab to mitigate this issue due the necessity of Colab Pro for accessing its shell. This limitation resulted in significant delays as we had to wait longer for the results.

The results we have achieved show promise; however, it is important to acknowledge that they are based on specific assumptions that may limit their applicability. Exploring alternative research avenues could lead to further improvements. One such avenue involves investigating alternative selection operators instead of relying solely on the greedy approach. Integrating deep learning techniques, such as reinforcement learning, to dynamically switch between different operators—by designing a reward function that evaluates the DE algorithm's performance and favors strategies yielding better outcomes—holds significant potential. Additionally, evaluating our algorithm on diverse benchmarks like the CEC testbed would provide valuable insights. Another intriguing direction is leveraging large language models (LLMs) to adaptively adjust parameters like F, CR, and population size based on real-time optimization dynamics. This innovative approach has the potential to advance optimization algorithms, yielding more efficient and effective results.

This project has been highly beneficial for us on several fronts. It has allowed us to acquire new knowledge and enhance existing skills, particularly in the field of metaheuristics, with a specific emphasis on the Differential Evolution algorithm and some of its variants. Examining existing literature was crucial, given that these topics were novel and outside the scope of our Master's program. This deep dive into metaheuristics not only expanded our theoretical understanding but also sharpened our practical problem-solving skills. We achieved this through hands-on experimentation and study of various algorithms to enhance performance and solution quality. Additionally, the project provided an opportunity to learn a new programming language, Python, and familiarize ourselves with the COCO framework. This experience has been instrumental in developing our skills and knowledge in the field of optimization research. Overall, this project has been a significant stepping stone in our academic and professional development, providing us with a solid foundation in metaheuristics and optimization algorithms.

Annex

Based on the analysis of the Tables: 3.1, 3.2, 3.3, 3.4, 3.5 and 3.6, we observe that in dimensions 2 and 3, the performances of the five algorithms are relatively close, with varying rates of solving instances. However, starting from dimension 5, the performance of COBIDE declines compared to the other algorithms; this decline is notable in all instances. Notably, CPDE, demonstrates superior performance by solving more instances of problems than the other algorithms, especially as the dimension increases. And what sets CPDEE apart is its ability to solve some instances of functions in dimensions 20 and 40, whereas the other algorithms fail to do so. This highlights the effectiveness of CPDE, whether it uses binomial or exponential crossover, particularly when the dimensions increase.

Functions									
Algorithm	**Separable**					**Moderate**			
	$f1$	$f2$	$f3$	$f4$	$f5$	$f6$	$f7$	$f8$	$f9$
BasicDE1	15/15	15/15	15/15	15/15	15/15	15/15	15/15	15/15	15/15
BasicDE2	15/15	15/15	14/15	11/15	15/15	12/15	15/15	14/15	12/15
COBIDE	6/15	6/15	7/15	7/15	3/15	3/15	5/15	6/15	5/15
CPDE	15/15	15/15	15/15	13/15	15/15	15/15	15/15	15/15	15/15
CPDEE	15/15	15/15	15/15	14/15	15/15	15/15	15/15	15/15	15/15

Algorithm	**Ill-conditioned**				**Multi-modal weakly structured**				
	$f10$	$f11$	$f12$	$f13$	$f14$	$f15$	$f16$	$f17$	$f18$
BasicDE1	15/15	15/15	15/15	15/15	15/15	15/15	6/15	11/15	1/15
BasicDE2	15/15	15/15	14/15	15/15	15/15	13/15	12/15	15/15	14/15
COBIDE	8/15	6/15	8/15	6/15	3/15	10/15	10/15	6/15	6/15
CPDE	15/15	14/15	15/15	15/15	15/15	12/15	12/15	15/15	15/15
CPDEE	14/15	15/15	15/15	15/15	15/15	13/15	9/15	15/15	15/15

Algorithm	**Multi-modal with global structure**					
	$f19$	$f20$	$f21$	$f22$	$f23$	$f24$
BasicDE1	4/15	15/15	15/15	15/15	0/15	0/15
BasicDE2	13/15	13/15	14/15	13/15	13/15	3/15
COBIDE	6/15	10/15	11/15	13/15	1/15	5/15
CPDE	9/15	14/15	14/15	13/15	2/15	2/15
CPDEE	14/15	13/15	15/15	13/15	0/15	3/15

Table 3.1: Success rates for the algorithms BasicDE1, BasicDE2, COBIDE, CPDE, and CPDEE for each function group in dimension 2.

Functions									
Algorithm	**Separable**					**Moderate**			
	$f1$	$f2$	$f3$	$f4$	$f5$	$f6$	$f7$	$f8$	$f9$
BasicDE1	15/15	15/15	15/15	7/15	15/15	15/15	15/15	15/15	15/15
BasicDE2	15/15	15/15	14/15	6/15	15/15	8/15	15/15	7/15	5/15
COBIDE	4/15	8/15	6/15	6/15	2/15	2/15	5/15	7/15	6/15
CPDE	15/15	15/15	15/15	12/15	15/15	15/15	15/15	15/15	15/15
CPDEE	15/15	15/15	14/15	15/15	15/15	15/15	15/15	15/15	15/15

Algorithm	**Ill-conditioned**				**Multi-modal weakly structured**				
	$f10$	$f11$	$f12$	$f13$	$f14$	$f15$	$f16$	$f17$	$f18$
BasicDE1	15/15	15/15	7/15	15/15	15/15	1/15	1/15	6/15	0/15
BasicDE2	14/15	14/15	3/15	14/15	15/15	10/15	6/15	15/15	13/15
COBIDE	6/15	1/15	7/15	3/15	4/15	5/15	6/15	2/15	4/15
CPDE	15/15	15/15	13/15	15/15	15/15	6/15	4/15	15/15	15/15
CPDEE	15/15	15/15	14/15	15/15	15/15	6/15	6/15	15/15	14/15

Algorithm	**Multi-modal with global structure**					
	$f19$	$f20$	$f21$	$f22$	$f23$	$f24$
BasicDE1	0/15	15/15	15/15	15/15	0/15	0/15
BasicDE2	2/15	11/15	11/15	8/15	10/15	0/15
COBIDE	0/15	3/15	11/15	9/15	0/15	0/15
CPDE	1/15	12/15	13/15	10/15	2/15	0/15
CPDEE	2/15	13/15	11/15	9/15	1/15	0/15

Table 3.2: Success rates for the algorithms BasicDE1, BasicDE2, COBIDE, CPDE, and CPDEE for each function group in dimension 3.

Functions									
Algorithm	**Separable**					**Moderate**			
	$f1$	$f2$	$f3$	$f4$	$f5$	$f6$	$f7$	$f8$	$f9$
BasicDE1	15/15	15/15	0/15	0/15	15/15	15/15	15/15	15/15	14/15
BasicDE2	15/15	15/15	11/15	2/15	15/15	8/15	15/15	0/15	0/15
COBIDE	3/15	2/15	2/15	2/15	0/15	0/15	6/15	2/15	3/15
CPDE	15/15	15/15	13/15	9/15	15/15	15/15	15/15	12/15	15/15
CPDEE	15/15	15/15	15/15	14/15	15/15	15/15	15/15	14/15	15/15

Algorithm	**Ill-conditioned**				**Multi-modal weakly structured**				
	$f10$	$f11$	$f12$	$f13$	$f14$	$f15$	$f16$	$f17$	$f18$
BasicDE1	15/15	15/15	3/15	0/15	15/15	0/15	0/15	0/15	0/15
BasicDE2	3/15	3/15	0/15	0/15	6/15	1/15	2/15	13/15	11/15
COBIDE	0/15	7/15	1/15	3/15	1/15	0/15	1/15	2/15	0/15
CPDE	15/15	15/15	14/15	15/15	15/15	1/15	3/15	11/15	9/15
CPDEE	15/15	15/15	15/15	15/15	15/15	1/15	0/15	10/15	10/15

Algorithm	**Multi-modal with global structure**					
	$f19$	$f20$	$f21$	$f22$	$f23$	$f24$
BasicDE1	0/15	0/15	13/15	9/15	0/15	0/15
BasicDE2	0/15	8/15	4/15	3/15	0/15	0/15
COBIDE	0/15	3/15	3/15	6/15	0/15	0/15
CPDE	0/15	4/15	4/15	6/15	0/15	0/15
CPDEE	0/15	10/15	8/15	6/15	0/15	0/15

Table 3.3: Success rates for the algorithms BasicDE1, BasicDE2, COBIDE, CPDE, and CPDEE for each function group in dimension 5.

Functions									
Algorithm	**Separable**					**Moderate**			
	$f1$	$f2$	$f3$	$f4$	$f5$	$f6$	$f7$	$f8$	$f9$
BasicDE1	15/15	15/15	0/15	0/15	15/15	0/15	15/15	0/15	0/15
BasicDE2	15/15	15/15	1/15	0/15	15/15	15/15	14/15	1/15	0/15
COBIDE	1/15	1/15	0/15	0/15	0/15	0/15	7/15	0/15	1/15
CPDE	15/15	15/15	12/15	9/15	15/15	15/15	14/15	15/15	14/15
CPDEE	15/15	15/15	15/15	15/15	15/15	15/15	12/15	15/15	13/15

Algorithm	**Ill-conditioned**				**Multi-modal weakly structured**				
	$f10$	$f11$	$f12$	$f13$	$f14$	$f15$	$f16$	$f17$	$f18$
BasicDE1	0/15	0/15	0/15	0/15	5/15	0/15	0/15	0/15	0/15
BasicDE2	1/15	0/15	0/15	0/15	1/15	0/15	0/15	15/15	0/15
COBIDE	1/15	0/15	0/15	0/15	0/15	0/15	0/15	0/15	0/15
CPDE	15/15	15/15	9/15	10/15	15/15	0/15	0/15	1/15	0/15
CPDEE	15/15	15/15	9/15	7/15	14/15	0/15	0/15	3/15	0/15

Algorithm	**Multi-modal with global structure**					
	$f19$	$f20$	$f21$	$f22$	$f23$	$f24$
BasicDE1	0/15	0/15	5/15	1/15	0/15	0/15
BasicDE2	0/15	2/15	3/15	0/15	0/15	0/15
COBIDE	0/15	0/15	2/15	1/15	0/15	0/15
CPDE	0/15	1/15	2/15	1/15	0/15	0/15
CPDEE	0/15	2/15	4/15	1/15	0/15	0/15

Table 3.4: Success rates for the algorithms BasicDE1, BasicDE2, COBIDE, CPDE, and CPDEE for each function group in dimension 10.

Functions									
Algorithm	**Separable**					**Moderate**			
	$f1$	$f2$	$f3$	$f4$	$f5$	$f6$	$f7$	$f8$	$f9$
BasicDE1	15/15	15/15	0/15	0/15	15/15	0/15	15/15	0/15	0/15
BasicDE2	15/15	15/15	0/15	0/15	15/15	0/15	13/15	0/15	0/15
COBIDE	0/15	0/15	0/15	0/15	0/15	0/15	0/15	0/15	0/15
CPDE	15/15	15/15	13/15	4/15	15/15	15/15	1/15	15/15	0/15
CPDEE	15/15	15/15	15/15	15/15	15/15	4/15	2/15	14/15	3/15

Algorithm	**Ill-conditioned**				**Multi-modal weakly structured**				
	$f10$	$f11$	$f12$	$f13$	$f14$	$f15$	$f16$	$f17$	$f18$
BasicDE1	0/15	0/15	0/15	0/15	0/15	0/15	0/15	0/15	0/15
BasicDE2	0/15	0/15	0/15	0/15	0/15	0/15	0/15	0/15	0/15
COBIDE	0/15	0/15	0/15	0/15	0/15	0/15	0/15	0/15	0/15
CPDE	0/15	0/15	0/15	0/15	0/15	0/15	0/15	0/15	0/15
CPDEE	0/15	0/15	0/15	0/15	0/15	0/15	0/15	0/15	0/15

Algorithm	**Multi-modal with global structure**					
	$f19$	$f20$	$f21$	$f22$	$f23$	$f24$
BasicDE1	0/15	0/15	0/15	0/15	0/15	0/15
BasicDE2	0/15	0/15	0/15	0/15	0/15	0/15
COBIDE	0/15	0/15	0/15	0/15	0/15	0/15
CPDE	0/15	0/15	0/15	0/15	0/15	0/15
CPDEE	0/15	2/15	4/15	0/15	0/15	0/15

Table 3.5: Success rates for the algorithms BasicDE1, BasicDE2, COBIDE, CPDE, and CPDEE for each function group in dimension 20.

Functions									
Algorithm	**Separable**					**Moderate**			
	$f1$	$f2$	$f3$	$f4$	$f5$	$f6$	$f7$	$f8$	$f9$
BasicDE1	15/15	15/15	0/15	0/15	15/15	0/15	0/15	0/15	0/15
BasicDE2	0/15	0/15	0/15	0/15	0/15	0/15	0/15	0/15	0/15
COBIDE	0/15	0/15	0/15	0/15	0/15	0/15	0/15	0/15	0/15
CPDE	15/15	15/15	0/15	0/15	15/15	0/15	0/15	0/15	0/15
CPDEE	15/15	15/15	15/15	0/15	15/15	0/15	0/15	0/15	0/15

Algorithm	**Ill-conditioned**				**Multi-modal weakly structured**				
	$f10$	$f11$	$f12$	$f13$	$f14$	$f15$	$f16$	$f17$	$f18$
BasicDE1	0/15	0/15	0/15	0/15	0/15	0/15	0/15	0/15	0/15
BasicDE2	0/15	0/15	0/15	0/15	0/15	0/15	0/15	0/15	0/15
COBIDE	0/15	0/15	0/15	0/15	0/15	0/15	0/15	0/15	0/15
CPDE	0/15	0/15	0/15	0/15	0/15	0/15	0/15	0/15	0/15
CPDEE	0/15	0/15	0/15	0/15	0/15	0/15	0/15	0/15	0/15

Algorithm	**Multi-modal with global structure**					
	$f19$	$f20$	$f21$	$f22$	$f23$	$f24$
BasicDE1	0/15	0/15	0/15	0/15	0/15	0/15
BasicDE2	0/15	0/15	0/15	0/15	0/15	0/15
COBIDE	0/15	0/15	0/15	0/15	0/15	0/15
CPDE	0/15	0/15	1/15	0/15	0/15	0/15
CPDEE	0/15	0/15	3/15	0/15	0/15	0/15

Table 3.6: Success rates for the algorithms BasicDE1, BasicDE2, COBIDE, CPDE, and CPDEE for each function group in dimension 40.

Bibliography

[1] M. Alswaitti, M. Albughdadi, and N.A.M. Isa. Variance-based differential evolution algorithm with an optional crossover for data clustering. *Applied Soft Computing*, 80: 1–17, 2019.

[2] C. Aranha, C.L. Camacho Villalón, F. Campelo, et al. Metaphor-based metaheuristics, a call for action: the elephant in the room. *Swarm Intelligence*, 16:1–6, 2022.

[3] V. Aslantas. An optimal robust digital image watermarking based on svd using differential evolution algorithm. *Optics Communications*, 282(5):769–777, Mar. 2009.

[4] Sunith Bandaru and Kalyanmoy Deb. Metaheuristic techniques. *Decision Sciences*, 2016.

[5] Nazia Bibi, Ali Ahsan, and Zeeshan Anwar. Project resource allocation optimization using search based software engineering—a framework. In *Ninth International Conference on Digital Information Management (ICDIM 2014)*, pages 226–229. IEEE, 2014.

[6] A. Biswas, S. Das, A. Abraham, and S. Dasgupta. Design of fractional-order piλdμ controllers with an improved differential evolution. *Engineering Applications of Artificial Intelligence*, 22(2):343–350, March 2009.

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

[8] J. Brest, S. Greiner, B. Bošković, M. Mernik, and V. Žumer. Self-adapting control parameters in differential evolution: a comparative study on numerical benchmark problems. *IEEE Transactions on Evolutionary Computation*, 10(6):646–657, 2006.

[9] Janez Brest and Matjaž Maučec. Self-adaptive differential evolution algorithm using population size reduction and three strategies. *Soft Computing - A Fusion of Foundations, Methodologies and Applications*, 15:2157–2174, 2011.

[10] H. R. Cai, C. Y. Chung, and K. P. Wong. Application of differential evolution algorithm for transient stability constrained optimal power flow. *IEEE Transactions on Power Systems*, 23(2), May 2008.

[11] W-D. Chang. Parameter identification of chen and lü systems: A differential evolution approach. *Chaos, Solitons and Fractals*, 32(4):1469–1476, May 2007.

[12] Rachid Chelouah and Patrick Siarry. A continuous genetic algorithm designed for the global optimization of multimodal functions. *Journal of Heuristics*, 6(2):191–213, 2000.

[13] Rachid Chelouah and Patrick Siarry. A hybrid method combining continuous tabu search and nelder–mead simplex algorithms for the global optimization of multiminima functions. *European Journal of Operational Research*, 161(3):636–654, 2005.

[14] S-L. Cheng and C. Hwang. Optimal approximation of linear systems by a differential evolution algorithm. *IEEE Transactions on Systems, Man, and Cybernetics—Part A: Systems and Humans*, 31(6):698–707, November 2001.

[15] J.-P. Chiou. A variable scaling hybrid differential evolution for solving large-scale power dispatch problems. *IET Generation, Transmission & Distribution*, 3(2):154–163, February 2009.

[16] J.-P. Chiou, C.-F. Chang, and C.-T. Su. Ant direction hybrid differential evolution for solving large capacitor placement problems. *IEEE Transactions on Power Systems*, 19: 1794–1800, November 2004.

[17] L. S. Coelho, J. G. Sauer, and M. Rudek. Differential evolution optimization combined with chaotic sequences for image contrast enhancement. *Chaos, Solitons and Fractals*, 42(1):522–529, Oct. 2009.

[18] I. L. Lopez Cruz, L. G. Van Willigenburg, and G. Van Straten. Efficient differential evolution algorithms for multimodal optimal control problems. *Applied Soft Computing*, 3(2):97–122, Sept. 2003.

[19] I. L. López Cruz, L. G. van Willigenburg, and G. van Straten. Optimal control of nitrate in lettuce by a hybrid approach: differential evolution and adjustable control weight gradient algorithms.

[20] Laizhong Cui and et al. Adaptive differential evolution algorithm with novel mutation strategies in multiple sub-populations. *Computers & Operations Research*, 67: 155–173, 2016.

[21] Amit Kumar Das and Dilip Kumar Pratihar. A direction-based exponential mutation operator for real-coded genetic algorithm. In *2018 Fifth International Conference on Emerging Applications of Information Technology (EAIT)*, pages 1–4. IEEE, 2018.

[22] S. Das and P. N. Suganthan. Differential evolution: A survey of the state-of-the-art. *IEEE Transactions on Evolutionary Computation*, 15:4–31, 2011.

[23] S. Das, A. Abraham, and A. Konar. Automatic clustering using an improved differential evolution algorithm. *IEEE Trans. Syst. Man Cybern. Part A*, 38(1):218–236, Jan. 2008.

[24] S. Das, A. Abraham, and A. Konar. Metaheuristic clustering. In *SCI 178*, pages 175–211. Springer-Verlag, 2009.

[25] S. Das, A. Ghosh, and S. S. Mullick. A switched parameter differential evolution for large scale global optimization – simpler may be better. In *MENDEL 2015: 21st International Conference on Soft Computing, Advances in Intelligent Systems and Computing*, volume 378, pages 103–125, Brno, Czech Republic, June 23–25 2015.

[26] Swagatam Das and Ponnuthurai Nagaratnam Suganthan. Differential evolution: A survey of the state-of-the-art. *IEEE transactions on evolutionary computation*, 15(1):4–31, 2010.

[27] Kalyanmoy Deb and Samir Agrawal. A niched-penalty approach for constraint handling in genetic algorithms. In *Proceedings of the International Conference on Artificial Neural Networks and Genetic Algorithms (ICANNGA-99)*, pages 235–243. Springer-Verlag, 1999.

[28] Thakur M Deep K. A new crossover operator for real coded genetic algorithms. *Appl Math Comput*, 188:895–911, 2007.

[29] Anton Dekkers and Emile Aarts. Global optimization and simulated annealing. *Mathematical programming*, 50:367–393, 1991.

[30] S. M. Elsayed, R. A. Sarker, and T. Ray. Differential evolution with automatic parameter configuration for solving the cec2013 competition on real-parameter optimization. In *Proceedings of IEEE Congress on Evolutionary Computation, 2013*, pages 1932–1937, Cancún, México, June 20–23 2013.

[31] I. De Falco, A. D. Cioppa, D. Maisto, and F. Tarantino. Differential evolution as a viable tool for satellite image registration. *Applied Soft Computing Journal*, 8(4):1453–1462, Sept. 2008.

[32] I. Fister, A. Tepeh, and I. Fister Jr. Epistatic arithmetic crossover based on cartesian graph product in ensemble differential evolution. *Applied Mathematics and Computation*, 283:181–194, 2016.

[33] Gerald J.Lieberman Frederick S.Hillier. *Introduction to Operations Research*. McGraw-Hill Education, 2014.

[34] Fred Glover. Tabu search: Part i. *ORSA Journal on Computing*, 1(3):190–206, 1989.

[35] W. Gong and Z. Cai. Differential evolution with ranking-based mutation operators. *IEEE Transactions on Cybernetics*, 43(6):2066–2081, 2013.

[36] Thomas Guilmeau, Emilie Chouzenoux, and Víctor Elvira. Simulated annealing: A review and a new scheme. In *2021 IEEE Statistical Signal Processing Workshop (SSP)*, pages 101–105. IEEE, 2021.

[37] N.M. Hamza, D.L. Essam, and R.A. Sarker. Constraint consensus mutation-based differential evolution for constrained optimization. *IEEE Transactions on Evolutionary Computation*, 20(3):447–459, 2015.

[38] Nikolaus Hansen et al. Coco: A platform for comparing continuous optimizers in a black-box setting. *Optimization Methods and Software*, 36(1):114–144, 2021.

[39] Lester Ingber. Simulated annealing: Practice versus theory. *Journal of Mathematical and Computer Modelling*, 18(11):29–57, 1993.

[40] M. W. Iruthayarajan and S. Baskar. Evolutionary algorithms based design of multivariable pid controller. *Expert Systems with Applications*, 36(5):9159–9167, July 2009.

[41] Daniel M Jaeggi, Geoffrey T Parks, Timoleon Kipouros, and P John Clarkson. The development of a multi-objective tabu search algorithm for continuous optimisation problems. *European Journal of Operational Research*, 185(3):1192–1212, 2008.

[42] Holland John. Holland. adaptation in natural and artificial systems, 1975.

[43] S. Katoch, S. S. Chauhan, and V. Kumar. A review on genetic algorithm: past, present, and future. *Multimedia Tools and Applications*, 80:8091–8126, 2021. doi: 10.1007/s11042-020-10139-6. URL `https://doi.org/10.1007/s11042-020-10139-6`.

[44] James Kennedy and Russell Eberhart. Particle swarm optimization. In *Proceedings of ICNN'95-international conference on neural networks*, volume 4, pages 1942–1948. ieee, 1995.

[45] S. Kirkpatrick, C.D. Gelatt, and M.P. Vecchi. Optimization by simulated annealing. *Science*, 220(4598):671–680, 1983.

[46] Y. Li, S. Wang, X. Hong, and Y. Li. Multi-objective task scheduling optimization in cloud computing based on genetic algorithm and differential evolution algorithm. In *2018 37th Chinese Control Conference (CCC)*, pages 4489–4494. IEEE, 2018.

[47] K. Mason, J. Duggan, and E. Howley. A multi-objective neural network trained with differential evolution for dynamic economic emission dispatch. *International Journal of Electrical Power & Energy Systems*, 100:201–221, 2018.

[48] Z. Michalewicz. Genetic algorithms, numerical optimization and constraints. In L. J. Eshelmen, editor, *Proceedings of the 6th International Conference on Genetic Algorithms*, pages 151–158. Morgan Kaufmann, San Mateo, CA, 1995.

[49] Zbigniew Michalewicz, Cezary Z Janikow, and Jacek B Krawczyk. A modified genetic algorithm for optimal control problems. *Computers & Mathematics with Applications*, 23(12):83–94, 1992.

[50] Jean-Yves Potvin Michel Gendreau. *Handbook of metaheuristics*. Springer Science & Business Media, 2010.

[51] Mitsunori Miki, Tomoyuki Hiroyasu, and Masahiko Ogura. Temperature parallel simulated annealing with adaptive neighborhood for continuous optimization problem. In *Second International Workshop on Intelligent Systems Design and Application*. IEEE, 2002.

[52] A.W. Mohamed and A.S. Almazyad. Differential evolution with novel mutation and adaptive crossover strategies for solving large scale global optimization problems. *Applied Computational Intelligence and Soft Computing*, 2017, 2017.

[53] A. Moharam, M.A. El-Hosseini, and H.A. Ali. Design of optimal pid controller using hybrid differential evolution and particle swarm optimization with an aging leader and challengers. *Applied Soft Computing*, 38:727–737, 2016.

[54] S. Moonchai, W. Madlhoo, K. Jariyachavalit, H. Shimizu, S. Shioya, and S. Chauvatcharin. Application of a mathematical model and differential evolution algorithm approach to optimization of bacteriocin production by lactococcus lactisc7. *Bioprocess and Biosystems Engineering*, 28:15–26, July 2005.

[55] A. Nobakhti and H. Wang. A simple self-adaptive differential evolution algorithm with application on the alstom gasifier. *Applied Soft Computing*, 8(1), Jan. 2008.

[56] N. Noman and H. Iba. Inferring gene regulatory networks using differential evolution with local search heuristics. *IEEE/ACM Trans. Comput. Biol. Bioinform.*, 4(4):634–647, Oct. 2007.

[57] N. Noman and H. Iba. Differential evolution for economic load dispatch problems. *Electric Power Systems Research*, 78(8):1322–1331, August 2008.

[58] S. Paterlini and T. Krink. Differential evolution and particle swarm optimization in partitional clustering. *Computational Statistics and Data Analysis*, 50(5):1220–1247, Mar. 2006.

[59] J. K. Pattanaik, M. Basu, and D. P. Dash. Opposition-based differential evolution for hydrothermal power system. *Protection and Control of Modern Power Systems*, 2017.

[60] Philipp Probst, Anne-Laure Boulesteix, and Bernd Bischl. Tunability: Importance of hyperparameters of machine learning algorithms. *Journal of Machine Learning Research*, 20(53):1–32, 2019.

[61] A. Kai Qin, Vicky Ling Huang, and Ponnuthurai N. Suganthan. Differential evolution algorithm with strategy adaptation for global numerical optimization. *IEEE transactions on Evolutionary Computation*, 13(2):398–417, 2008.

[62] C.M. Saporetti, L. Goliatt, and E. Pereira. Neural network boosted with differential evolution for lithology identification based on well logs information. *Earth Science Informatics*, 14(1):133–140, 2021.

[63] R. A. Sarker, S. M. Elsayed, and T. Ray. Differential evolution with dynamic parameters selection for optimization problems. *IEEE Transactions on Evolutionary Computation*, 18(5):689–707, 2014.

[64] S. Sayah and K. Zehar. Modified differential evolution algorithm for optimal power flow with non-smooth cost functions. *Energy Conversion and Management*, 49(11): 3036–3042, November 2008.

[65] Patrick Siarry and Gérard Berthiau. Fitting of tabu search to optimize functions of continuous variables. *International Journal for Numerical Methods in Engineering*, 40: 2449–2457, 1997.

[66] H. Silverio and L. R. Bitello. A differential evolution approach for protein folding using a lattice model. *Journal of Computer Science and Technology*, 22(6), 2007.

[67] M. Song, K. Chen, and J. Wang. Three-dimensional wind turbine positioning using gaussian particle swarm optimization with differential evolution. *Journal of Wind Engineering and Industrial Aerodynamics*, 172:317–324, 2018.

[68] R. Storn and K. V. Price. Differential evolution: A simple and efficient heuristic for global optimization over continuous spaces. *Journal of Global Optimization*, 1997.

[69] G. Sun, Y. Lan, and R. Zhao. Differential evolution with gaussian mutation and dynamic parameter adjustment. *Soft Computing*, 23(5):1615–1642, 2019.

[70] E Talbi. *Metaheuristics: from design to implementation*. John Wiley & Sons, 2009.

[71] El-Ghazali Talbi. *Metaheuristics: From Design to Implementation*. John Wiley & Sons, 2009.

[72] Ryoji Tanabe and Alex Fukunaga. Reviewing and benchmarking parameter control methods in differential evolution. *IEEE transactions on cybernetics*, 50(3):1170–1184, 2019.

[73] A. Trivedi, D. Srinivasan, S. Biswas, and T. Reindl. A genetic algorithm–differential evolution based hybrid framework: case study on unit commitment scheduling problem. *Information Sciences*, 354:275–300, 2016.

[74] Yong Wang, Jianqiang Li, Xiang Gao, and Hui Yu. Differential evolution based on covariance matrix learning and bimodal distribution parameter setting. *Applied Soft Computing*, 18:232–247, 2014.

[75] A. H. Wright. Genetic algorithms for real parameter optimization. In G. J. E. Rawlins, editor, *Foundations of Genetic Algorithms I*, pages 205–218. Morgan Kaufmann, San Mateo, 1991.

[76] R. Xu, G. K. Venayagamoorthy, and D. C. Wunsch. Modeling of gene regulatory networks with hybrid differential evolution and particle swarm optimization. *Neural Networks*, 20(8):917–927, October 2007.

[77] M. Yang, Z. Cai, C. Li, and J. Guan. An improved adaptive differential evolution algorithm with population adaptation. In Christian Blum, editor, *Proceedings of the 15th annual conference on Genetic and Evolutionary Computation (GECCO '13)*, pages 145–152, New York, NY, USA, 2013. ACM.

[78] Daniela Zaharie. Influence of crossover on the behavior of differential evolution algorithms. *Applied soft computing*, 9(3):1126–1138, 2009.

Acronyms

EA Evolutionary Algorithm

SA Simulated Annealing

FSA Fast Simulated Annealing

SMC-SA Sequential Monte Carlo simulated annealing

CSA Curious Simulated Annealing

TS Tabu Search

CTS Continuous Tabu Search

MOTS Multi-Objective Tabu Search

PRMOTS Pareto Ranked Multi-Objective Tabu Search

GA Genetic Algorithm

HX Heuristics Crossover

NUM Non-Uniform Mutation

LX Laplace Crossover

RGA Real coded Genetic Algorithm

DEM Direction-based Exponential Mutation

PM Polynomial Mutation

DE Differential Evolution

CR Crossover Rate

F Mutation Factor

P Population

NP, N Population size

PSO Particle Swarm Optimization

BBOB Black Box Optimization Benchmarking

COCO COmparing Continuous Optimizers platform

JADE Adaptive Differential Evolution with Optional External Archive

CEC Congress on Evolutionary Computation

JDE self-adaptive DE

JDEbin self-adaptive DE/rand/1/bin

JDEexp self-adaptive DE/rand/1/exp

JDEbest self-adaptive DE/best/1/bin

CC Constraint Consensus

GPDE Gaussian Mutation and Dynamic Parameter Adjustment Differential Evolution

VDEO Variance-based Differential Evolution Algorithm with Optional Crossover

eXEDE Epistatic Arithmetic Crossover Operator based on Cartesian Graph Product in Ensemble Differential Evolution

ANDE Adaptive Novel Differential Evolution

ANN Artificial Neural Network

DEED Dynamic Economic Emission Dispatch

PID Proportional-Integral-Derivative

ALC-PSODE Aging Leader and Challengers Particle Swarm Optimization with Differential Evolution

D Dimensional size of a given problem

HGADE Hybridizing Genetic Algorithm with Differential Evolution

PCM Parametre Control Methode

COBIDE Covariance Matrix Learning and Bimodal Distribution Parameter Setting Differential Evolution

CPDE Cauchy Partitioned Differential Evolution

CPDEE Cauchy Partitioned Exponential Differential Evolution

HTML HyperText Markup Language

ECDF Empirical Cumulative Distribution Function

COCOPP COCO Post Processing

ERT Expected Running Time

LLM Large Language Model