

THESE

En vue de l'obtention du : **DOCTORAT**

Structure de Recherche : Laboratoire Conception et Systèmes
(Signaux, Electronique et Informatique)

Discipline : Informatique

Spécialité : Intelligence Artificielle

Présentée et soutenue le : 20 Septembre 2021 par :

Mohamed Amine EL MAJDOULI

**Using Metaheuristics for NP-Hard and Big Data Optimization Problems:
Conception, Adaptation and a Proposal for a Smarter application**

JURY

Mohamed JEDRA	PES, Faculté des Sciences, Université Mohammed V, Rabat	Président
Abdelaziz BOUROUMI	PES, Faculté des Sciences Ben M'sik, Université Hassan II, Casablanca	Rapporteur / Examineur
Khalid ZINE-DINE	PES, Faculté des Sciences, Université Mohammed V, Rabat	Rapporteur / Examineur
Soumia ZITI	PH, Faculté des Sciences, Université Mohammed V, Rabat	Rapporteur / Examineur
Khalid CHOUGDALI	PH, Ecole Nationale des Sciences Appliquées, Université Ibn Tofail, Kenitra	Rapporteur / Examineur
Abdelhakim Ameur EL IMRANI	PES, Faculté des Sciences, Université Mohammed V, Rabat	Directeur de Thèse

Année Universitaire : 2020/2021

AVANT PROPOS

Les travaux présentés dans cette thèse ont été effectués au sein du laboratoire Conception et Systèmes (Signaux, Electronique et Informatique) de la Faculté des Sciences de Rabat sous la direction du Professeur Abdelhakim Ameer EL IMRANI.

Je tiens tout d'abord à exprimer mes sincères gratitude à Mr. Abdelhakim Ameer EL IMRANI, Professeur de l'Enseignement Supérieur à la Faculté des Sciences de Rabat, pour la confiance qu'il m'a accordée en acceptant de diriger mes travaux de recherche durant ces années de thèse et pour l'intérêt qu'il a porté à mes travaux. Le soutien permanent dont il m'a constamment fait preuve est à l'origine de ce travail.

Je souhaite exprimer ma profonde gratitude à Mr. Mohamed JEDRA, Professeur de l'Enseignement Supérieur à la Faculté des Sciences de Rabat, pour l'honneur qu'il me fait en acceptant de présider le jury de cette thèse.

Je présente à Mr. Abdelaziz BOUROUMI, Professeur de l'Enseignement Supérieur à la Faculté des Sciences Ben M'sik de l'Université Hassan II Casablanca, l'expression de ma profonde gratitude pour avoir accepté de porter un jugement sur ce travail en tant que rapporteur et examinateur.

Je remercie vivement Mr. Khalid ZINEDINE, Professeur de l'Enseignement Supérieur à la Faculté des Sciences de Rabat, pour avoir accepté d'examiner et de rapporter sur ce travail.

Mes vifs remerciements vont au Mme. Soumia ZITI, Professeur Habilité à la Faculté des Sciences de Rabat, pour avoir accepté autant que rapporteur et examinateur d'évaluer et de juger ce mémoire de thèse.

Je tiens à remercier Mr. Khalid CHOUGDALI, Professeur Habilité à l'Ecole Nationale des Sciences Appliquées de l'Université Ibn Tofail à Kénitra, d'avoir accepté d'évaluer et de juger ce travail autant que rapporteur et examinateur.

Que les collègues du laboratoire : Issam EL AFI, Saad BOUGRINE, Ismail RBOUH, trouvent ici le témoignage de ma sincère amitié.

Enfin je ne saurai oublier tous ceux qui ont contribué de près ou de loin à l'aboutissement de ce travail.

ABSTRACT

This work describes an effort towards the conception of novel metaheuristics and their application to NP-hard and big data optimization problems. Indeed, four major contributions are reported in this thesis. The first contribution introduces a new nature-inspired search model called “Lightning Inspired Search Algorithm”. The model uses a constructive mechanism, similar to the pattern of the natural lightning formation, that promotes diversity during the search to avoid premature convergence. The experiments performed over the traveling salesman problem show a very competitive behavior of the algorithm. The second contribution presents an adaptation of the Fireworks Algorithm to single machine scheduling problems. The proposed algorithm redesigns the exploration and exploitation mechanism implemented by the spark’s generation phase. The experimental results are satisfactory in comparison to state-of-the-art algorithms. The third contribution concerns the application of the Fireworks Algorithm to big data optimization problems. The original fireworks algorithm is revisited to enhance the speed of convergence while preserving its robustness. The results show that the proposed solution outperforms all proposed methods for *BigOPT*. The last contribution offers a new design pattern for using metaheuristics based on automatic algorithm configuration and per-instance algorithm selection techniques for a smarter usage of metaheuristics in future applications.

Keywords: *Metaheuristics, NP-Hard problems, Fireworks Algorithm, BigOPT, SMTWT.*

RESUMÉ

Ce mémoire décrit des travaux de recherche menés pour la conception de nouvelles métaheuristiques et leurs applications aux problèmes d'optimisation NP-Difficiles et Big Data. En effet, quatre contributions majeures sont apportées. La première contribution présente un nouveau modèle de recherche inspiré de la nature appelé « Lightning-Inspired Search Algorithm ». Le modèle utilise un mécanisme constructif, similaire au modèle de la formation naturelle de la foudre, qui favorise la diversité pendant la recherche pour éviter une convergence prématurée. Les expériences réalisées sur le problème du voyageur de commerce montrent un comportement très compétitif de l'algorithme. La deuxième contribution présente une adaptation de l'algorithme « Fireworks Algorithm (*FA*) » aux problèmes d'ordonnancement à une seule machine. L'algorithme proposé modifie le mécanisme d'exploration et d'exploitation de la phase de génération d'étincelles. Les résultats expérimentaux sont satisfaisants par rapport aux algorithmes de l'état de l'art. La troisième contribution concerne l'application de *FA* à des problèmes d'optimisation de big data. L'algorithme original *FA* est revisité pour améliorer sa vitesse de convergence tout en préservant sa robustesse. Les résultats montrent que la solution proposée surpasse toutes les méthodes proposées pour *BigOPT*. La dernière contribution introduit de nouvelles idées sur le développement des métaheuristiques, en introduisant des techniques de configuration automatique d'algorithmes et de sélection d'algorithmes par instance, pour une utilisation plus intelligente des métaheuristiques dans les applications futures.

Mots-clés : *Métaheuristiques, Problèmes NP-difficiles, Fireworks Algorithm, BigOPT, SMTWT.*

RESUMÉ DÉTAILLÉ

Ce mémoire décrit des travaux de recherche menés pour la conception de nouvelles métaheuristiques et leurs applications aux problèmes d'optimisation NP-Difficiles et Big Data. En effet, résoudre ce genre de problèmes en utilisant l'optimisation mathématique éprouve souvent de nombreuses difficultés telles que l'absence de convexité ou la grande complexité de résolution pour atteindre l'optimalité. Ainsi, des méthodes approximatives telles que les métaheuristiques sont utilisées en raison de leur compromis revendiquant un optimum de bonne qualité dans un temps de résolution raisonnable. Alors que les problèmes du monde réel évoluent continuellement vers de nouveaux niveaux de complexité, de nouvelles conceptions et adaptations des métaheuristiques s'avèrent nécessaires.

Dans ce contexte, quatre contributions majeures sont apportées. La première contribution présente un nouveau modèle de recherche inspiré de la nature appelé « Lightning-Inspired Search Algorithm ». Pour construire des solutions, le modèle utilise un mécanisme constructif, similaire au modèle de la formation naturelle de la foudre, qui favorise la diversité pendant la recherche pour éviter une convergence prématurée. En effet, les particules atmosphériques sont considérées comme des nœuds s'ajoutant, suivant le potentiel local et l'intensité du champ électrique, à un des leaders chargés ascendants ou descendants en fonction de leurs positions soit près du nuage ou près du sol. Ainsi, deux groupes de solutions partielles sont formés. Une fois que les leaders descendants et les leaders ascendants s'interceptent, un chemin de foudre complet est formé pour permettre à un nuage de se décharger au sol. Le trajet de foudre permettant la quantité de décharge maximale est considéré comme la meilleure solution trouvée. Une phase de recherche locale est également intégrée pour améliorer la capacité d'exploitation de l'algorithme. Les expériences réalisées sur le problème du voyageur de commerce montrent un comportement très compétitif de l'algorithme.

La deuxième contribution présente une adaptation de l'algorithme « Fireworks Algorithm (*FA*) » aux problèmes d'ordonnancement à une seule machine. En effet, L'algorithme « Discrete Fireworks Algorithm (*DFA*) » proposé modifie le mécanisme d'exploration et d'exploitation au cours de la phase de génération d'étincelles pour prendre en charge les structures combinatoires. L'algorithme est testé sur le problème de retard total pondéré d'une seule machine. Les résultats expérimentaux sont satisfaisants par rapport aux algorithmes de l'état de l'art. Cependant, certains

problèmes d'exploitation ont empêché l'algorithme de résoudre toutes les instances du problème à l'optimalité. Pour surmonter ces problèmes, une recherche locale à voisinage variable est intégrée. Le nouvel algorithme hybride présente de bonnes performances.

La troisième contribution concerne l'application de *FA* à des problèmes d'optimisation de big data. Ces problèmes sont caractérisés par une dimensionnalité élevée mais aussi un temps de résolution très court est exigé. Pour répondre à la contrainte de temps, l'algorithme original *FA* est revisité pour améliorer la vitesse de convergence tout en préservant sa robustesse. Dans l'approche proposée, deux algorithmes mono objectif et multi-objectifs sont conçus sous la forme d'un Framework d'optimisation appelé FAF. Pour valider ses performances, FAF est configuré pour optimiser le problème de nettoyage des données EEG en tant qu'instance de problème d'optimisation Big Data. Les expériences sont menées en utilisant les instances Big Optimization of EEG Signals (*BigOPT*). Les résultats montrent que les méthodes proposées dans le Framework surpassent toutes les méthodes proposées pour *BigOPT*.

La dernière contribution offre de nouvelles idées sur le développement futur et les tendances dans le domaine des métaheuristiques. En effet, un nouveau design-pattern pour l'utilisation des métaheuristiques est proposé. Ce concept, baptisé SAPIAS, introduit des techniques de configuration automatique d'algorithmes et de sélection d'algorithmes par instance pour une utilisation meilleure et plus intelligente des métaheuristiques dans les applications futures. On s'attend à ce que le concept SAPIAS rassemble davantage d'orientations de recherche pour combler le fossé entre la théorie et la pratique. La recherche théorique peut bénéficier de l'analyse du comportement des composants mis en œuvre et le relier à l'évolution dynamique de l'espace des caractéristiques.

Mots-clés : *Métaheuristiques, Problèmes NP-difficiles, Fireworks Algorithm, BigOPT, SMTWT.*

RESEARCH PAPER LIST

- *EL MAJDOULI, Mohamed Amine & EL IMRANI Abdelhakim Ameer. Lightning Inspired Search Algorithm: Introduction & application to the traveling salesman problem. In: Proceedings of 10th International Conference on Intelligent Systems: Theories and Applications (SITA). IEEE, 2015. pp. 1-6.*
- *EL MAJDOULI, Mohamed Amine et EL IMRANI, Abdelhakim Ameer. Discrete fireworks algorithm for single machine scheduling problems. In: International Journal of Applied Metaheuristic Computing (IJAMC), 2016, vol. 7, no 3, pp. 24-35.*
- *EL MAJDOULI Mohamed Amine, BOUGRINE Saad, RBOUH Ismail & EL IMRANI Abdelhakim Ameer. Fireworks Algorithm for single objective big optimization of signals. In: Proceedings of IEEE/ACS 13th International Conference of Computer Systems and Applications (AICCSA). IEEE, 2016. pp. 1-7..*
- *EL MAJDOULI Mohamed Amine, BOUGRINE Saad, RBOUH Ismail, El Benani Bouazza & EL IMRANI Abdelhakim Ameer. Fireworks algorithm framework for Big Data optimization. In: Memetic computing, 2016, vol. 8, no 4, pp. 333-347.*
- *EL MAJDOULI Mohamed Amine, BOUGRINE Saad, RBOUH Ismail & EL IMRANI Abdelhakim Ameer. A comparative study of the EEG signals big optimization problem using evolutionary, swarm and memetic computation algorithms. In: Proceedings of the genetic and evolutionary computation conference companion. 2017. pp. 1357-1364*
- *El MAJDOULI, Mohamed Amine. SAPIAS concept: towards an independent self-adaptive per-instance algorithm selection for metaheuristics. In : Proceedings of the Genetic and Evolutionary Computation Conference Companion. 2019. pp. 1474-1477.*

CONTENT

GENERAL INTRODUCTION	07
----------------------------	----

CHAPTER I: Background & definitions

1.1	The question of optimization	12
1.2	Problem formulation	13
1.3	Problem modeling	15
1.3.1	Classic modeling	15
1.3.2	The multimodality challenge	17
1.3.3	The multi-objective challenge	17
1.4	Solving approaches for difficult optimization problems	19
1.4.1	Complexity and NP-Hard problems.....	19
1.4.2	Exact methods	21
1.4.3	Approximate methods	22
1.4.3.1	Approximation methods	22
1.4.3.2	Heuristics and metaheuristics	23

CHAPTER II: Metaheuristics

2.1	General design of metaheuristics	26
2.1.1	Solution representation	26
2.1.2	Exploration Vs. Exploitation	27
2.1.3	Performance assessment and measure	28
2.1.4	The no-free-lunch theorem	29
2.1.5	Classification of algorithms	29
2.2	Single-Solution based algorithms	30
2.2.1	Hill-Climbing	30
2.2.2	Simulated Annealing	31
2.2.3	Tabu Search	32
2.3	Population-Based algorithms	33
2.3.1	Evolutionary algorithms	34
2.3.1.1	Genetic Algorithm	34
2.3.1.2	Clonal Selection Algorithm.....	36
2.3.2	Swarm algorithms	38
2.3.2.1	Ant Colony Optimization	38

2.3.2.2	Particle Swarm Optimization	42
2.3.3	Memetic computation	43
2.3.3.1	Cultural Algorithm	43
2.3.3.2	Artificial Bee Colony	45
2.4	Conclusion.....	46

CHAPTER III: A Lightning Inspired Search Algorithm: A novel metaheuristic for unconstrained permutation-based NP-Hard problems

3.1	Introduction	48
3.2	Lightning in nature and simulation models	49
3.3	Analogy and search model	51
3.4	LISA pseudo-code and discussion	54
3.5	Benchmark problem: The traveling salesman problem	54
3.6	Experimental results and discussion	56
3.6.1	Experimental design and parameter settings	56
3.6.2	Results of TSP.....	57
3.7	Conclusion	65

CHAPTER IV: Adaptation and Application of the Fireworks Algorithm to NP-Hard Scheduling Problems & Big Data Optimization

4.1	Fireworks Algorithm	67
4.1.1	General design	67
4.1.2	Sparks generation's control parameters	68
4.1.3	Next-generation selection	68
4.2	A Fireworks Algorithm for the total weighted tardiness problem	69
4.2.1	SMTWT: Problem definition	69
4.2.2	Discrete Fireworks Algorithm for SMTWT Problem	70
4.2.2.1	Solution encoding	70
4.2.2.2	Sparks generation	72
4.2.2.3	DFA pseudo-code	73
4.2.3	Variable Neighborhood Search	73
4.2.4	Results and discussion	74
4.2.5	Conclusion and perspectives	77
4.3	Applying the Fireworks Algorithm to Big Data optimization problems	77
4.3.1	In the age of Big Data	77

4.3.2	BigOpt Problem: Background & related works	79
4.3.3	Fireworks Algorithm Framework for BigOpt	81
4.3.3.1	Framework initialization	81
4.3.3.2	SOFWA: A Single-Objective Fireworks Algorithm	82
4.3.3.3	MOFWA: Multi-Objective Fireworks Algorithm	85
4.3.4	Experimental results & discussion	88
4.3.4.1	Results of SOFWA for BigOPT	89
4.3.4.2	Results of MOFWA for BigOPT	93
4.3.5	Discussion & insights	97
4.3.6	Conclusion	98

CHAPTER V: SAPIAS: Towards Automation

5.1	Introduction: A “Meta-View” of Limitations	100
5.1.2	Automatic Algorithm Configuration	101
5.1.3	Per-instance Algorithm Selection	102
5.2	SAPIAS: A Self Adaptive Per-Instance Algorithm Selection	103
5.2.1	Prediction layer: Defining the per-instance Algorithm Selection	106
5.2.2	Advisor layer: Setting up the solving process	106
5.2.3	AAC layer: Two-Phase Automatic Algorithm Configuration	107
5.2.4	Update layer: Online Update of Performance Space	108
5.3	Questions and challenges	108
5.4	Conclusion	111

GENERAL CONCLUSION	112
---------------------------------	-----

REFERENCES	115
-------------------------	-----

GENERAL INTRODUCTION

During the last decades, various important areas such as transport networks, communication and computer systems, resource management, medical technologies, and many others are experiencing a gradual but especially rapid evolution. This evolution is, in part, due to successive attempts to solve the underlying optimization problems [1].

Optimization is the prediction needle between success and failure. Indeed, an “Optimization Problem” aims to describe the need to find so-called “Optimal Solutions” that best meet the quality and management objectives predetermined by a decision-maker. Several aspects characterize optimization problems. One aspect is their nature. An optimization problem can be defined over discrete or continuous spaces. Following this definition, a solving approach can be studied [2]. However, the most important aspect from a computational point of view is the complexity cost. Therefore, two classes of problems are distinguished. One class contains problems where the resolution is not complex; called easy. The other one groups the problems that are complex; called difficult or hard. Despite the importance of the first class, it is often noticed that complex problems are the most met in reality. Moreover, for a good part of these difficult problems, no algorithm that can solve them to optimality, in a reasonable space-time, is yet known [3].

A famous example of this situation is the traveling salesman problem (*TSP*) [4]. By visiting a set of n cities, each once only, we shall find the shortest path connecting all the cities and ending with the starting city. Although the problem statement is easy to understand, the problem is very complex to solve because the number of solutions to enumerate increases exponentially with n , defining the so-called “Curse of Dimensionality”.

To tackle these kinds of problems, two approaches have emerged. The first one is an exact approach that focuses on finding “mathematically proven” optimal solutions [5]. Given the curse of dimensionality, solving algorithms often suffer from high complexity, and therefore, they lack efficiency even in moderate problem instances. This shortcoming is treated by the second approach [6], which is based on “fast” approximate resolutions where the solutions found have a relatively “good quality” without providing any guarantee of the optimality ensured by the first approach.

As time in the business world costs money, Approximate methods, more specifically metaheuristics, start to attract more attention. From a business

perspective, it turns out that metaheuristic-based solvers are a considerable option as they can provide good quality solutions that are near the optimal solution, thus, making the search effort of the guaranteed optimal solution less worthy. For example, using a metaheuristic to approximately solve the supply chain planning to maximize a company's profits by 40% is certainly preferable to a costly search of a proven optimal solution that may finally bring the optimal maximization of the company's profits to only 1% more (optimal solution profit is 41%).

In this perspective, the contributions reported in this thesis concern One: the conception, Two: the adaptation, and Three: the application of metaheuristics to solve difficult optimization problems such as NP-Hard optimization problems. The first contribution concerns *the conception* of a novel metaheuristic for combinatorial optimization problems called the “Lightning Inspired Search Algorithm (*LISA*)” [143]. Indeed, *LISA* uses a constructive mechanism to build promising solutions and efficiently cover the search space. This mechanism is inspired by the natural lightning formation as reported by the models simulating the phenomenon. As lightning is the result of a perfect minimization process of the total charge amount found near or within the cloud by using a linking path between the near cloud and the near ground particles respectively, *LISA* takes advantage of this to solve optimization problems. To validate it, *LISA* is tested with several benchmark instances of the symmetric and asymmetric traveling salesman problem.

Following its increasing success in solving continuous optimization benchmark problems, due to its efficiency, simplicity, and more importantly its rapid convergence to good optimums, the second contribution introduces an *adaptation* of the “Fireworks Algorithm” for combinatorial single machine scheduling problems [144]. Taking advantage of the robust design of the original Fireworks Algorithm, a new adaptation of sparks generation mechanisms is proposed with a novel use of the control parameters. To further strengthen the exploration performance of the algorithm, a hybridization with Variable Neighborhood Search is also implemented. To validate it, the proposed method is tested with several benchmark instances of the single machine total weighted tardiness problem.

The third contribution presents a novel optimization framework based on the *adaptation* of the Fireworks Algorithm for Big Data Optimization problems [145-147]. Indeed, the proposed framework is composed of two optimization algorithms. A single objective Fireworks Algorithm and a multi-objective Fireworks Algorithm are proposed for solving the Big Optimization of Signals problem “BigOPT”, a new variant of the

Big Data Optimization problems. The single objective Fireworks Algorithm adopts a modified search mechanism to ensure rapidity and preserve the explorative capabilities of the original Fireworks Algorithm. Afterward, the algorithm is extended to handle multi-objective optimization of BigOPT with a supplementary special sparks phase and a novel strategy for next-generation selection. To validate the performance of the framework, extensive tests on six EEG datasets are performed. The framework is also compared with several approaches from the recent state of the art.

The last contribution reported by this thesis is a deep reflection on the future *application* of metaheuristics and its algorithmic design pattern [148]. In the age of machine learning, Per-Instance Algorithm Selection PiAS and Automatic Algorithm Configuration AAC have recently gained growing interests. Unlike classical approaches of designing metaheuristics and manually tune their parameters, AAC and PiAS try to automatically configure and run metaheuristics using machine learning techniques. However, these approaches face many limitations. For instance, the performance of these methods is deeply influenced by factors like the accuracy of the underlying prediction model, feature space correlations, incomplete performance space for new problem instances, instances sampling, and many others. As a remedial, We propose a cooperative design pattern at the conceptual level, labeled “SAPIAS”, composed of a self-adaptive online Algorithm Selection system and an offline Automatic Algorithm Configuration system, working together to deliver the most accurate performance. Additionally, SAPIAS is proposed as a methodic concept that the metaheuristics community may adopt to fill in the gap between theory and practice in the field, by providing theoreticians the ability to continuously analyze the evolution of problems’ characteristics and the behavior of solving techniques as well as providing a ready to use solving framework for practitioners.

The remainder of this thesis is as follows: Chapter 1 provides basic mathematical background and definitions to start solving an optimization problem, including classification of problems, their continuous and combinatorial nature, optimization typologies such as mono, multi, and many objectives along with a description of some necessary solving approaches in order to introduce the role of metaheuristics and their utility.

Chapter 2 describes in more detail the basic foundations of metaheuristics and the role of the inspiration of natural phenomena in their development as well as the role of the search strategies used. We will then proceed to an extensive state-of-the-art

of existing methods with an attempt to compare different classifications of meta-heuristics.

Chapter 3 describes the novel “Lightning Inspired Search Algorithm” (LISA) for unconstrained permutation-based NP-Hard problems. Indeed, LISA takes advantage of the discrete lightning propagation pattern formation, as described by simulation models and scientific observations reporting the charge minimization process during lightning flashes, to offer a promising constructive mechanism capable of both exploring and exploiting the search space. To confirm LISA’s performance, the traveling salesman problem “TSP” belonging to the NP-Hard transportation and logistics problems class is used as a benchmark.

Chapter 4 reports the research work of optimizing two difficult problems using the Fireworks Algorithm (FA). The first is of a combinatorial nature often encountered in logistics and the management of production chains, called the single machine total weighted tardiness problem. The second problem is of great medical importance, entitled the Big Optimization of ElectroEncephalography Signals which deals with noise filtering in electrical signals emitted by the brain. The problem is expressed in two formulations, the first one is mono-objective while the second is multi-objective. We propose a Framework based on the FA algorithm for both formulations, then we report an extensive comparison study with other metaheuristic strategies.

In Chapter 5, we outline the basics of AAC and PiaS approaches. The generic conceptual framework called SAPIAS is then proposed, combining PiAS with AAC to build an autonomous and automatic system for metaheuristics. The various advantages of SAPIAS and the possible challenges are also discussed.

Right after, we present a general conclusion of the various works carried out and their perspectives, we then discuss the various possible directions of future research.

Chapter I

Background & definitions

1.1 The question of optimization

Nowadays, Optimization became a big issue in our life. Whether in engineering, industry, or management, decision-makers need to be assisted consistently when facing an optimization problem by providing them reasonable solutions. For this reason, three important interconnected phases are to consider (*Figure 1.1*).

- *Phase One: Formulating the problem.*

In this phase, an optimization problem is identified. Afterward, an initial statement of the problem is made by formulating a so-called “*objective function*”. This formulation can be imprecise and may need to involve the decision-maker to outline the internal and external factors of the problem at hand.

- *Phase Two: Modeling the problem.*

An abstract mathematical model [7] is built for the problem. This mathematical model can be different from the formulation of the previous phase. Usually, the model describes a simplification of the formulated real problem and its objective function, in the perspective of mapping it to a well-studied optimization problem for which a solving method or procedure is already known. This mapping operation may induce approximations and sometimes omits complex processes described by the formulation.

- *Phase Three: Optimizing the problem*

A solving method is selected depending on the developed mathematical model in the previous phase. The algorithm designer is confronted with either reusing already known algorithms or inject problem-specific knowledge into the solving process by conceiving new methods, using existing ones, or adapting them to the problem requirements. The solving procedure generates a solution that is optimal or suboptimal. It is important to remember that the model being solved is a representation of the real-world problem. Thus, the more accurate the representation is, the more suitable are the solutions found from a real-world feasibility perspective.

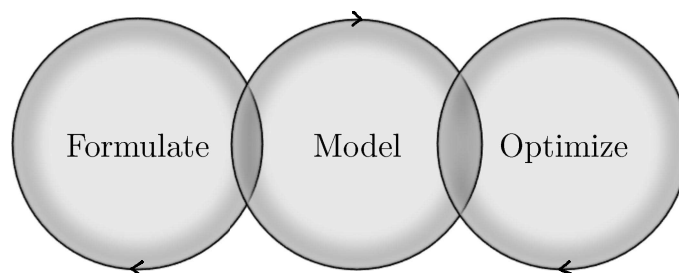


Figure 1.1 General optimization phases

1.2 Problem formulation

Before starting to formulate a problem, we need to define what an optimization problem is.

Definition 1.0 Optimization problems.

An optimization problem can be formally expressed as follows:

$$\begin{array}{ll} \text{optimize } f(s) \\ s \in S \\ \text{subject to: } g_i(s) \end{array} \quad (1.1)$$

where “optimize” stand for either minimizing or maximizing a function f over the set of solutions S . g_i is a set of constraint functions where $i = \{1, 2, 3 \dots n\}$. Without the loss of generality, it is also possible to express every minimization problem as a maximization problem and vice versa as follows:

$$\max_{s \in S} f(s) \xLeftrightarrow{\text{coincide}} \min -f(s) \quad (1.2)$$

where “max” stands for maximizing and “min” for minimizing.

Definition 1.1 Decision Variables

A solution s of an optimization problem is composed of a set of decision variables. A decision variable c is a problem component defined over a decision space D where the modification impacts directly the solution.

The nature of an optimization problem is defined by the type of its solution set structure. Thus, we distinguish two optimization categories: continuous and discrete.

Definition 1.2 Continuous optimization

A continuous optimization problem is the problem of optimizing a function f defined over a solution set S , where all decision variables are defined over a continuous set most frequently represented by $\mathbb{R}^n$.

Definition 1.3 Discrete optimization

Without loss of generality, a discrete optimization problem is a problem of optimizing a function f defined on a solution set S where at least one of the decision variables is defined over a discrete set of elements.

Definition 1.4 Combinatorial optimization

A combinatorial optimization problem is a discrete optimization problem where each solution s , in the set S , is a combination of the decision variables c_i in the decision subset D' of a discrete set D .

Once the nature of the problem at hand is outlined, defining an objective function is the next step. The objective function f is a key element in the optimization process as it mathematically describes the goal of the optimization.

Definition 1.5 Objective Function.

An objective function f associates to each solution s of the set S , a corresponding value q in a set Q that describes the quality of s .

$$\begin{aligned} f : S &\rightarrow Q \text{ such that } f(s) = q \\ s &\in S \\ q &\in Q \end{aligned} \tag{1.3}$$

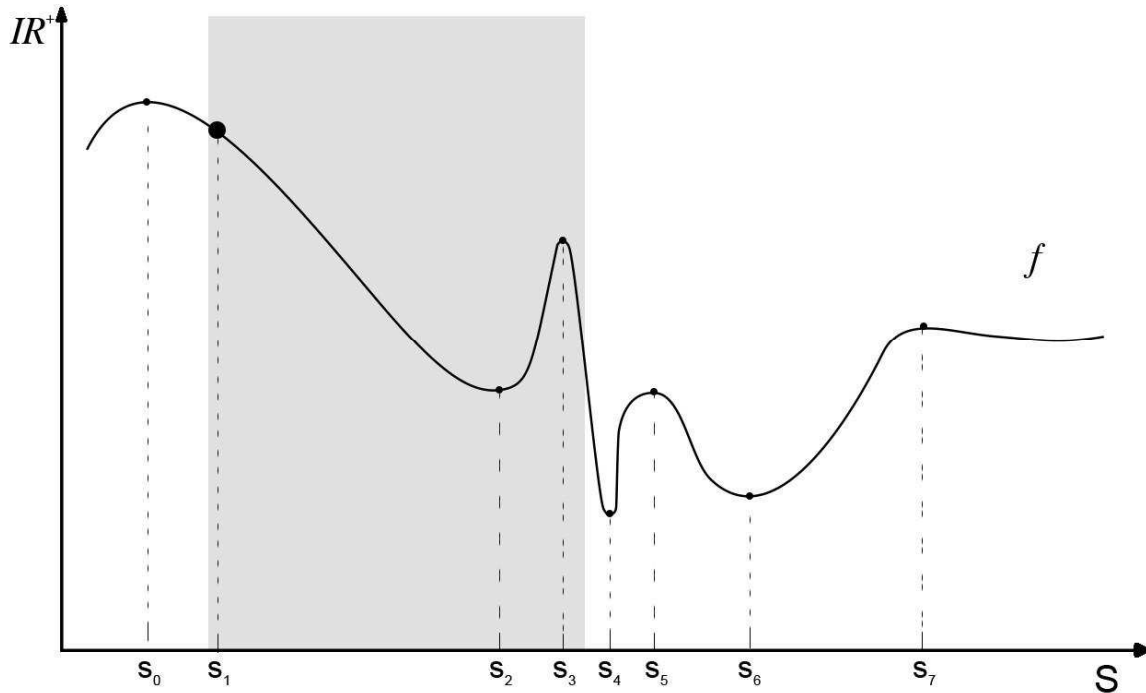


Figure 1.2 Example of optimums in a One-dimension objective function

Indeed, this allows the transformation of S from a simple set to a space of ordered elements. The space of reals $\mathbb{R}$ can (but not necessarily) represent Q for most applications even when the optimization problem is a binary decision problem.

Definition 1.5 Global optimum

A solution $s^* \in S$ is a global optimum if it has a better objective value $f(s^*)$ than all solutions of the search space, that is:

- in the case of a minimization problem: $\forall s \in S, f(s^*) \leq f(s)$
- in the case of a maximization problem: $\forall s \in S, f(s) \leq f(s^*)$

The goal of any optimization is to find a solution(s) called *optimum*(s) or optimal solution(s). For example, let's suppose the objective function in *Figure 1.2*.

Each solution s_i , $i = 0, 1, \dots, 7$ is an *optimum*. If the goal is to minimize f , then clearly the solution s_4 is the one to pick as it has the lowest objective value. So, s_4 is called a *global minimum*. Similarly, if the goal is to maximize f , then s_0 is the optimal solution. It is called then a *global maximum*. However, if only the range of points between s_1 and s_3 is considered, then the minimum value is s_2 . Knowing that s_4 is the global minimum, s_2 is then called a *local minimum*. Applying the same analysis for maximizing, s_1 is called a *local maximum*. The notion of local optimality is tightly connected with the notion of distance. The distance defines a range in which a paradigm of local optimality of a given solution s is measurable in a subset of solutions. For the previous example, s_3 is not an optimum because it is not a local minimum neither a local maximum in the range defined. However, in the range between s_3 and s_5 , it is a local maximum and consequently an optimum.

A first formulation of the problem is achieved once the nature of the problem and the objective function are defined. In the next step, the problem modeling phase starts.

1.3 Problem modeling

1.3.1 Classic modeling

The goal of this phase is to accurately prepare the previously formulated problem for the resolution phase. As the optimization field is continuously accumulating new methods, we need to reformulate the problem to match as much as possible a mathematical model. The reformulation is often referred to as “*classic modeling*” and must satisfy every assumption the mathematical model requires. This is particularly important because, for each defined model, several solving procedures or algorithms are already known. Indeed, numerous models used for this purpose are classified in *Figure 1.3*.

Among the most used classes, usually *Mathematical Programming (MP)* and *Constraint Programming (CP)* are used. One of the famous mathematical programming models is *linear programming (LP)*, formulated as follows.

Definition 1.6 Linear programming model

A mathematical programming model is a linear programming model if:

$$\begin{aligned} \min \quad & c \cdot x \\ \text{subject to: } & A \cdot x \geq b \\ & x \geq 0 \end{aligned} \tag{1.4}$$

where x is a vector of continuous decision variables, c and b (resp. A) are constant vectors (resp. matrix) of coefficients such that $c \cdot x$ and $A \cdot x \geq b$ are linear functions.

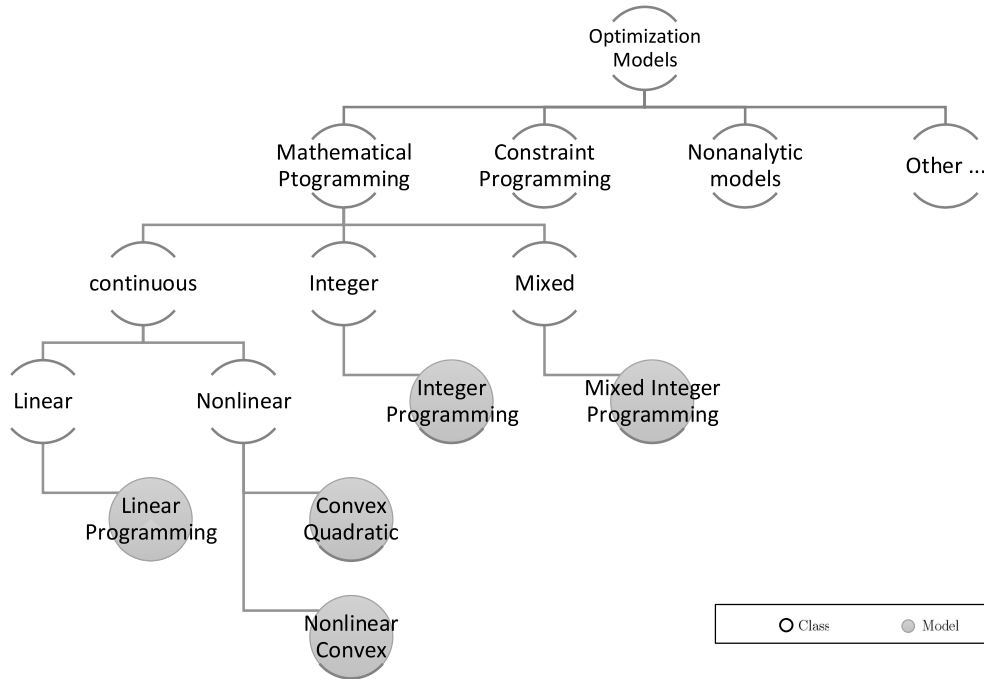


Figure 1.3 Representation of optimization models classes tree

The linearity property of both the objective function $c \cdot x$ and the constraints $A \cdot x \geq b$, makes them convex and the feasible region of the problem becomes a convex set. These characteristics guarantee for the solving algorithms to find a global optimum. In general, there is no reason to use other modeling approaches when a problem is successfully modeled as an LP continuous problem.

In the absence of linearity in the objective function or one of the constraints, another popular model can be used called the *nonlinear programming model*. In this case, the modeling focuses on linearization techniques, which generally inject new variables and/or constraints and add approximations to some degree [9]. This can be further complicated when the variables are defined over discrete spaces like in scheduling and supply chain planning problems. In this case, the modeling goal is to model the problem

as an *Integer programming problem*, or a *mixed integer programming problem* if the decision variables are defined on both continuous and discrete spaces [10].

Generally, there are many ways to mathematically model an optimization problem as the figure above suggests. However, techniques such as MP require explicit mathematical assumptions for which the availability may not be possible in many real-world problems, thus, making these techniques deprecated. Additionally, factors like high dimensionality, non-differentiability, multimodality, multi-objectives, and others make it even more problematic for obtaining optimality.

1.3.2 The multimodality challenge

Many real-world optimization problems are natively multimodal. We express by multimodality the problem characteristic of having multiple global optimums or local optimums as opposed to a single best optimum in classical optimization. A formal statement can be given as follows.

Definition 1.7 A multimodal optimization problem

An optimization problem, with an objective function f and a global optimum s in the solution set S , is multimodal if:

$$\exists s' \neq s \text{ such that } f(s) = f(s') \text{ where } s, s' \in S \quad (1.5)$$

The solving algorithms of the beforementioned models encounter many difficulties to find all global optimums due to the iterative approach of the solving process. In the absence of the linearity assumptions, the algorithms need to perform multiple runs using niching methods built on top to discover more optimums, however, no guarantee is provided. Further details on niching methods can be found in [134].

1.3.3 The multi-objective challenge

Another problem that arises in the modeling phase is the number of objective functions to optimize. Indeed, many problems in practice are multi-objective by nature and that makes the classic modeling even more complex.

Definition 1.7 A multi-objective optimization problem

A multi-objective optimization problem (MOP) can be stated as follows:

$$\min F(s) \text{ such that } F(s) = (f_1(s), \dots, f_n(s)) \quad (1.6)$$

$$s \in S$$

where n ($n \geq 2$) is the number of objectives, the decision space S represents the set of feasible solutions, and $F(x)$ is the vector of objectives to be optimized.

This introduces another level of difficulty, particularly when trying to answer the question of how to quantify solutions' qualities composed of many objective values and how to order them. This is critical to some solving approaches in order to guide the optimization process. The question becomes more complex when the objective functions are conflicting. To overcome this difficulty, a partial order relation, said *Pareto dominance*, is proposed [11]. Thus, an optimality measure can be obtained using the concept of *Pareto optimality*.

Definition 1.8 Pareto dominance

An objective vector $u = (u_1, \dots, u_n)$ is said to dominate $v = (v_1, \dots, v_n)$ (denoted by $u < v$) if and only if no component of v is smaller than the corresponding component of u and at least one component of u is strictly smaller, that is,

$$\forall i \in \{1, \dots, n\}: u_i \leq v_i \quad \wedge \quad \exists i \in \{1, \dots, n\}: u_i < v_i \quad (1.7)$$

Definition 1.9 Pareto optimality

A solution $s^* \in S$ is Pareto optimal if: $\forall s \in S$, $F(s)$ does not dominate $F(s^*)$, that is, $F(s) \not< F(s^*)$.

Searching for such optimality may lead to a set of many non-dominated solutions known as the *Pareto optimal set*. The notion of *Pareto front* is also introduced to represent the image of the Pareto optimal set in the objective space where each solution is represented by its objective vector. The Pareto front is a very important notion as it drives the optimization process. Graphically, the Pareto front is very informative for a decision-maker and gives a clear visualization of each solution quality in the objective space as shown in Figure 1.4.

Definition 1.10 Pareto optimal set

For a given MOP (F, S) , the Pareto optimal set is defined as:

$$P^* = \{s \in S \mid \nexists s' \in S \text{ such that } F(s') < F(s)\} \quad (1.8)$$

Definition 1.11 Pareto front

For a given MOP (F, S) and its Pareto optimal set P^* , the Pareto front is defined as:

$$PF = \{F(s), s \in P^*\} \quad (1.9)$$

Nevertheless, this visualization and performance metrics are challenged when the number of objectives is more than three like in many-objective optimization. Many techniques have been developed to overcome this limitation to some extent including heatmaps [12] and others [13]. However, the issues are still an open question of research.

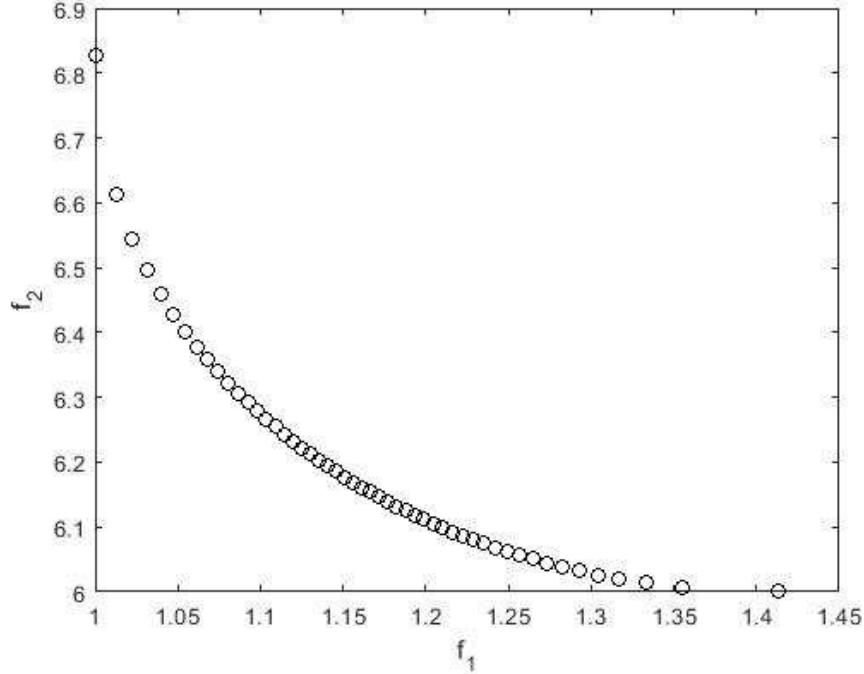


Figure 1.4 A Pareto front example of a bi-objective problem

1.4 Solving approaches for difficult optimization problems

1.4.1 Complexity and NP-Hard problems

The complexity of a problem is equivalent to the complexity of the best algorithm solving that problem [8]. Space and time are the main factors an algorithm relies on during the solving process. Studying the time complexity of an algorithm aims to predict the number of steps required to solve a given problem. The time complexity is usually defined in terms of an asymptomatic bound analysis often represented by the popular *Big-O* notation defined below. However, there might be other notations for the complexity analysis such as *Big-Ω* and *Big-Θ* notations.

Definition 1.12 *Big-O* notation.

An algorithm has a complexity $f(n) = O(g(n))$ if there exist positive constants n_0 and c such that:

$$\forall n > n_0, f(n) \leq c \cdot g(n) \quad (1.10)$$

A problem is said *tractable* (or easy) if there exists a polynomial-time algorithm to solve it. A problem is said *intractable* (or difficult) if no polynomial-time algorithm exists to solve the problem.

Definition 1.13 Polynomial-time algorithm.

An algorithm is a polynomial-time algorithm if its complexity is $O(p(n))$, where $p(n)$ is a polynomial function of n . A polynomial function of degree k can be defined as follows:

$$p(n) = a_k \cdot n^k + \dots + a_j \cdot n^j + \dots + a_1 \cdot n + a_0 \quad (1.11)$$

where

$$a_k > 0 \text{ and } a_j \geq 0, \forall 1 \leq j \leq k - 1.$$

An optimization problem can be transformed into a decision problem by introducing a threshold value y in which the solution is either *yes* or *no*. In the case of a minimization problem, the equivalent decision problem would be: Does a feasible solution x exist where $f(x) \leq y$.

Generally, decision problems are classified depending on their solving algorithm's worst-case complexity. Thus, two complexity classes are considered: P and NP .

The class P represents all decision problems that can be solved by a deterministic algorithm in polynomial time. Decision problems belonging to class P are relatively “easy” to solve. The complexity class NP represents the set of all decision problems that can be solved by a nondeterministic algorithm in polynomial time. A nondeterministic means that there are no particular rules used in the solving process. Clearly, for each problem in class P , there is a nondeterministic algorithm solving it. Thus, $P \subseteq NP$. However, whether $P = NP$ is still an open question. A decision problem $A \in NP$ is *NP-complete* if all other NP problems are polynomially reducible to A .

Definition 1.14 Polynomial reducibility.

A problem A is polynomially reducible to a problem B if there exists a polynomial-time algorithm that transforms all inputs of A to inputs for B .

An *NP-hard* problem is an optimization problem whose associated decision problem is *NP-complete*. Most real-world optimization problems are *NP-hard* for which provably efficient polynomial time algorithms do not exist “yet”. They require exponential time to be solved.

Definition 1.14 Exponential-time algorithm.

An algorithm is an exponential-time algorithm if its complexity is $O(c^n)$, where c is a real constant strictly superior to 1.

Many popular academic problems are *NP-hard* such as the traveling salesman problem, graph coloring, knapsack, and others. In this thesis, a particular interest in this class of problems is given by contributing to the effort of their resolution.

Big Data Optimization problems are also a very recent and important type of problems. These problems often deal with a huge mass of data that needs to be optimized under a small time-window constraint. In the absence of efficient algorithms for optimality, this constraint forces the use of approximate optimality.

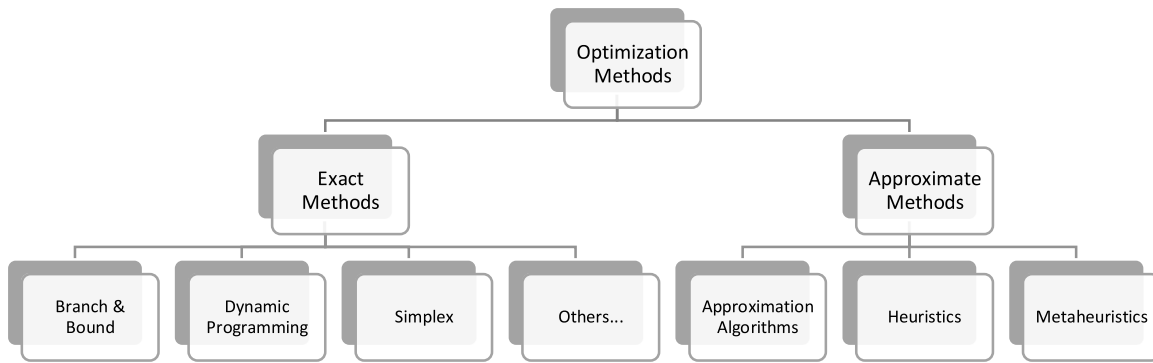


Figure 1.5 Optimization approaches classification

Depending on the problem complexity, the solving process can be either an exact or an approximate method, as illustrated in *Figure 1.5*. Exact methods deliver proven global optimum. Nevertheless, for *NP-hard* problems, exact algorithms are nonpolynomial-time. Approximate methods, however, generate high-quality solutions in a reasonable time but without any guarantee of finding a global optimum.

1.4.2 Exact methods

In the class of exact methods, one can find many classical algorithms. For problems successfully modeled using mathematical programming, the Simplex algorithm, introduced by Dantzig [14], is commonly used, especially for linear programming models. Although it is known for its good performance, the simplex algorithm's worst-case complexity is bounded by an exponential function. Additionally, the simplex algorithm relies on the assumptions of linear programming, which makes it very hard to be extended to other problems where the analytical assumptions are not available.

For nonlinear programming models, the problems are often approximated under convexity assumptions and solved using convex optimization or quadratic programming techniques [15]. In other cases, the Nelder and Mead Simplex algorithm may be used [16]. If such methods are not suitable, techniques like Branch and Bound are used.

Branch and bound (B&B) [17] is an enumerative technique relying on continuously deriving sub-problems of the original problem with convex or linear approximations to achieve a so-called lower bound. The use of bounds with the value of the current best solution enables the algorithm to cover the solution space implicitly. Initially, only one subset exists, which is the solution space. The unexplored subspaces are represented as nodes in a dynamically generated search tree. Each iteration has three main components: Selection of the node to process, Bound calculation, and Branching.

Dynamic programming (DP) [18] is a similar approach. It is based on the recursive division of a problem into simpler subproblems. Unlike B&B, the procedure avoids a total enumeration of the search space by pruning partial decision sequences that cannot lead to the optimal solution.

To find an optimal solution for an NP-Hard problem, methods such as B&B and DP can be extremely time-consuming as the number of nodes can be too large. Thus, using an approximate method becomes a wise decision where two subclasses of algorithms are considered: approximation algorithms and metaheuristic algorithms.

1.4.3 Approximate methods

1.4.3.1 Approximation methods

Approximation algorithms are problem-dependent algorithms that provide provable solution quality and provable run-time bounds. Williamson et al. [19] define an approximation algorithm for an optimization problem as follows:

Definition 1.2 An α -approximation algorithm.

an α -approximation algorithm is a polynomial-time algorithm that produces a solution whose value is within a factor α of the value of an optimal solution. α is called the performance guarantee.

In practice, approximation algorithms need complex transformations to bring mathematical rigor. This requires generally expensive algorithmic techniques and complex data structures leading to implementation difficulties. Additionally, the provided guarantee might not be considered when it shows to be relatively far from

optimality. However, this guarantee can be very helpful in studying the behavior of the problem at hand and solving procedures like heuristics. Indeed, the approximation guarantee can be seen as a lower worst-case bound indicator that can be integrated into heuristic methods as a problem knowledge. The algorithms can be used to understand and predict whether a heuristic can well perform on a given problem or not.

1.4.3.2 Heuristics and Meta-heuristics

“Heuristic” is an ancient Greek word that means find or discover. Formally, a heuristic in optimization is any procedure or strategy used to discover a set of solutions. In the absence of classical modeling and approximation, a simple heuristic would be to start with a solution s_0 , enumerate other solutions, compare them and pick the best one. As the solution space of the NP-Hard problem can be very large, only a small subset of solutions is enumerated. Then, how do we define this subset? One can proceed by defining a distance measure between solutions. All solutions found within a given distance d from s_0 are enumerated. The subset is then called a *neighborhood* of s_0 . We call *heuristic* any procedure or search strategy that can perform such an enumeration. To achieve that, heuristics need to have a detailed knowledge of the solution space structure. Thus, heuristics are mostly problem-dependent which makes their extension to other problems a challenging task.

Let’s consider the following function representing an NP-Hard minimization problem in *Figure 1.6*. Applying the same logic described above, the process of enumerating solutions will iteratively drive the solving process to better quality solutions. However, the solving process, in this case, is not efficient because it stops when no ameliorating solution is found. This is the case when the local optimum s_1 is reached and better solutions are left undiscovered. To overcome this problem, a supervising strategy can be used to control d in order to include s_2 in the current solution’s neighborhood or restart the search within an area such as the highlighted grey area. This is exactly the task of metaheuristics.

A metaheuristic [20], is a generic problem-independent algorithmic pattern that controls the behavior of the underlying heuristics. The metaheuristics field is a major field in stochastic optimization. Indeed, enormous attention and scientific efforts have been dedicated to this field in the last three decades. Engineering design, aerodynamics, Machine learning, telecommunications, and many other fields involve the use of metaheuristics. This is because most real-world problems are NP-Hard. Thus, none of

the before-mentioned mathematical models and enumerative techniques can efficiently solve them. While mathematical programming and enumerative methods choose to drop the running time from their performance measure to satisfy optimality, metaheuristics focus on finding satisfactory solutions in a reasonable time. However, no guarantee of optimality neither an approximation to it is provided.

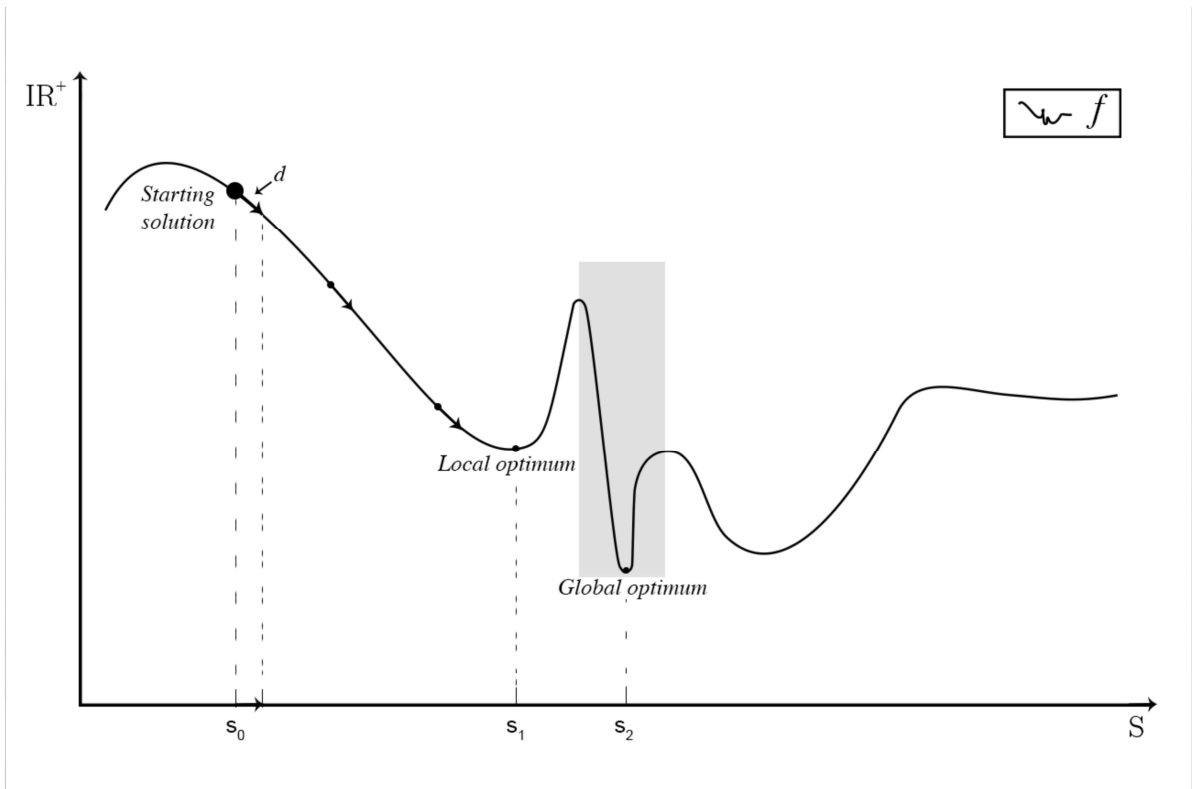


Figure 1.6 Illustration of a simple search heuristic

Chapter II

Metaheuristics

2.1 General design of metaheuristics

Generally, a metaheuristic is a simple generic algorithmic design composed of three main components. These components cooperate to form an algorithm where until a stop-criteria is reached. Three stages are performed:

- Provide or construct an initial candidate solution(s). This is ensured by the *Initialization component*.
- Iteratively transform (stochastically) the candidate solution(s), to produce different candidate solution(s). This is done by the *Perturbation component*.
- Select (stochastically) which candidate solution(s) to retain and which to reject for the next Perturbation, this is performed by the *Selection component*.

However, before executing these stages, many prerequisites are needed. The first one is the solution structure the algorithm would operate on. The second one is the search space coverage strategy, which defines how the algorithm would loop upon the space of solutions. Finally, a guiding function that would guide the algorithm in its navigation throughout the search space.

2.1.1 Solution representation

The computational structure of a solution is a key concept in metaheuristics design. As metaheuristics need to initialize and perturb solutions, the chosen structure, also called representation, plays a major role in demonstrating the efficiency of a search strategy. Upon the definition of a solution representation, many crucial consequences are determined such as the distance between two solutions, the move from a solution to another, the perturbation operations procedures, and others. Additionally, the chosen solution encoding should be suitable to handle the optimization problem characteristics. Indeed, several representations might be suitable for the same problem. The main properties of a representation are Completeness, Connexity, and Efficiency.

- Completeness: Every solution in the search space is fully considered.
- Connexity: Given two solutions S_1 and S_2 of an optimization problem, there is at least one search path to attain S_2 from S_1 and vice versa.
- Efficiency: The representation must not increase the complexity of the perturbation procedures.

Generally, two classes of representations are distinguished. Linear representations and non-linear representations. The first one is mainly a vector-based representation where, depending on the optimization problem, the encoding is implemented using a vector of elements. Classic linear representations can be seen as a set of strings of a given

alphabet. For problems where the decision variables model the absence or the presence of an element such as SAT problems [52], a solution s is encoded in a vector of binary elements where:

$$\forall i, s_i = \begin{cases} 1 & \text{if the element is present} \\ 0 & \text{Otherwise} \end{cases} \quad (2.1)$$

For discrete optimization, the same encoding is used where each element of the vector represents a set of discrete decision variables with their corresponding values. The same encoding is also used for permutation-based problems, where the vector represents all decision variables, however, each different sequence of the vector can be a different solution. For global optimization, the same logic is applied except the fact that every decision variable has a fixed position in the vector and the values are real numbers. However, the problem structure can demonstrate some aspects not explicitly supported by classic representations. Thus, non-classic linear representations are used such as Messy, Diploid, Quantum representations, and others. Further details can be found in [21-25].

Non-linear representations are more complex structures where the encoding implementation is mostly carried out by trees, as in the case of genetic programming [26]. In some applications, encoding can be extended to graphs or finite state machines [8].

2.1.2 Exploration Vs. Exploitation

Once a representation is defined, the most important concept of metaheuristics is how to move between solutions using this representation. Two conflicting aspects are defined: exploration and exploitation. To illustrate this definition, let's suppose an algorithm design in which the assumption of an unlimited time for execution is given. For exploration, the metaheuristic design should ensure that the resulting algorithm is capable of reaching every search region of the search space. Similarly, for exploitation, the design ensures the coverage of every single solution in each region. As one cannot have such an assumption in reality, the whole intelligence of a metaheuristic is how to efficiently explore promising regions and exploit them to find good quality solutions, either implicitly or explicitly as illustrated in *Figure 2.1*. Thus, as time is limited, a tradeoff is introduced between exploring regions in the search space and exploiting a specific region when the solutions demonstrate good qualities. So, how do we define

such a tradeoff? Surprisingly, this tradeoff is, in most cases, inspired by natural phenomena such as intelligent natural organisms, cooperative swarms, etc...

This tradeoff is often implemented in the Perturbation and the Selection components. Indeed, using the solution representation, adequate operators are used to navigate in the search space by introducing modifications to the current candidate solutions being inspected in the perturbation component. Similarly, the Selection component can also implement the tradeoff by allowing or blocking solutions from certain regions using a so-called *fitness function* generally represented by the objective function of the problem at hand.

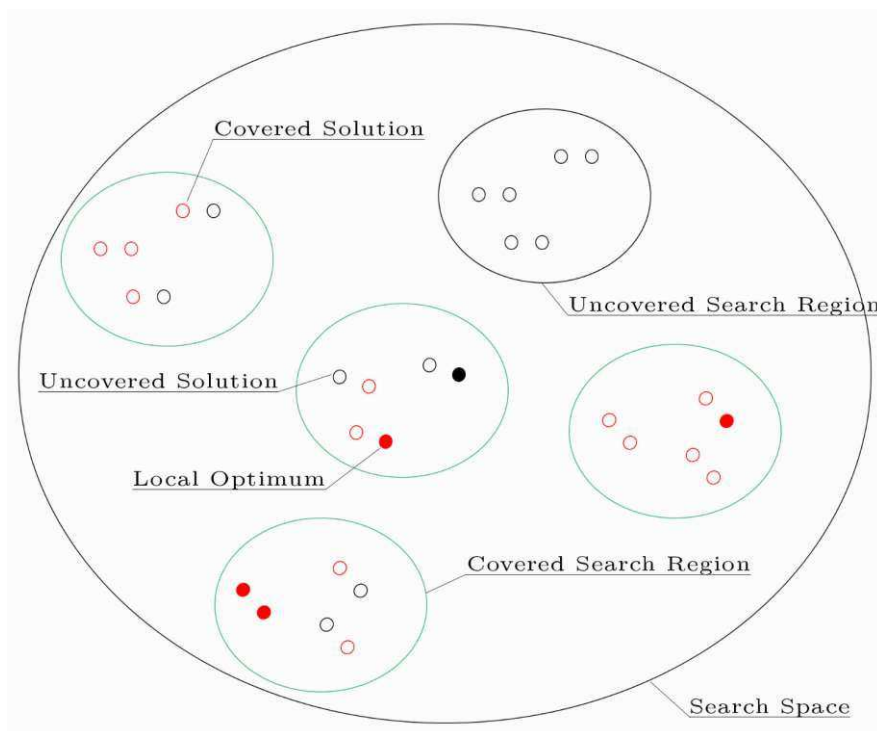


Figure 2.1 - A good coverage of a metaheuristic by discarding weak regions and exploring all promising regions and exploiting the majority of solutions

2.1.3 Performance assessment and measure

As metaheuristic algorithms are mostly stochastic methods, a performance assessment should be defined to evaluate the performance, but also to be able to compare the results between algorithms. Thus, three performance indicators can be defined: Quality of solutions, computational effort, and robustness.

For the quality of solutions, the resulting solutions of a metaheuristic are compared to the global optimal solution of the problem. In the case of the absence of a global optimal solution, the best-known solution is used as a baseline. The computational effort

represents the average computational time required by the algorithm to produce similar results or the average number of fitness function evaluations. As for robustness, some statistical indicators such as the best, worst, mean, percent deviation are commonly used in the literature.

2.1.4 The no-free-lunch theorem

Various metaheuristics have been proposed in the literature. A question would arise pointing out the need for many metaheuristic methods to solve instances of the same optimization problem because a universal best performing method doesn't exist. Indeed, an impossibility theorem, called the no-free-lunch theorem [27], states that "a general-purpose universal optimization strategy is impossible, and the only way one strategy can outperform another is if it is specialized to the structure of the specific problem under consideration". That means no single metaheuristic can outperform every other metaheuristic in all problems. This situation caused an unprecedented growth of metaheuristic methods in the literature.

2.1.5 Classification of algorithms

Several attempts [28] have been made to introduce a classification of different metaheuristics. Four classification approaches are distinguished:

- **Deterministic versus non-deterministic algorithms**

A deterministic metaheuristic performs deterministic solving decisions, which means starting the optimization process with the same initial solution will always end by finding the same final solution. On the other hand, non-deterministic metaheuristics use randomness during the search, thus, a different solution may result from the same initial solution in a new optimization attempt.

- **Iterative versus greedy**

This class identifies the algorithms using their solution construction strategy. Greedy algorithms iteratively build a solution. At each step, a variable of the problem is assigned until a complete solution is obtained. While iterative algorithms start with a complete solution or a set of solutions then apply a series of transformations using algorithmic techniques called operators.

- **Nature-inspired versus non-nature inspired algorithms**

Nature inspiration is by far the most known aspect of metaheuristics. A part of the literature uses this classification for distinction. The source of inspiration includes systems from biology such as immune systems and genetics, natural phenomena from physics such as annealing, and many others.

- **Single-Solution versus Population-based algorithms:**

In this class, algorithms are distinguished by the number of solutions they handle at each iteration. In single-solution based algorithms, only a single solution is used. This allows the algorithm to focus on local regions around the solution using a more explorative approach. On the other hand, population-based algorithms use a set of solutions called a population. A population evolves at each iteration allowing the knowledge exchange between solutions and promoting diversity. Population-based algorithms often target the efficient exploration of the search space to avoid local stagnation.

As many metaheuristics are proposed in the literature, the latter classification is used to introduce the most important metaheuristics among them some are studied in this work.

2.2 Single-solution based algorithms

2.2.1 Hill-Climbing

The Hill-Climbing algorithm [29] is a straightforward technique in metaheuristics. The name comes from its optimization process which iteratively seeks better quality solutions. Hill Climbing uses a local exploitation strategy such as neighborhood enumeration search to explore the search space. Given an initial base solution S , Hill-Climbing tries to modify S using a perturbation strategy to find a better solution S' . Once found, S' becomes the new base solution and so on the process is performed until a stop-criteria is satisfied.

The main part of the algorithm is the perturbation strategy. It is responsible for controlling the exploration and the exploitation capacities of the algorithm. To switch between exploration and exploitation, the parameter *depth* defines the severity of the changes the solution S undergoes. Depending on the problem and the solution representation, a suitable neighborhood structure *Ntype* is chosen for the search heuristic as an input parameter.

Algorithm 2.1 Hill-Climbing

- *Initialize: Parameters(S , $Ntype$, $depth$)*
 - *Repeat:*
 - $S' \leftarrow \text{Perturb}(\text{Copy}(S), \text{depth})$
 - *Repeat:*
 - $S'' \leftarrow \text{GenerateNeighbour}(\text{Copy}(S'), Ntype)$
 - *if* $\text{Quality}(S'') > \text{Quality}(S')$ *then*
 $S' \leftarrow S''$
 - *until stop-criteria is satisfied*
 - *return* S
-

2.2.2 Simulated annealing

The Simulated Annealing method is inspired by the thermodynamic considerations during the annealing process of molten metal cooling. The longer period the metal is cooled, the more chances are given for atoms to form strong crystals, which results in good quality metal. The algorithm is based on the Metropolis algorithm which simulates the generation of physical samples obeying the Boltzmann distribution [30]. Simulated annealing can be seen also as an enhancement of the hill-climbing algorithmic design pattern. Unlike the classical Hill-Climbing, the algorithm is distinguished with its decision to allow, using a probability distribution, the selection of bad quality solutions in order to avoid stagnation in a local optimum. This specificity is implemented using the probability p :

$$p(t, S', S) = e^{(\text{Quality}(S') - \text{Quality}(S)) / t} \quad (2.2)$$

where $t \geq 0$ is a parameter called temperature, simulating the effect of cooling down. If t is high, the probability goes to 1. Thus, S' may still replace S even if it has slightly worse quality than S . If t is close to 0, the probability is close to 0. Thus, selecting a less quality solution becomes very rare.

Algorithm 2.2 Simulated Annealing

- *Initialize: Parameters*($S, Ntype, depth, \mu, t, S_{best}$)
 - *Repeat:*
 - $S' \leftarrow Perturb(Copy(S), Ntype, depth)$
 - *if* $Quality(S') > Quality(S)$ *or* $rand < p(t, S', S)$ *then*
 $S \leftarrow S'$
 - *decrease t:* $t = \mu \cdot t$
 - *if* $Quality(S) > Quality(S_{best})$ *or* $rand < p(t, S', S)$ *then*
 $S_{best} \leftarrow S$
 - *until* $t = 0$ *or a stop-criteria is satisfied*
 - *return* S_{best}
-

As shown in *Algorithm 2.2*, the optimization strategy aims to initiate the search using an explorative approach by setting t to a high number, then slowly decrease it to zero to increase the exploitation capability. Here, the behavior of the algorithm becomes almost similar to Hill-Climbing. This decreasing procedure is controlled using the parameter μ . To keep tracking the best solution found so far, S_{best} is used.

2.2.3 Tabu Search

Tabu Search, introduced in [31], is a metaheuristic that uses a similar mechanism as hill-climbing for exploitation, however, it promotes diversity in the search by focusing on avoiding the already visited solutions which are filled into a memory list called Tabu list l_{tabu} . What differentiates Hill-Climbing and Tabu Search is that instead of comparing only the quality of S and S' , Tabu Search adds the constraint for S' to not be already present in l_{tabu} .

A legitimate question is raised for the Tabu Search paradigm: if it is probable in combinatorial solution spaces to visit the same solution twice, how do we use l_{tabu} in continuous spaces where visiting the same solution is so unlikely? Another question is that even in combinatorial spaces with very high dimensionality, Tabu Search can stay in an exploitation mode when l_{tabu} is very large, there can be still other optimums in the same local space, so how do we solve that?

Algorithm 2.3 Tabu Search

- *Initialize: Parameters($S, l_{tabu}, size, N_{type}, depth, t, S_{best}$)*
 - *Insert S into l_{tabu}*
 - *Repeat:*
 - *$S' \leftarrow \text{Hill-Climbing}(\text{Copy}(S), N_{type}, depth)$*
 - *if S' is not in l_{tabu} then*
 - $S \leftarrow S'$*
 - Insert S' in l_{tabu}*
 - *if $\text{Quality}(S) > \text{Quality}(S_{best})$ or $r < p(t, S', S)$ then*
 - $S_{best} \leftarrow S$*
 - *if $\text{length}(l_{tabu}) > size$ then*
 - remove the oldest element*
 - *until a stop-criteria is satisfied*
 - *return S_{best}*
-

This can be resolved by inserting in l_{tabu} , the information about the perturbation operations applied to S by the hill-climbing procedure to produce S' . However, this approach can be very restrictive for other solutions that may provide promising optimums using these changes. This issue has gained a lot of interest in the Tabu Search community, more details can be found in [32].

2.3 Population-based algorithms

Population-based algorithms are mostly nature-inspired methods. The following classes are distinguished based on their search paradigm: evolutionary computation, swarm intelligence optimization algorithms, and memetic computation.

Evolutionary computation is a conceptual metaphor extracted from a common understanding of how natural systems evolve to help solving complex computational problems. One of the directions this approach considers is the abstraction of the Darwinian theory of evolution where the strong individuals are the ones to probably adapt to extreme conditions and transfer their characteristics to the next generations. Generally, after the initialization of a set of solutions often called a population, an

evolutionary algorithm proposes two iterative phases: Co-evolution and competition. The coevolution phase applies evolutionary operators to solutions producing new (better) ones. The competition phase then decides which among these solutions is likely to help the search process finding additional promising regions and avoid premature convergence.

The Swarm intelligence approach mainly simulates a cooperative behavior between the set of solutions called the Swarm. Each solution of the swarm adjusts it-self according to the information provided by the other members of the swarm. Generally, in the swarm intelligence approach, no selection mechanism is explicitly implemented. The swarm evolves in an equal manner where each solution influences one or more other solutions in the swarm.

Memetic Computation represents the class of algorithms where any stochastic population-based approach is combined with a separate local search improvement or a learning phase with (or without) problem-specific knowledge.

As the state of the art is too large to be covered in this chapter, the following representative algorithms of each class are summarized in the next paragraphs.

2.3.1 Evolutionary algorithms

2.3.1.1 Genetic algorithm

Proposed by Holland et al. [33], genetic algorithms are one of the most popular evolutionary algorithms. Inspired by the natural systems evolution theory and genetics, the genetic algorithm uses the paradigm of survival of the fittest. A set of solutions, called a population of individuals (P), is initialized, then an iterative process starts modifying the population by exchanging available information or exploiting the solutions near space with the goal to find new better-quality solutions. At each iteration, often called generation, these modifications are performed using Reproduction, Crossover, and Mutation operators. Reproduction, also known Selection, deals with selecting appropriate individuals, called parents, to undergo a Crossover operation. The Crossover operation represents an information exchange between the individuals of the population where new solutions are created, called offspring, by exchanging parts of the solutions between the selected individuals. A Mutation operation is then performed to introduce randomized information to diversify the search directions among the offspring. A basic genetic algorithm pseudo-code can be given as follows, where ρ_c is the crossover rate, ρ_m the mutation rate, d is the problem dimension.

Algorithm 2.4 Basic Genetic Algorithm

- *Initialize: Parameters(P , ρ_c , ρ_m , d)*
 - *while stop-criteria is not satisfied do:*
 - *Select two or more individuals (parents) from population P*
 - *Crossover parents with distribution ρ_c to obtain Offspring*
 - *Replace parents by Offspring*
 - *Return best individual*
-

Genetic algorithms are classically associated with the use of binary solution structures, nevertheless, it is applied to all varieties of solution structures.

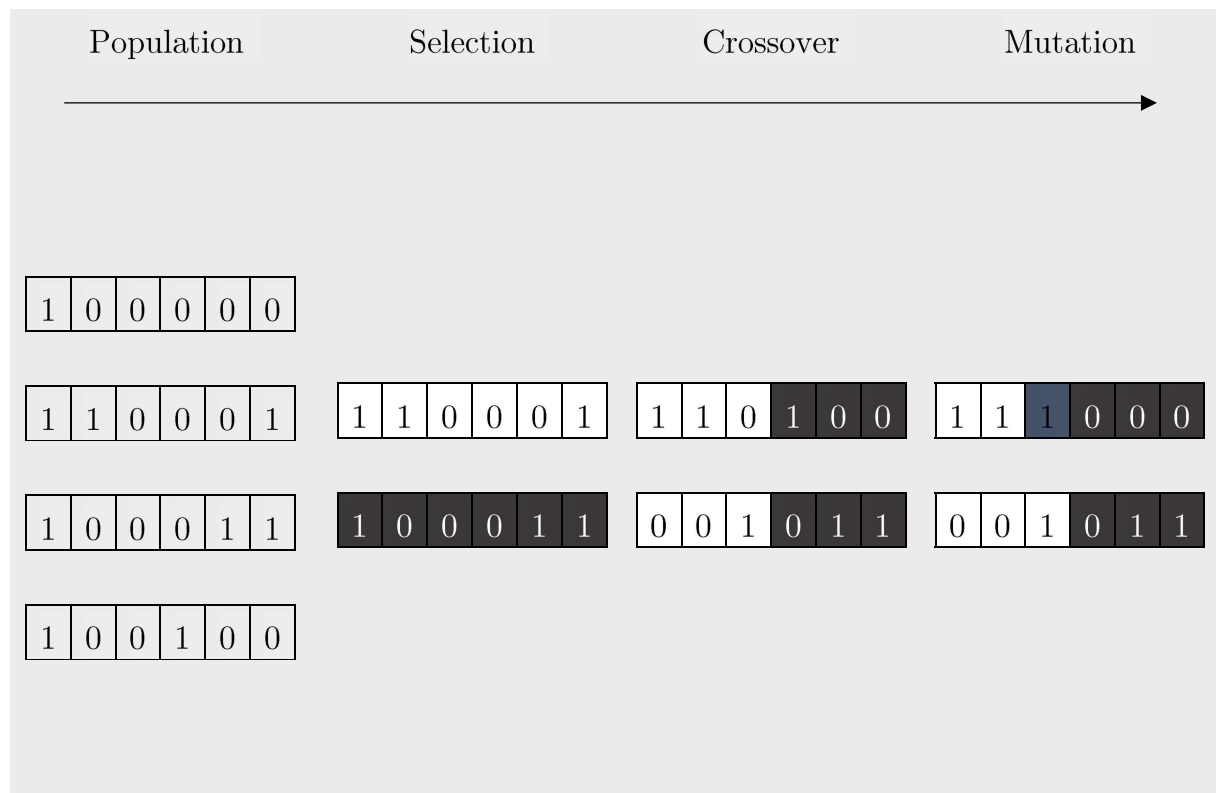


Table 2.2 - Example of selection, crossover, and mutation operations

To illustrate the optimization process, let's consider the following example where the goal of the optimization is to find a solution with maximum bits equal to one. Supposing we have a population P of four initial individuals using a binary structure.

The selection is performed using an elitist selection that is proportional to the quality of the individuals. Many other selection approaches are proposed such as Tournament or Competition, Roulette Wheel, and many others. Further details can be found in [34]. Next, a crossover operation is performed. By using an *n-point* crossover, where every individual is split at the same location ($n=1$). Like selection, many crossover variants, such as uniform distribution crossover, exist in the literature [35]. Afterward, the mutation operation is applied to the offspring. We can see that mutation introduces diversity in the offspring, where fitness varies in $\{0, 1, 2, 3, 5\}$, while we had less variation in the original offspring. The last step is replacing the parents. In a classical Genetic Algorithm, offspring systematically replaces parents in a memoryless mode where we don't keep tracking the best solution found so far. Many other variants have solved this by using one of the selection procedures beforementioned with fixed population size. For further details, a survey paper on genetic algorithms can be found in [36].

2.3.2.2 Clonal Selection Algorithm

The clonal selection theory emphasizes that during the lifetime of any individual, its immune system repertoire undergoes a selection mechanism. The theory states that “on binding with a suitable antigen, activation of lymphocytes occurs. Once activated, clones of the lymphocyte are produced expressing identical receptors to the original lymphocyte that encountered the antigen. Thus, a clonal expansion of the original lymphocyte occurs. This ensures that only lymphocytes specific to an activating antigen are produced in large numbers”. The theory also states that “any lymphocyte that has antigen receptors specific to molecules of the organism's own body must be eliminated during the development of the lymphocyte. This ensures that only antigen from a pathogen might cause a lymphocyte to clonally expand and thus elicit a destructive adaptive immune response” [37]. Thus, the immune system can be viewed as a classifier of antigens into either self antigen or non-self antigen. Non-self antigen are assumed to be from a pathogen and thus must be removed from the body.

During the clonal expansion of B-cells, the average antibody affinity increases for the antigen that triggered the clonal expansion. As explained in [38], “This phenomenon is called affinity maturation and is responsible for the fact that upon subsequent exposure to the antigen, the immune response is more effective due to the antibodies having a higher affinity for the antigen. Affinity maturation is caused by a somatic hypermutation and selection mechanism that occurs during the clonal expansion of B-cells.

Somatic hypermutation alters the specificity of antibodies by introducing random changes to the genes that encode for them”.

The clonal selection algorithm is a straightforward implementation of this immune system mechanism to perform computational optimization. In particular, inspiration has been taken from the antigen-driven affinity maturation process of B-cells, with its associated hypermutation mechanism. De Castro and Timmis [39] highlighted two important features of affinity maturation in B-cells that can be exploited from the computational viewpoint. The first of these is that the proliferation of B-cells is proportional to the affinity of the antigen that binds it, thus the higher the affinity, the more clones are produced. Secondly, the mutations suffered by the antibody of a B-cell are inversely proportional to the affinity of the antigen it binds. A pseudocode of the clonal selection algorithm is given as follows, where P_d is a population of solutions, d is the population size dimension, n_c is the number of clones, ρ is the mutation rate, τ_B is the maximum age.

Algorithm 2.5 CLONALG Algorithm

- *Initialize: Parameters(P_d , d , n_c , ρ , τ_B)*
 - *while stop-criteria is not satisfied do:*
 - *Generate a Clone population P_c by Cloning P_d*
 - *Hypermutate P_c with a rate ρ*
 - *Apply Aging to $P_c \cup P_d$ using maximum age τ_B*
 - *Select d best B cells from $P_c \cup P_d$*
 - *Return best B cell in P_d*
-

By exploiting these two features, De Castro and Von Zuben developed one of the popular clonal selection algorithms called CLONALG, which has been used to carry out the tasks of pattern matching and multi-modal function optimization, but also for large scale optimization problem as reported by Cutello et al. [40]. CLONALG processes the optimization in four main steps: Cloning, Hypermutation, Aging, and Selection. In the first step, a population P_c is created by duplicating each B cell in the initial population P_d n_c times. In the second step, a hypermutation operator is performed over the population P_c . This hypermutation, previously proposed in [41,42], is inversely proportional to the quality of each B cell in P_c . Thus, the number of

mutations decreases if the quality of B is high and vice versa. The parameter ρ is used to calculate the mutation rate. In the third step, the Aging operator is used. It allows the algorithm to mark the B cells prone to elimination due to a higher age. The age here simply represents the number of iterations this B cell has lived. If it exceeds a maximum age τ_B , the B cell is considered dead. The authors make an exception here by allowing the best quality B cell to live longer by resetting its age to zero. A newly generated cloned B cell inherits its parent's age. If it successfully mutates to a better-quality B cell, its age is reset to zero as well. In the last step, the best B cells that survived the Aging operator in $P_c \cup P_d$ are selected to be the population of the next iteration. If only a number $d' < d$ has lived after aging, then a selection competition is performed among the B cells marked as dead to increase the total number of selected B cells to d .

2.3.3 Swarm algorithms

2.3.3.1 Ant Colony Optimization

Swarm Intelligence methods have been successfully adopted due to their competitive performance in NP-Hard optimization. One of the most successful swarm intelligence frameworks is the Ant Colony Optimization “ACO” [43]. One key feature of the success of ACO is the strong inspirational background. In nature, foraging Ants are in charge of searching for a food source and communicate its location to their nestmates. This social communication is indirectly ensured by chemical substances called Pheromones. Indeed, foraging Ants deposit pheromones all along their paths while searching for food and back to their nest. In this way, the nestmates can be easily guided to the food source. By that time, pheromone chemicals evaporate. Thus, only the most used path to the food source stays available as it is receiving more pheromones deposits. This path is found to be effectively the shortest path among all other paths between the food source and the nest. This natural process comprising pheromones deposit, evaporation, and path modification is smartly interpreted to design the optimization model of ACO.

The ACO solutions' search mechanism is often seen as an explorative mechanism [44]. Although it provides some search parameters to control the exploitation behavior, this control stays mainly based on the pheromone's information. The pheromone communication is naturally a long-distance communication and is analogically interpreted as an explorative strategy more than an exploitative one. The ACO model compensates for this by proposing an optional local search phase

- **ACO Search Model**

ACO search model relies mainly on the imitation of the natural chemical communication between Ants. It assumes several artificial Ants constructing solutions to an optimization problem using an associative memory. ACO iterates over three stages. Firstly, Ants construct several solutions using a pheromone trail. Optionally, a local search algorithm is used to improve these solutions. Then, the pheromone trail is updated. The figure below presents the search model of ACO.

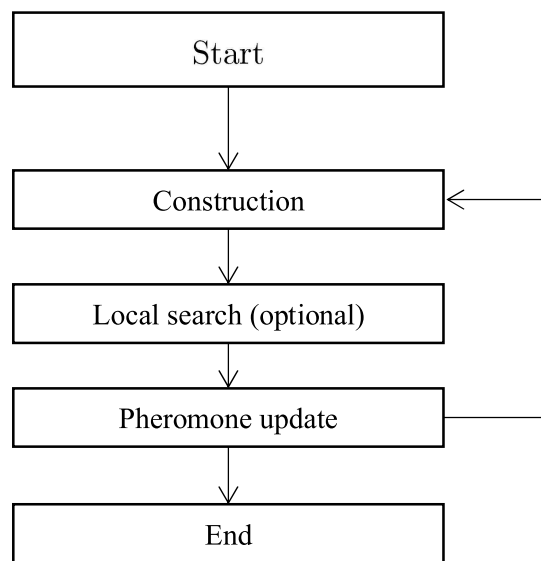


Figure 2.3 - Ant Colony Optimization search model

- **Construction**

- A set of m Ants construct m solutions by iteratively adding a component to a partial solution. The construction begins with an empty partial solution and at each step; an Ant adds a new feasible solution's component until the solution's construction is completed. The choice of a solution component is performed by a stochastic mechanism that uses the pheromone and the information of the optimization problem.

- **Local search (optional)**

- At this stage, a local search algorithm is used to improve the solutions constructed in the previous stage.

- **Pheromone Update**

- When all Ants construct solutions or during solution construction, some of them update the pheromone trail. This update aims to help Ants to construct good solutions in the next iterations.

- **The main ACO algorithms**

The first Ant algorithms to appear in the literature are the Ant System [45], Ant Colony System [46] by Dorigo et al., and Min-Max Ant System [47] by Stützle and Hoos. Since then, many Ant algorithm versions are developed to mainly address the premature convergence and the local optima trapping problems. To explain how this kind of algorithms work, a brief description of the implementation of ACO main algorithms for TSP, notably AS, ACS and MMAS is given.

- **Ant System**

Ant System “AS” is the first proposed algorithm in the ACO framework. To build a solution, an artificial Ant in AS executes n steps moving from a node to another. These moves are ruled by a stochastic mechanism biased by pheromone values. Thus, an Ant k lying in the node i chooses to go to the node j using the probability given by:

$$p_{ij}^k = \begin{cases} \frac{\tau_{ij}^\alpha \mu_{ij}^\beta}{\sum_{l \in N_i} \tau_{il}^\alpha \mu_{il}^\beta} & \text{if } j \in N_i \\ 0 & \text{otherwise} \end{cases} \quad (2.3)$$

where N_i is the set of nodes not visited yet by the Ant k , α and β are two parameters, τ_{ij} is the amount of pheromone trail on the edge (i, j) and μ_{ij} is the heuristic value for switching from a node i to another node j defined as follows:

$$\mu_{ij} = \frac{1}{d_{ij}} \quad (2.4)$$

where d_{ij} is the distance between the nodes i and j .

When all Ants finish their tours, pheromone trails are updated using:

$$\tau_{ij} = (1 - \rho) \cdot \tau_{ij} + \sum_{k=1}^m \Delta \tau_{ij}^k \quad (2.5)$$

where $\rho \in [0, 1]$ is the evaporation rate, and $\Delta \tau_{ij}^k$ is the quantity of pheromone deposited by the Ant k on the edge (i, j) , defined below.

$$\Delta \tau_{ij}^k = \begin{cases} Q/L_k & \text{if ant } k \text{ used } (i, j) \\ 0 & \text{otherwise} \end{cases} \quad (2.6)$$

where Q is a constant, and L_k is the quality of solution constructed by Ant k .

▪ Ant Colony System

Ant colony system is also an ACO algorithm. The difference between ACS and AS lies in the state transition rule and the pheromone update. On one hand, ACS uses the state transition rule called pseudo-random-proportional rule given by:

$$j = \begin{cases} \arg \max_{l \in N_i} (\tau_{il}^\alpha \cdot \mu_{il}^\beta) & \text{if } q \leq q_0 \\ P & \text{otherwise} \end{cases} \quad (2.7)$$

where q is a random number uniformly distributed over $[0,1]$, $q_0 \in [0,1]$ is a parameter, and P is a random variable.

On the other hand, ACS uses two updating rules. The global updating rule and the local updating rule. The global one concerns only the global best solution found by an Ant gb which is defined as follows:

$$\tau_{ij}^{gb} = (1 - \alpha) \cdot \tau_{ij}^{gb} + \alpha \cdot \Delta \tau_{ij}^{gb} \quad (2.8)$$

where $\alpha \in [0,1]$ is the pheromone decay coefficient, and $\Delta \tau_{ij}^{gb}$ is given by:

$$\Delta \tau_{ij}^{gb} = \begin{cases} 1/L_{gb} & \text{if Ant } gb \text{ used } (i,j) \\ 0 & \text{otherwise,} \end{cases} \quad (2.9)$$

where L_{gb} is the length of the best solution found by Ant gb .

Furthermore, the local updating rule is applied while Ants build their solutions using:

$$\tau_{ij} = (1 - \rho) \cdot \tau_{ij} + \rho \cdot \tau_0 \quad (2.10)$$

where τ_0 is the initial value of pheromone.

▪ Min-Max Ant System

The third most known ACO algorithm is the Max-Min Ant System “MMAS”. In its turn, MMAS differs by performing the global updating rule applied for the globally or locally best Ant ($best$) or both using the equation below. Additionally, the pheromone value is bounded by boundaries of the interval $[\tau_{min}, \tau_{max}]$.

$$\tau_{ij} = \left[(1 - \rho) \cdot \tau_{ij} + \Delta \tau_{ij}^{best} \right]_{\tau_{min}}^{\tau_{max}} \quad (2.11)$$

where $\Delta\tau_{ij}^{best}$ is similar to (equation 2.6) using $Q = 1$ and the operator $[x]_a^b$ is defined by:

$$[x]_a^b = \begin{cases} a & \text{if } x > a \\ b & \text{if } x < b \\ x & \text{otherwise} \end{cases} \quad (2.11)$$

2.3.3.2 Particle Swarm Optimization

The Particle Swarm Optimization “*PSO*” is a very popular technique in stochastic optimization developed by Eberhart and Kennedy [48]. *PSO* essentially imitates the birds' flock and simulates their decentralized flying pattern to find a place with a sufficient amount of food.

PSO assembles a set S of cooperative particles of size n , called swarm to essentially collect the best information found by the swarm globally and by each particle locally to integrate it in the solving process as a problem knowledge. Each particle in the swarm has a velocity and a position. As *PSO* was initially developed for high dimensional continuous optimization, each particle's position X is considered as a candidate solution to the problem. Afterward, the problem knowledge sharing is implemented in each particle by its velocity V , which is calculated using the particle's best-known location so far P_{best} and the best-known location G_{best} found by the whole swarm S as follows:

$$X^{t+1} = X^t + V^{t+1} \quad (2.12)$$

where

$$V^{t+1} = V^t + \alpha \cdot (C_1 \cdot (P_{best} - X^t) + C_2 \cdot (G_{best} - X^t)) \quad (2.13)$$

α defines the particle step towards the new position, C_1 and C_2 are weighting parameters to define the contribution of the local optimums P_{best} and G_{best} found by the particle and the swarm respectively.

The tradeoff between exploitation and exploration is ensured by the movement equation (2.12). When C_1 is fairly greater than C_2 , the particle is then intensifying the search in its neighborhood for a better local optimum, whereas a value of C_2 greater than C_1 favors the exploration of farther solutions in the search space. The parameter α defines how strong the effect of the chosen configuration on the new position. Thus,

C_1 and C_2 are very important parameters to control the behavior of the algorithm. A basic pseudo-code of *PSO* is given below.

Algorithm 2.6 Particle Swarm Optimization Algorithm

- *Initialize: Parameters(S, n, α, C_1, C_2)*
 - *while stop-criteria is not satisfied do:*
 - *For every particle P_i do:*
 - *Update Velocity V_i^{t+1}*
 - *Move to the new position X_i^{t+1}*
 - *If $Quality(X_i^{t+1})$ is better than $Quality(P_{best})$ then*
 - $P_{best} \leftarrow X_i^{t+1}$
 - *If $Quality(X_i^{t+1})$ is better than $Quality(G_{best})$ then*
 - $G_{best} \leftarrow X_i^{t+1}$
 - *Return G_{best}*
-

2.3.4 Memetic algorithms

2.3.4.1 Cultural algorithm

The Cultural algorithm “CA” introduced by R. G. Reynolds [49], is considered as a variant of evolutionary algorithms. It is mainly inspired by characteristics and behaviors that have a cross-generation transmission capability during the human social and cultural evolution. To optimize, *CA* uses a model of cultural change where culture is symbolically represented and transmitted between successive populations. The main principle in this process is firstly based on rejecting unacceptable beliefs, and positively valuating and promoting socially accepted beliefs secondly.

In the same context of memetic computation, the Cultural algorithm (*CA*) is considered a memetic algorithm. Besides applying the evolutionary process, Cultural Algorithm uses a separate phase of improvement or learning. At the design level, *CA* defines two components. The population space P is the optimization component where individuals (solutions) can evolve using an evolutionary approach such as a Genetic Algorithm. The belief space B is a generic component where the problem knowledge can be stored, fine-tuned, and reused. Both components coevolve using a so-called Vote-Inherit-Promote protocol. It implements a mapping procedure to ensure:

- One, a Vote function mapping of an individual performance in the population space to the belief space
- Two, an Inherit function learning patterns that have led to such an individual's performance or enhance the qualities of these individuals
- Three, a Promote function reintegrating the new individuals or the knowledge extracted to influence the search in the population space.

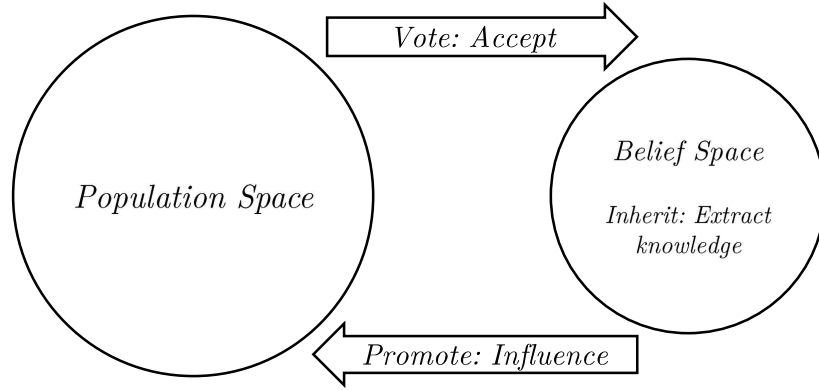


Figure 2.4 - Cultural Algorithm search components

Several approaches can represent the type of knowledge the individuals may acquire during the optimization at the population space level such as normative knowledge where certain value ranges for some decision variables are favored, or spatial knowledge where the landscape of the problem is implicitly extracted during the search using local search operators. The pseudocode of *CA* is given below.

Algorithm 2.7 Cultural Algorithm

- *Initialize: Parameters(P, B)*
 - *while stop-criteria is not satisfied do:*
 - *Map P to B*
 - *Adjust B*
 - *Influence P : Promote knowledge from B to P*
 - *Evolve P*
 - *Return the best individual*
-

2.3.4.2 Artificial Bee Colony

The efficient foraging behavior and food source discovery by bee colonies have been always intriguing researchers. Indeed, bees are known to be highly selective when choosing between different patches of flowers to exploit the most profitable ones. Tereshko and lee [50] tried to model this foraging behavior assuming the roles of three actors: employed foragers and unemployed foragers and nectar (food) sources.

An employed bee is responsible for carrying both the location and the profitability of nectar to the colony members. This information is communicated using a “waggle dance” on an area near the opening of the hive. The length of dance determines the profitability of the nectar. Thus, another forager is employed. Unemployed bees’ mission is to either discover the environment for previously undiscovered nectar sources, in this case, bees are called Scouters, or stay in the hive waiting for an employed bee to follow after its dance, thus called Onlookers.

Based on this model, Karaboga et al. [51] developed the Artificial Bee Colony algorithm (ABC). ABC considers three types of bees: Employed, Onlookers, and Scouter bees. From an inspiration point of view, ABC is considered as a swarm intelligence model as it translates an intelligent collective behavior, however, ABC for solving optimization problems is classified as a memetic computation technique due to its search strategy. Indeed, ABC defines three main search steps. The first one is performed by the employed bees, where each bee modifies a candidate solution represented by nectar. The onlooker bees select a subset of these nectars depending on their quality to further focus the search process. For better performance, onlookers can use, at this stage, problem-specific knowledge and/or local search methods, whence the memetic aspect of the algorithm. The third step, performed by the Scouter bees, mainly deals with selecting the promising nectar to keep on the search for the next iteration of the algorithm.

As described by the pseudocode below, In the initialization phase, a set of nectars P of size n is initialized (solutions) is initialized. Employed bees then start exploiting these nectars by looking at new nearby nectars in the neighborhood and evaluate their fitness. Next, employed bees communicate their nectar information with onlooker bees. Afterward, the Onlooker bees probabilistically choose the nectar, to exploit further, using a fitness-based selection technique such as the roulette wheel selection method. The Scout bees are considered to be employed bees for whom the current nectar is no more profitable and cannot be further exploited once it reached a maximum limit l_{max} . These nectars are abandoned, and Scouters start to search for new nectars randomly.

The process continues iteratively until a stop criterion is satisfied and the best nectar N_{best} is returned. The pseudocode of the ABC algorithm is given as follows.

Algorithm 2.8 Artificial Bee Colony Algorithm

- *Initialize: Parameters($P, n, l = 0, l_{max}, N_{best}$)*
 - *While stop-criteria is not satisfied do:*
 - *Send Employed bees to P (Modify P)*
 - *Send Onlooker bees to further exploit P*
 - *Save best nectar N_{best} in P*
 - *If $l = l_{max}$*
 - *Send Scout bees to replace abandoned nectars in P with new ones*
 - *Increment l*
 - *Return the best nectar in N_{best}*
-

2.4 Conclusion

In this chapter, the necessary mathematical background and most known metaheuristics methods have been presented in the current and previous chapters. These notions and methods are extensively referred to, especially the metaheuristics, which are used for comparison purposes.

It hard to analyze these methods theoretically and independently of a given problem as they are non-deterministic, and their behavior is purely stochastic. Also, a successful method in a given problem does not guarantee its solving supremacy for the simple reason of the fast evolution of problems landscapes in real-world applications. Hence, developing new designs and novel adaptations is a necessity.

In the next chapter, we introduce our first contribution which offers an implementation a new nature-inspired metaheuristic called “Lightning-Inspired Search Algorithm”.

Chapter III

A Lightning Inspired Search Algorithm: A novel metaheuristic for unconstrained permutation-based NP-Hard problems

3.1 Introduction

During the last few decades, nature-inspired metaheuristics have known a noticeable growth due to their successful adoption by the scientific community for their encouraging results in various fields such as optimal design and manufacturing [53], frequency assignment [54], and many others. However, optimality problems have been raised such as being trapped in poor local optima and fast convergence to these bad optima. Also, some methods use complex designs that need special tuning before hybridizing or applying them to other problems. Indeed, except for Ant Colony Optimization algorithms, the majority of the proposed algorithms have been originally designed and initially tested for continuous optimization problems. One of the main problems these methods encounter is how to carefully preserve their exploitation and exploration mechanisms when adapting the search parameters and the solution encoding to a combinatorial space like the permutation-based space.

Therefore, the main goal of this paper is to present a new memetic nature-inspired optimization method for permutation-based NP-Hard combinatorial optimization problems. Inspired by the pattern of lightning propagation, the Lightning Inspired Search Algorithm “LISA” offers an intuitive global search mechanism that couples efficiently with local search techniques to discover high-quality optimums. LISA takes advantage of the discrete lightning propagation pattern formation, as described by simulation models and scientific observations reporting the charge minimization process during lightning flashes.

To confirm LISA’s performance, the traveling salesman problem “TSP”, which belongs to the NP-Hard transportation and logistics problems class, is used. TSP can also model many other important problems. For instance, by the launch of the genome project [55], the research has concentrated on utilizing TSP to study groups of genes or proteins. In a study presented by Johnson and Liu [56], TSP is employed to predict protein functions. Their prediction method was more promising and advantageous than traditional methods. Consequently, providing an efficient solving technique for TSP will positively impact the research in different applications in genomics and proteomics areas.

3.2 Lightning in nature and simulation models

Lightning is a natural phenomenon occurring during thunderstorms and hurricanes due to an electrostatic discharge. The electrification of the atmospheric particles and the charge structure of thunderstorms are the most common sources of lightning.

Essentially, lightning is a transient, high-current electric discharge whose path length is measured in kilometers. It is generally classified into several categories. The main three categories are Cloud to Cloud, Ground to Cloud, and Cloud to Ground lightning. This latter is the most widely studied because of its practical importance for power and communication systems disturbance, injuries and deaths, forest fires, and many others [57].

It happens when one (or many) negatively charged leader called the downward leader,



Figure 3.1 Cloud to Ground lightning

is initiated near or within the cloud, then propagates in a series of discrete steps towards the ground. Consequently, the electric field intensity at the ground becomes sufficient enough to initiate one (or many) upward connecting positive leader, which initiates from particles near the ground and propagates upwards to meet up with a negative downward leader. When the leaders are connected, they form a direct link between the cloud and the ground, allowing a complete discharge of the negatively charged cloud as illustrated in *Figure 3.2*. Similarly, Ground-to-Cloud Lightning follows the same logic. The only difference is the lightning being initiated first from charges near the ground. Finally, Cloud-to-Cloud lightning remains the most occurring in nature as half of all lightning discharge within the thunderstorms, also called intra-cloud discharges. This category is not extensively studied because intra-cloud lightning leaders are difficult to photograph and study with optical instruments [58]. Scientific observations of Cloud to Ground lightning have revealed three major phases: downward leader propagation, upward leader propagation, and leaders' connection.

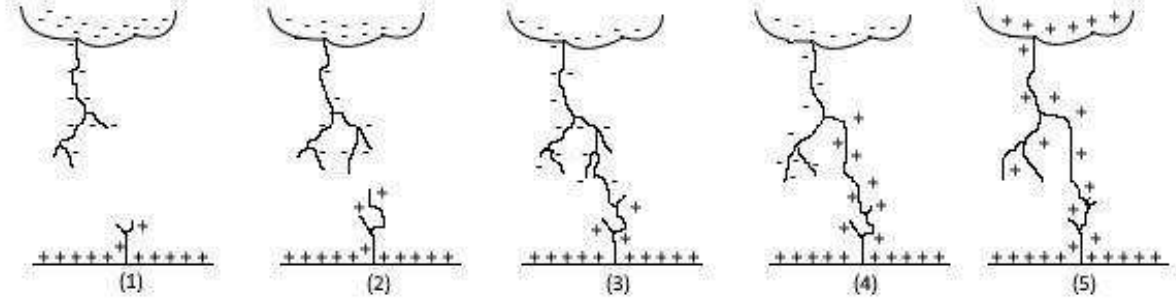


Figure 3.2 Natural lightning illustration

To understand the downward leader propagation, many simulation models have been proposed [59] [60] [61] [62]. Niemeyer et al. [63] proposed the Dielectric Breakdown (DB) simulation model. The DB model suggests that the propagation of the downward leader is following a predefined local potential distribution on all possible propagation points, obtained by solving the discrete Laplace equation for electrical potential P under predefined boundary conditions as follows:

$$\nabla^2 P = 0 \quad (3.1)$$

In other words, the selection of the next propagation point in the DB model is performed based on the local potential at each point obtained by solving the discrete form of *equation 3.1* in two-dimensional space given by:

$$P_{i,k} = \frac{1}{4} (P_{i+1,k} + P_{i-1,k} + P_{i,k+1} + P_{i,k-1}) \quad (3.2)$$

where $P_{i,k}$ is the local potential of a point at the location's coordinates (i, k) . In a one-dimensional space, the local potential should be given as follows:

$$P_i = \frac{1}{2} (P_{i+1} + P_{i-1}) \quad (3.3)$$

The particles having the highest local potentials are iteratively selected. However, the DB model doesn't provide a simulation of upward leader propagation. Thus, Perera & Sonnadara [64] completed this simulation model in their study of the fractal nature of lightning. They suggested that the propagation of the upward leader follows a probabilistic mechanism, which is proportional to the power of the local electric field at each point (particle's location) approximated by:

$$E_{RQ} = \frac{|P_R - P_Q|}{d_{RQ}} \quad (3.4)$$

where P_R and P_Q are the local potential at the points R and Q respectively, and d_{RQ} is the distance separating them. The bigger the value of E_{RQ} , the higher the probability of selecting the point Q.

The simulation continues until the downward-moving leader intercepts one of the upward connecting positive leaders, forming a complete lightning path. The contact between the leaders is the connection phase. In this phase, a high transient current is triggered due to the negative charge movement towards the earth and vice versa, allowing a complete discharge.

3.3 Analogy and search model

It is important to mention that LISA is not a simulation or an artificial lightning optimization system, but rather a memetic search mechanism inspired by the lightning formation and the way it propagates. Roughly speaking, LISA is modeled in three main constructive parts: downward leader construction, upward leader construction, and leaders' connection.

Let's first consider a construction space composed of an atmospheric particle system. As we deal with discrete optimization problems, the construction space has a finite number of elements which is the problem dimension N . Consider a solution S to this system that is a lightning path which maximizes the discharge amount $g(S)$ given by:

$$g(S) = \frac{\gamma}{f(S)} \quad (3.5)$$

where $f(S)$ is the solution quality represented by the lightning path length and γ is a constant parameter. From *equation 3.5*, it is clear that the shorter the lightning path, the bigger the discharge amount. As shown in *Figure 3.3.a*, to construct the lightning path, we need to specify particles that are near the cloud and those near the ground where both the downward and the upward leaders are going to propagate. For this purpose, the construction space is divided into two subspaces: Near Cloud particles' (NC) subspace and Near Ground particles' (NG) subspace (*Figure 3.3.b*). This subdivision is controlled by a parameter α that is defined as the cardinal of the NC subspace. Implicitly, the cardinal of the NG subspace is $N - \alpha$. Next, the value of the local potential on each particle's location is calculated by transforming the solved Laplace equation into one-dimensional space (*equation 3.3*). The transformation assumes that the potential in a particle's location is approached using the distances from neighbor particles as described in *equation 3.6*.

$$P_i = \frac{1}{2} (d(P_i, P_{i-1}) + d(P_i, P_{i+1})) \quad (3.6)$$

Once potentials are obtained, the lightning leaders' propagation can start. As the initiation of leaders is not yet understood in reality, a random particle is chosen as a starting point on both subspaces for both propagations. Afterward, adding particles from NC subspace to the downward leader is performed by selecting iteratively the particle having the best local potential value. Right after, an upward leader propagation is started on NG space by adding, iteratively, particles with the highest probability which depends on the local electric field power value (*Figure 3.3.c*). The particle having the best value of local electric field power is selected, but only accepted and added to the leader if a uniformly generated number less than 1 is bigger than a predefined parameter β in $[0,1]$, otherwise, a random particle is selected.

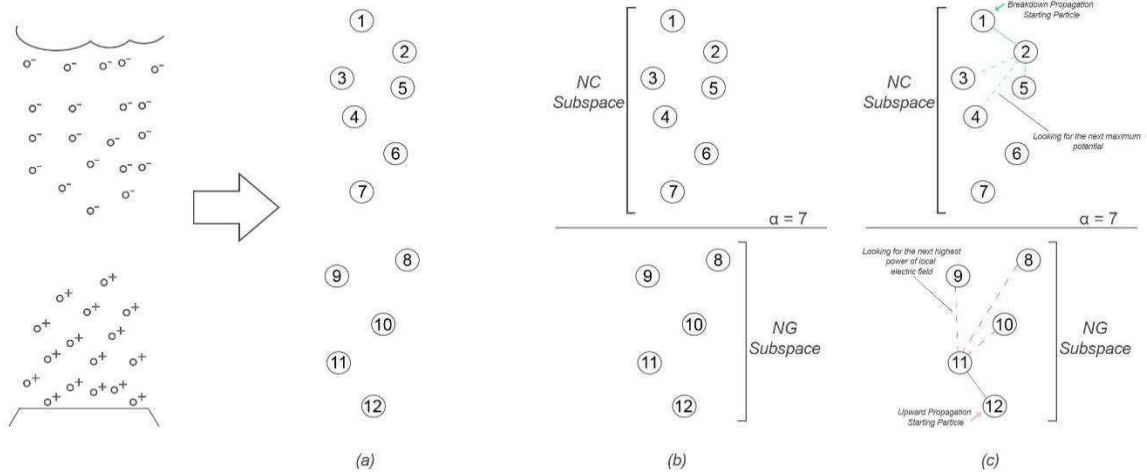


Figure 3.3 NC and NG Subspaces definition

As summarized in *Figure 3.4*, the third step consists of finishing the solution construction by connecting the downward leader with the upward leader to form a lightning path. The most important operation at this stage is the particles' movements to exchange the charge. In fact, almost all particles in the NG space will move towards the NC subspace and vice versa (*Figure 3.4.c*). This is very important in the search model as it represents the lightning current triggering and permits LISA to change the search region and consequently to have a good explorative capacity. Next, an exploitation stage using a fine-tuning mechanism, or a local search heuristic is performed to balance the search process. Finally, a global update is performed to memorize the best solution, and the new configuration is set for both NG and NC subspaces for a new propagation cycle using the new particles' distribution of the best-found solution of the current iteration (*Figure 3.4.d*). The propagation, connection, and

local tuning operations are repeated until a satisfying number of propagation cycles k is reached.

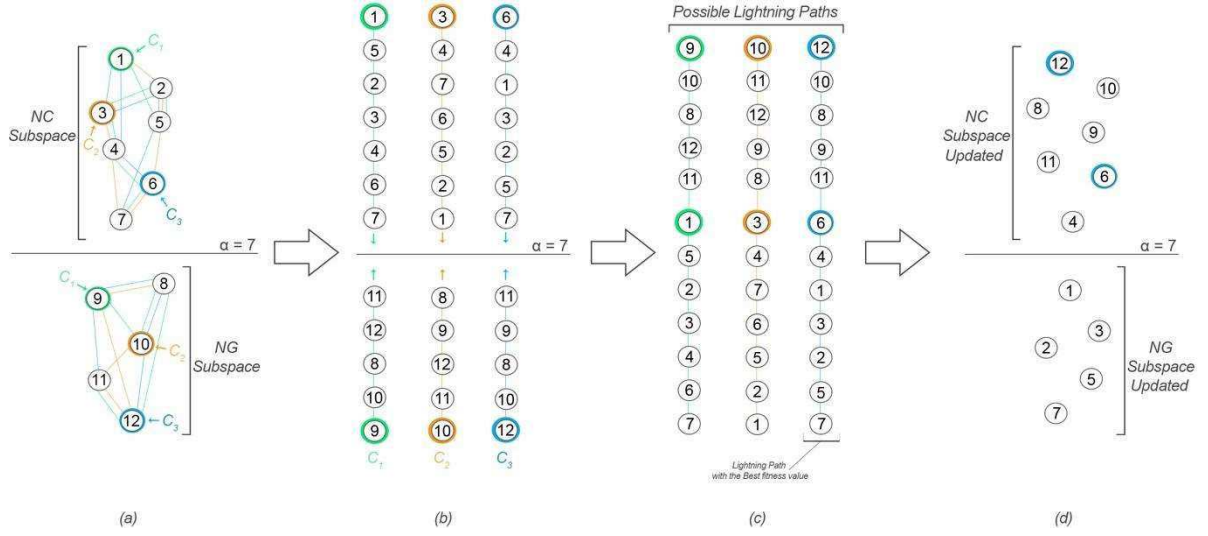


Figure 3.4 Lightning generation and subspaces update in LISA

This number of cycles models the fractal nature of the real word lightning (Figure 3.4.a), where each cycle $C_i, i = 1, \dots, k$ starts an upward and a downward propagation simultaneously from a random position in their respective subspaces. The flowchart of the search model implemented by LISA is given in Figure 3.5 below.

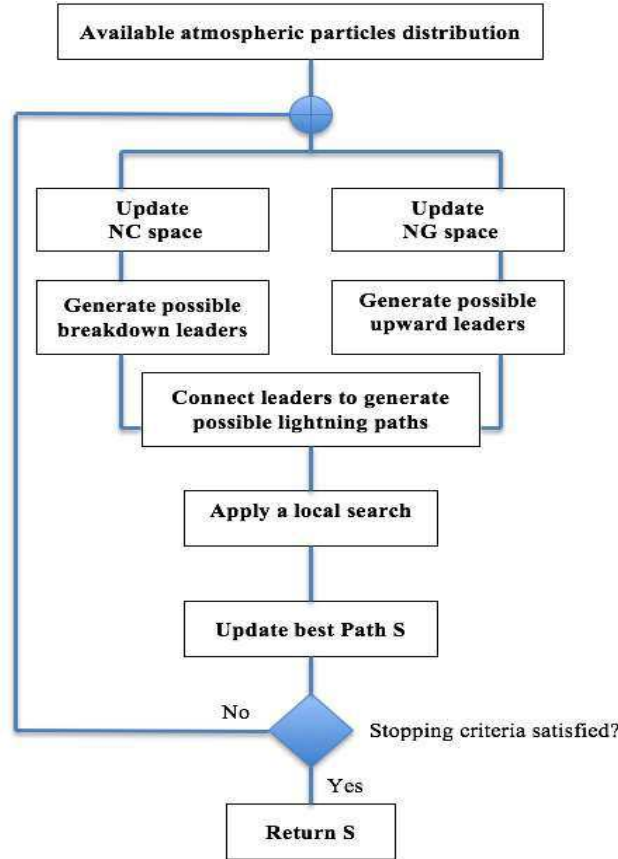


Figure 3.5 Model of LISA

3.4 LISA pseudo-code and discussion

One key feature of LISA is its powerful explorative capacity. Indeed, the search mechanism implemented by LISA allows the construction of solutions using two different strategies at the same time. The first strategy used by the downward leader construction consists of selecting, iteratively, particles with the highest potential values. The potential value is calculated using the distances from neighboring particles. This construction allows LISA to break edges with large distances for a minimization context (or short for a maximization) and recombine new edges that may lead to other promising search regions. The second strategy is the constructive selection based on the power of the local electric field used by the upward leader. This selection acts like a probabilistic nearest neighbor construction. It is used to ensure a good orientation of the solution construction. It is also possible to use directly a distance-based probabilistic nearest neighbor selection to speed up the constructive phase when the difference between the potential values is neglected against the distance in the equation (3.4). The parameter β controls the selection acceptance rate which allows to increase the exploration when initialized with values near one and vice versa.

Furthermore, the connection of leaders performs a particle exchange between the two subspaces which allows the LISA mechanisms to act on new regions of the search space, which results in efficient space coverage. The combination of the two mechanisms provides solutions that can be exploited by the local search phase, so, we obtain good solutions. The LISA pseudo-code is given below (*Algorithm 3.1*).

3.5 Benchmark problem: The Traveling salesman problem

The Traveling Salesman problem can be defined as the problem to find the minimal closed path (also called Hamiltonian path) performing a tour of all cities visiting each one only once. Mathematically, it can be stated as finding the permutation π of cities $c_1, c_2, \dots, c_n$ that minimizes the sum of distances given by:

$$f(\pi) = d(c_n, c_1) + \sum_{i=1}^{n-1} d(c_i, c_{i+1}) \quad (3.7)$$

The distance $d(c_i, c_j)$ between two cities c_i and c_j can be obtained using many metrics, depending on the problem instance. TSP is a NP-Complete problem [64], hence, no efficient polynomial-time algorithm can solve TSP for a large instance-size of N cities. The computational time exponentially increases with the number of cities. It exists $((N-1)!)/2$ feasible tours in this case.

Algorithm 3.1 Lightning-Inspired Search Algorithm

- *Initialize: Parameters(α, β, γ , number of cycles, NC, NG, t , best)*
- *While Stop criteria is false*
 - *For $p = 0$: number of cycles*
 - *Calculate potential distribution on all particles using (2);*
 - *Update NC space with first α elements of best temp solution;*
 - *Update NG space with $N - \alpha$ elements of best temp solution;*
 - *Init array downward leader with a random element from NC space;*
 - *Init array upward leader with a random element from NG space;*
 - *For $i = 1 : \alpha$ in NC space*
 - *Choose the NC particle with the best local potential value;*
 - *Add the chosen particle to downward leader;*
 - *End For*
 - *For $j = 1 : N - \alpha$ in NG space*
 - *Choose the NG particle with the best local electric power value;*
 - *Generate a number “rand” uniformly in $[0,1]$*
 - *If rand $> \beta$*
 - *Add the chosen particle to upward leader;*
 - *Else*
 - *Add a random particle to upward leader;*
 - *End If*
 - *End For*
 - *Connect leaders to form a new solution S ;*
 - *Apply a local search on S ;*
 - *If $g(S) > g(\text{best})$*
 - *Update best solution with S ;*
 - *Update charge amount $C - g(S)$;*
 - *End If*
 - *End For*
 - *End While*
 - *Return best*

To verify the performance of LISA, TSP is used as a benchmark problem to assess and compare the achieved results with other state-of-the-art algorithms. In the following, a summary of recent studies on TSP. Zhou [65] used the classic Lagrangian of the dual

problem for the TSP and obtained the conclusion that the classic Lagrangian may not be applicable to solve it. Hence, the use of metaheuristics is of valuable importance. TSP has been tackled by several metaheuristics. Beginning with one of the most used which is the Ant Colony Optimization. Dorigo et al [66] initially proposed an artificial ant system simulating the real ants' process using chemical pheromones to find the shortest path leading to food starting from their nest. Stützle et al [67] proposed a Min-Max Ant System with pheromone control parameters. Later on, Le Louarn et al [68] replaced the probabilistic nearest neighbor heuristic with an alternative insertion procedure known as GENI heuristic. Guo et al [69] proposed a moderate ant colony by designing a novel transition rule inspired by the adaptive behavior of some *Monomorium* ant species. Recently, Escario et al [70] introduced a task division between ants with a regulation policy using a state-space exploration approach instead of the classical construction graph. Besides ACO, Genetic Algorithms have been also widely applied to TSP. To increase the GA performance, Albayrak et al [71] developed a new mutation operator based on a greedy search method. Victor Paul et al [72] revisited and analyzed the population seeding techniques and proposed three ordered distance-based initialization techniques. Moreover, many nature-inspired algorithms have been proposed for solving TSP. Karaboga et al [73] developed a combinatorial artificial bee colony. Zulaiha et al [74] presented a water flow-like algorithm. Ouavarab et al [75] [76] proposed a discrete Cuckoo search using a random key encoding scheme to switch from continuous to combinatorial space. Osaba et al [77] adapted a discrete Bat-inspired algorithm. Zhou et al [78] proposed a discrete invasive weed optimization algorithm combined with two local search operators. Other research focused on hybridizing methods like Genetic Algorithms and Simulated Annealing with particle swarm optimization and Ant Colony Optimization [79] [80] [81] [82]. Neural networks were also proposed for solving TSP. Créput et al [83] introduced a memetic neural network combining the self-organizing map with an evolutionary algorithm. Also, Avşar et al [84] investigated a parallelized neural network system using the divide and conquer approach based on a self-organizing map.

3.6 Experimental results and discussion

3.6.1 Experimental design and parameter settings

LISA is applied to optimize 65 instances of the traveling salesman problem from the TSPLIB. Three distance metrics are used to calculate the edges' weights, depending on the instance type: Rounded Euclidian, Geographical, and ATT distances [85]. The results are extracted from thirty consecutive independent runs, not necessarily the best

neither the worst runs. The best, worst, mean, and best-known solutions “BKS” and running time averages in seconds are reported. PD_{Best} and PD_{Avg} show the percent deviation from the best and average solution’s fitness value respectively. Stdev means the standard deviation. $S_{1\%}/S_{opt}$ is given by:

$$\frac{S_{1\%}}{S_{opt}} = \frac{\text{Number of solutions found within 1\% optimality}}{\text{Number of optimal solutions found}} \quad (3.8)$$

The experiments were run on a 3.2 GHz CPU. After a thorough parameter tuning, the parameter settings are given below:

Table 3.1. Computational results of LISA for small and medium scale TSP benchmark instances

Parameter meaning	Variable in pseudo-code	Value for TSP
Maximum number of iterations	<i>Stop criteria</i>	100
Maximum cycles number	<i>number of cycles</i>	200
α	α	11
β	β	0.7
γ	γ	1

3.6.2 Results of TSP

We classified the TSPLIB instances into three categories to facilitate the analysis of the results. The first category contains small-scale instances where the number of cities is less than 100. Medium-scale instances where cities’ number doesn’t exceed 400 are grouped in the second category and large-scale instances where the number of cities is beyond 400 in the third category. *Table 3.2* shows the results of LISA on the first and the second categories. LISA is performing very well, as it can solve small-scale instances to optimality in all runs in a very competitive time. *Figure 3.6* shows the variation of the best, worst, and average time over each instance. We notice that the average time is always closer to the best time which means that LISA could solve the instances very quickly in most runs. As for medium-scale instances, LISA could find the optimal solutions of 19 out of 27 instances, 42% of them have been solved to optimality in all runs. The eight unsolved instances have an average error that doesn’t exceed 0.5%.

LISA performance on large-scale instances is very promising as reported in *Table 3.3*, knowing that LISA had a running time limit of 1800s. For PD_{Best} and PD_{Avg} , we can say that more than 83% of large instances are within 2% optimality. This means the best solutions found by LISA approximate in less than 2% the best-known solutions of large instances.

Table 3.2 *Computational results of LISA for small and medium scale TSP benchmark instances*

Instance	BKS	Best	Worst	Average	Time _{Avg.}	PD _{Best}	PD _{Avg.}	S _{1%} / S _{Opt}	Stdev
ulysses16	6859	6859	6859	6859.00	0.00	0.00%	0.00%	30/30	0.00
ulysses22	7013	7013	7013	7013.00	0.00	0.00%	0.00%	30/30	0.00
eil51	426	426	426	426.00	0.38	0.00%	0.00%	30/30	0.00
berlin52	7542	7542	7542	7542.00	0.03	0.00%	0.00%	30/30	0.00
st70	675	675	675	675.00	0.28	0.00%	0.00%	30/30	0.00
eil76	538	538	538	538.00	0.68	0.00%	0.00%	30/30	0.00
pr76	108159	108159	108159	108159.00	0.19	0.00%	0.00%	30/30	0.00
gr96	55209	55209	55209	55209.00	2.11	0.00%	0.00%	30/30	0.00
rat99	1211	1211	1211	1211.00	0.75	0.00%	0.00%	30/30	0.00
kroA100	21282	21282	21282	21282.00	0.44	0.00%	0.00%	30/30	0.00
kroB100	22141	22141	22141	22141.00	2.00	0.00%	0.00%	30/30	0.00
kroC100	20749	20749	20749	20749.00	0.36	0.00%	0.00%	30/30	0.00
kroD100	21294	21294	21294	21294.00	1.10	0.00%	0.00%	30/30	0.00
kroE100	22068	22068	22068	22068.00	1.36	0.00%	0.00%	30/30	0.00
rd100	7910	7910	7910	7910.00	1.81	0.00%	0.00%	30/30	0.00
eil101	629	629	630	629.13	3.95	0.00%	0.02%	30/26	0.34
lin105	14379	14379	14379	14379.00	0.51	0.00%	0.00%	30/30	0.00
pr107	44303	44303	44303	44303.00	0.95	0.00%	0.00%	30/30	0.00
pr124	59030	59030	59030	59030.00	0.45	0.00%	0.00%	30/30	0.00
bier127	118282	118282	118313	118283.03	5.28	0.00%	0.00%	30/29	5.56
ch130	6110	6110	6113	6110.10	6.10	0.00%	0.00%	30/29	0.54
pr136	96772	96798	97177	96985.33	19.45	0.03%	0.22%	30/0	82.12
gr137	69853	69853	69853	69853.00	3.32	0.00%	0.00%	30/30	0.00
pr144	58537	58537	58537	58537.00	0.58	0.00%	0.00%	30/30	0.00
ch150	6528	6528	6549	6536.67	19.13	0.00%	0.13%	30/11	8.77
kroA150	26524	26524	26550	26524.93	11.85	0.00%	0.00%	30/27	4.66
kroB150	26130	26130	26143	26130.83	12.60	0.00%	0.00%	30/26	2.60
pr152	73682	73682	73682	73682.00	1.95	0.00%	0.00%	30/30	0.00
u159	42080	42080	42396	42090.53	4.18	0.00%	0.03%	30/29	56.72
rat195	2323	2323	2335	2330.27	35.64	0.00%	0.31%	30/1	3.45
d198	15780	15786	15830	15814.10	40.01	0.04%	0.22%	30/0	10.95
kroA200	29368	29368	29554	29385.03	32.69	0.00%	0.06%	30/20	46.28
kroB200	29437	29438	29565	29495.37	47.44	0.00%	0.20%	30/0	29.46
gr202	40160	40217	40403	40277.27	53.24	0.14%	0.29%	30/0	47.70
ts225	126643	126643	126643	126643.00	3.79	0.00%	0.00%	30/30	0.00
pr226	80369	80369	80496	80401.57	38.44	0.00%	0.04%	30/6	35.43
gr229	134602	134892	135562	135220.10	83.20	0.22%	0.46%	30/0	166.22
gil262	2378	2379	2389	2383.50	74.19	0.04%	0.23%	30/0	2.63
pr264	49135	49135	49135	49135.00	10.12	0.00%	0.00%	30/30	0.00
a280	2579	2579	2602	2582.70	30.08	0.00%	0.14%	30/24	8.11
pr299	48191	48201	48432	48277.43	106.00	0.02%	0.18%	30/0	61.73
lin318	42029	42140	42399	42240.07	144.29	0.26%	0.50%	30/0	71.34

Also, the relatively small values of the standard deviation show the stability of the algorithm. As for very large instances where the number of cities is beyond 2000, the biggest error is within 5% to the best and 6.5% to the average, except on fnl4461 where the percent deviation is around 11%.

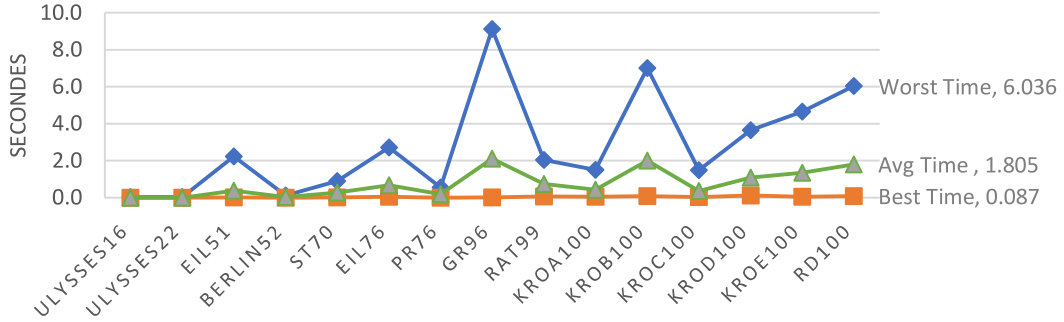


Figure 3.6 LISA Best, Worst, and Average running time

Table 3.3 Computational results of LISA for large TSP benchmark instances

Instance	BKS	Best	Worst	Average	Time _{Avg.}	PD _{Best}	PD _{Avg.}	Stdev
rd400	15281	15352	15486	15417.70	270.59	0.46%	0.89%	33.28
gr431	171414	172812	174454	173676.90	382.20	0.82%	1.32%	339.43
pr439	107217	107453	108888	107768.33	297.04	0.22%	0.51%	284.84
pcb442	50778	51003	51595	51200.93	263.43	0.44%	0.83%	137.98
d493	35002	35301	35561	35432.37	477.52	0.85%	1.23%	61.16
ali535	202310	203561	207628	204677.03	774.55	0.62%	1.17%	1061.99
rat575	6773	6892	6951	6918.27	669.10	1.76%	2.14%	15.11
p654	34643	34693	34731	34709.73	639.65	0.14%	0.19%	10.08
u724	41910	42382	42837	42610.30	1000.0	1.126%	1.671%	114.42
rat783	8806	8944	9051	9002.83	1000.0	1.567%	2.235%	27.23
pr1002	259045	262844	267158	265395.00	1000.0	1.467%	2.451%	932.43
u1060	224094	227606	230447	228818.23	1000.0	1.567%	2.108%	678.64
vm1084	239297	241992	245920	243716.57	1000.0	1.126%	1.847%	873.49
fl1400	20127	20245	20523	20318.40	1000.0	0.586%	0.951%	57.11
fl1577	22249	22352	22735	22507.77	1000.0	0.463%	1.163%	108.51
u1817	57201	58645	59699	59029.3	1800.02	2.52%	3.20%	316.77
d2103	80450	81131	83161	81907.7	1800.02	0.85%	1.81%	729.71
u2152	64253	65815	66589	66332.2	1800.02	2.43%	3.24%	237.64
pr2392	378032	393132	401012	395808.47	1000.0	3.994%	4.702%	2045.47
pcb3038	137694	145309	147794	146545.50	1000.0	5.530%	6.428%	624.41
fl3795	28772	29456	31318	30208.13	1000.0	2.377%	4.991%	442.42
fnl4461	182566	200274	204190	202403.7	1800.22	9.70%	10.87%	1294.54

These results are encouraging, especially when comparing them with other recent methods as shown in *Table 3.4*. LISA outperforms almost all methods except the discrete Cuckoo Search where the best-found solutions seem to be similar, excluding pr136, d198, lin318, pr439. Also, the hybrid approaches combining GA, ACO, PSO, and Simulated Annealing [79] seem to perform better on rat575. However, LISA always gets better averages and standard deviations except on d198 and pr226 where the discrete cuckoo search is better. As the optimal values of all small and medium scale instances are already known, the red values in *Table 3.4* reported in [86] do not represent new optimums but rather a different calculation of the distances between nodes. In fact, the authors forgot to round up the distance between two nodes as described in [85]. Also, the red value cited in [77] is probably a printing error as the optimal solution is 22141.

Figure 3.7 compares the computational time. To fairly present the comparison, LISA was run in the same platform used by Discrete Bat algorithm (DBA). Additionally, *Figure 3.8* shows the quality of the results regarding the obtained averages of LISA and the “Fast Opposite Gradient Search with Ant Colony Optimization” (FOGS-ACO) for TSP [87]. The Figures confirm the superiority of LISA in providing good solutions in a shorter time.

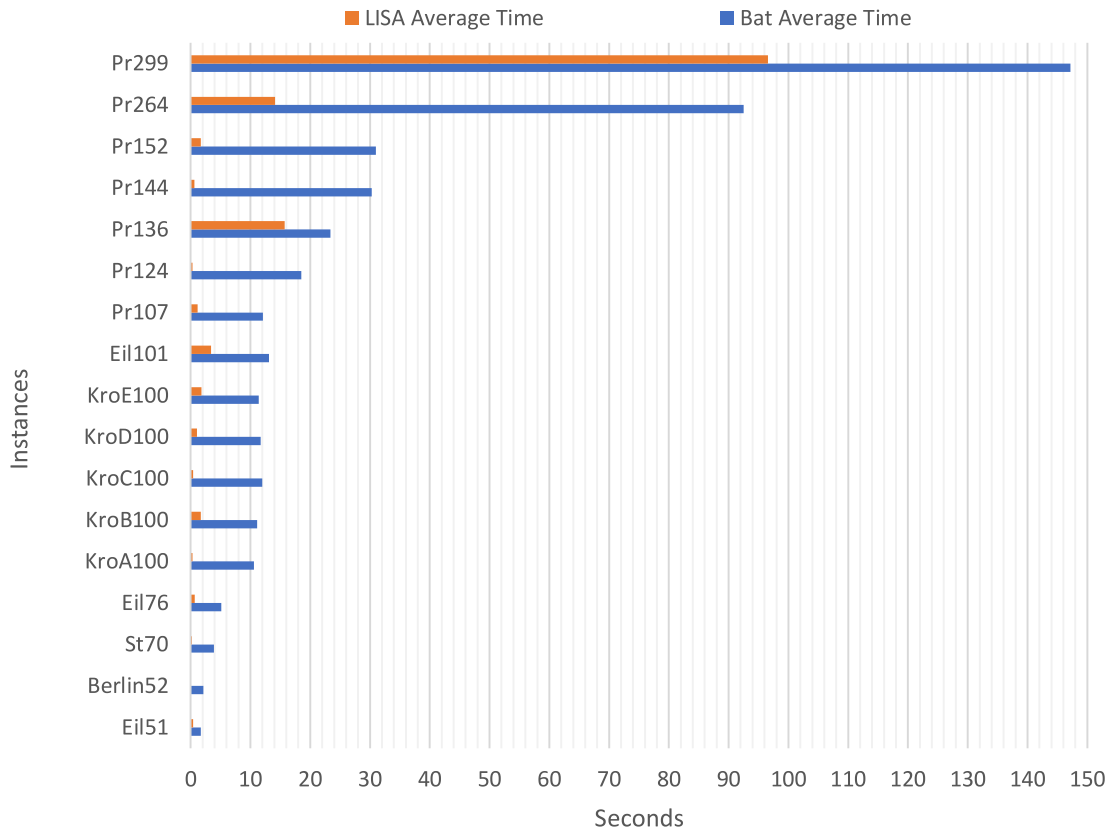


Figure 3.7 Running time averages of LISA and Discrete Bat Algorithm

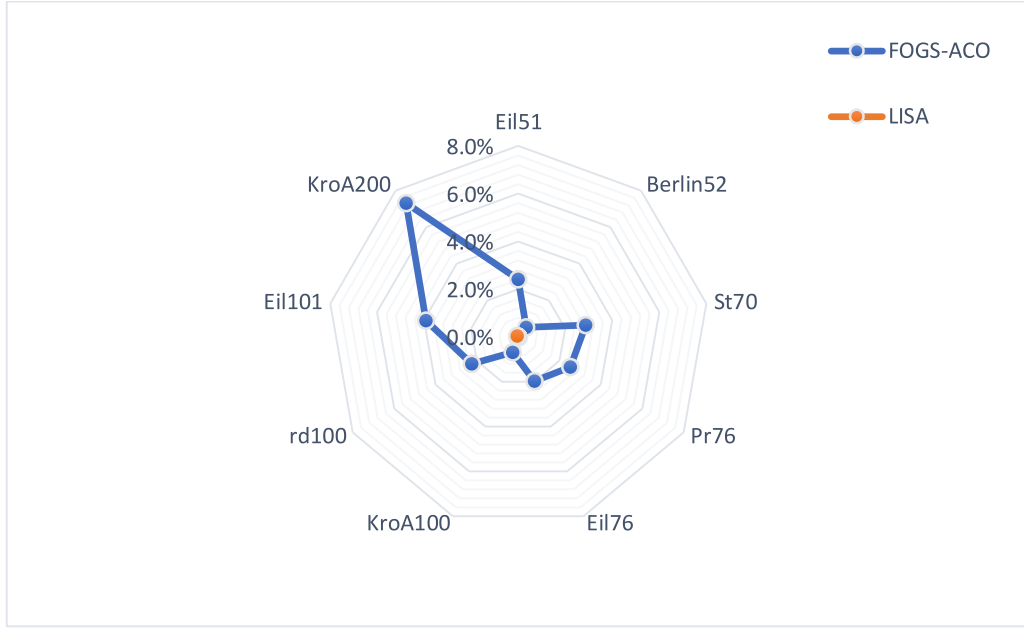


Figure 3.8 Percent Deviation Averages of LISA and FOGS-ACO

For the objectivity sake in the interpretation of the results above, we can say that LISA is mostly suitable for solving small and medium-scale instances. Also, for solving large-scale instances, LISA presents competitive results in comparison with the other methods used in this study. However, there exist more accurate optimizers for large-scale TSP instances.

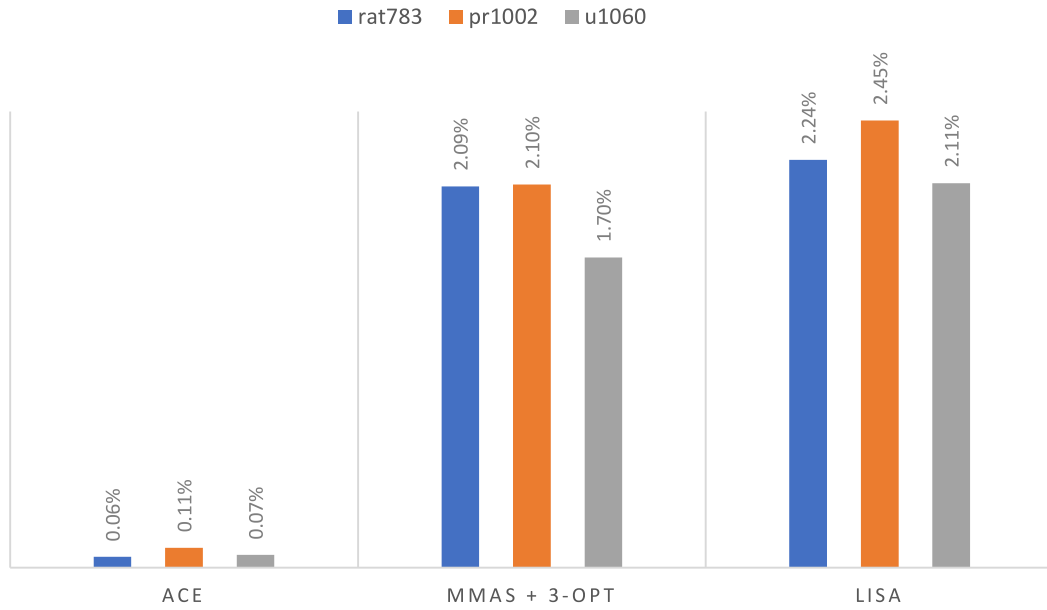


Figure 3.9 Average percent deviation of LISA, Extended ACO (ACE) and MMAS+3-Opt

Figure 3.9 shows the performance of LISA with an Extended ACO (ACE) and MMAS using a 3-Opt local search [16]. Although LISA uses only a 2-Opt based local search, which is smaller in comparison to other neighborhood structures such as 3-Opt or the neighborhood, LISA performance is comparable to MMAS with 3-Opt. Moreover, Table 3.5 presents a comparison of LISA with HBMO [88] and DGSA [89]. In their references, both methods are ranked in the top 10 among 50 TSP optimizers in the 8th DIMACS Implementation Challenge [90]. The performance of LISA is similar to both methods in small instances, but the comparison in terms of large instances shows that HBMO and DGSA are more efficient.

Table 3.5 Comparison of Computational results of LISA, DGSA and HBMO

Instance	Optimal fitness	LISA	DGSA	HBMO
rd400	15281	0.89%	0.12%	0.00%
pr439	107217	0.51%	0.13%	0.00%
pcb442	50778	0.83%	0.21%	0.00%
d493	35002	1.23%	0.31%	0.00%
rat575	6773	2.14%	0.49%	0.00%
p654	34643	0.19%	0.18%	0.00%
rat783	8806	2.24%	0.67%	0.00%
pr1002	259045	2.45%	0.82%	0.00%
fl1400	20127	0.95%	0.68%	0.01%
fl1577	22249	1.16%	0.21%	0.02%
d2103	80450	1.81%	1.16%	0.04%
pr2392	378032	4.70%	0.91%	0.03%
	Total Average	1.59%	0.49%	0.01%

Table 3.4 Comparison Table of LISA with six meta-heuristics.

Instance s	BKS	LISA			Simulated Annealing [86] (2011)			Discrete Cuckoo Search [75] (2014)			DIWO [78] (2015)			Chen & Chien [79] (2011)			Discrete Bat Algorithm [77] (2016)			WFA + 3opt [74] (2013)		
		best	Av (%)	SD	best	Av (%)	SD	best	Av (%)	SD	Best	Av (%)	SD	best	Av (%)	SD	best	Av(%)	SD	best	Av(%)	SD
eil51	426	426	0.00	0.00	428,872	0.67		426	0.00	0.00	428,9	0.70	0.35	427	0.30		426	0.49	1.6	426	0.14	0.52
berlin52	7542	7542	0.00	0.00	7544.37	0.03		7542	0.00	0.00	7544.3	0.03	0.00	7542	0.00		7542	0.00	0	7542	0.00	0.00
st70	675	675	0.00	0.00	677.11	0.31		675	0.00	0.00	677.1	0.31	0.00	-	-		675	0.61	2.8	-	-	-
eil76	538	538	0.00	0.00	544.369	1.18		538	0.00	0.17	-	-	-	538	0.41		539	1.88	3.8	538	0.27	1.51
pr76	108159	108159	0.00	0.00	108159	0.00		108159	0.00	0.00	-	-	-	-	-		-	-	-	-	-	-
kroA100	21282	21282	0.00	0.00	21285.4	0.01		21282	0.00	0.00	21286.3	0.04	0.02	21282	0.42		21282	0.77	116.5	-	-	-
kroB100	22141	22141	0.00	0.00	22139.1	-		22141	0.00	0.00	22284.9	0.88	0.19	22141	0.64		22140	1.65	221.3	-	-	-
kroC100	20749	20749	0.00	0.00	20750.8	0.00		20749	0.00	0.00	-	-	-	20749	0.63		20749	1.45	164.7	-	-	-
kroD100	21294	21294	0.00	0.00	21294.3	0.03		21294	0.04	0.00	-	-	-	21309	1.53		21294	1.41	141.6	-	-	-
kroE100	22068	22068	0.00	0.00	22106.3	0.20		22068	0.06	0.00	-	-	-	22068	0.52		22068	1.28	169.6	-	-	-
eil101	629	629	0.02	0.34	640.212	1.83		629	0.22	1.14	-	-	-	630	0.99		634	2.77	4.9	629	0.72	3.47
lin105	14379	14379	0.00	0.00	14383	0.02		14379	0.00	0.00	-	-	-	14379	0.19		-	-	-	14379	0.56	1.38
pr107	44303	44303	0.00	0.00	44301.7	-		44303	0.00	12.90	44440.3	0.48	0.16	-	-		44303	1.11	232.4	-	-	-
pr124	59030	59030	0.00	0.00	59030.7	0.00		5903	0.00	0.00	-	-	-	-	-		59030	0.65	265.9	-	-	-
bier127	118282	118282	0.00	5.56	118294	0.05		118282	0.06	12.73	-	-	-	118282	0.96		-	-	-	118282	0.09	131.12
ch130	6110	6110	0.00	0.54	6110.72	0.18		6110	0.42	21.24	-	-	-	6141	1.57		-	-	-	6115	0.06	6.85
pr136	96772	96798	0.22	82.12	96966.3	0.31		96790	0.24	134.43	97004.3	0.94	0.50	-	-		97547	2.67	707.2	-	-	-
pr144	58537	58537	0.00	0.00	58535.2	0.01		58537	0.00	0.00	-	-	-	-	-		58537	0.58	295.6	-	-	-
ch150	6528	6528	0.13	8.77	6530.9	0.16		6528	0.33	20.51	-	-	-	6528	0.55		-	-	-	6539	2.64	60.82

kroA150	26524	26524	0.00	4.66	26524.9	0.05	26524	0.17	56.26	26720.3	0.78	0.05	26524	1.41	-	-	-	-
kroB150	26130	26130	0.00	2.60	26140.7	0.18	26130	0.11	34.72	26203.2	0.32	0.07	26130	1.22	-	-	-	-
pr152	73682	73682	0.00	0.00	73683.6	0.01	73682	0.00	0.00	-	-	-	-	-	73921	1.35	426.5	-
rat195	2323	2323	0.31	3.45	2345.22	1.07	2324	0.81	8.49	-	-	-	-	-	-	-	-	-
d198	15780	15786	0.22	10.95	15830.6	0.41	15781	0.17	17.02	15847.9	0.67	0.34	-	-	-	-	-	-
kroA200	29368	29368	0.06	46.28	29411.5	0.23	29382	0.26	95.68	-	-	-	29383	1.26	-	-	-	-
pr226	80369	80369	0.04	35.43	80542.1	0.39	80369	0.02	60.31	80377.0	0.22	0.19	-	-	-	-	-	-
gil262	2378	2379	0.23	2.63	2393.64	0.86	2382	0.68	9.56	-	-	-	-	-	-	-	-	-
pr264	49135	49135	0.00	0.00	49135	0.00	49135	0.24	159.98	-	-	-	-	-	49756	3.61	887	-
a280	2579	2579	0.14	8.11	-	-	2579	0.51	11.86	2590.1	0.77	0.70	-	-	-	-	-	-
pr299	48191	48201	0.18	61.73	48269.2	0.28	48207	0.58	131.79	-	-	-	-	-	48310	3.08	1200.1	-
lin318	42029	42140	0.50	71.34	42306.7	0.84	42125	0.96	185.43	-	-	-	42487	2.32	-	-	42210	0.57
rd400	15281	15352	0.90	33.28	15350.7	0.97	15447	1.65	60.56	15543.8	2.42	0.86	-	-	-	-	-	-
fl417	11861	11866	0.14	4.47	11940.4	1.54	11873	0.41	20.45	-	-	-	-	-	-	-	-	-
pr439	107217	107453	0.51	284.84	110020	2.80	107447	0.69	438.15	-	-	-	-	-	115153	7.49	3825.8	-
pcb442	50778	51003	0.83	137.98	51063.9	0.96	-	-	-	51595.5	2.17	0.80	-	-	-	-	-	-
rat575	6773	6892	2.15	15.11	6872.11	1.94	6896	2.71	35.74	-	-	-	6891	2.38	-	-	-	4.34
rat783	8806	8944	2.24	27.23	8954.36	2.00	9043	3.44	38.09	-	-	-	8988	3.10	-	-	-	6.16
pr1002	259045	262844	2.45	932.43	263512	2.01	266508	3.70	1126.86	266039.2	3.19	2.60	-	-	270016	5.93	3617.8	-

3.7 Conclusion

We proposed in this chapter, LISA, a new nature-inspired algorithm for unconstrained combinatorial optimization problems. The main concept of LISA is based on lightning observations and different lightning simulation models defined in the literature.

In LISA, atmospheric charged particles are considered as nodes being added, following the local potential and electric field intensity, to a downward or an upward charged leader based on their positions either near the cloud “NC” or near the ground “NG”. Once the downward leader meets up with the upward leader, a complete lightning path is formed to allow a cloud to ground discharge. The lightning path allowing the maximum discharge amount is considered as the optimal solution found. The downward leader and the upward leader construction spaces are then updated, simulating the electric current properties by exchanging positions of some NC particles with NG particles, and a new generation of upward and downward leaders is constructed.

To prove its performance, LISA is tested with benchmarking instances of the traveling salesman problem. The obtained results demonstrate a high efficiency on small and medium scale instances in comparison with other optimization techniques reported in this study. However, other optimizers such as DGSA and HBMO outperform LISA in large scale instances as they proceed their local search using bigger neighborhood structures such as 3-Opt for ACE, while DGSA combines 2-Opt in large scale instances with other techniques like the Fixed Radius Search, Candidate Lists and don’t look bits [75]. Also, HBMO uses three neighborhood structures notably 2-Opt, 2.5-Opt, and 3-Opt [76]. We believe that a significant performance is expected when proceeding with a similar approach to enhance the local search phase for LISA on the TSP large-scale instances. Such implementation is currently under investigation as a short-term perspective of this study.

Chapter IV

Adaptation and Application of the Fireworks Algorithm to
NP-Hard Scheduling Problems & Big Data Optimization

4.1 Fireworks Algorithm: Introduction and basic design

Fireworks Algorithm (FA) continues to record increasing growth and success in continuous optimization. Inspired by the physical behavior of an exploding firework, FA [100] showed interesting performance on continuous optimization. In fact, during a fireworks explosion, numerous sparks are generated around the explosion center. The process can be seen as a search process where the explosion's location represents a solution, while sparks represent an exploration of the local search space.

4.1.1 General design

Fireworks Algorithm, initially proposed for complex functions optimization, is a novel memetic technique inspired by fireworks explosions. Indeed, the explosion process can be considered as a search process in the local space around the explosion's location through the generated sparks. As shown in *Figure 4.1.1*, during each generation of an explosion, n locations (solutions) are selected in order to set n fireworks off. Once the explosion happens, the locations of sparks are collected and evaluated. Most of the generated sparks and some of the current fireworks locations are selected to be the locations of the next generation explosions, and so on, an iterative process runs until an optimal solution is found, or a stopping criterion is met.

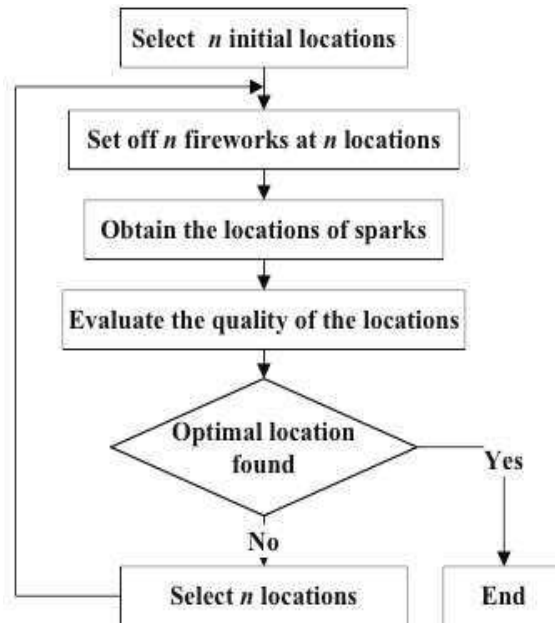


Figure 4.1 The general design of the Fireworks model

4.1.2 Sparks generation's control parameters

To correctly mimic the sparks generation process, many control parameters are defined such as the number of generated sparks for each firework, and the explosion amplitude. These parameters control the search process and the explorative behavior of the algorithm. The equation defining the number S_i of generated sparks in a given location of a firework x_i is:

$$S_i = m \times \frac{y_{max} - f(x_i) + \varepsilon}{\sum_{i=1}^n y_{max} - f(x_i) + \varepsilon} \quad (4.1)$$

where y_{max} is the worst value of the objective function, $y_{max} = \max f(x_i)$, ($i = 1, \dots, n$). m is a parameter controlling the total number of generated sparks for n fireworks. ε denotes the smallest computer constant to avoid zero division error.

S_i is then bounded to avoid the overwhelming effects of sparks, and then rounded as shown below:

$$S_i = \begin{cases} \text{round}(a \times m) & \text{if } S_i < a \times m \\ \text{round}(b \times m) & \text{if } S_i > b \times m \\ \text{round}(S_i) & \text{otherwise} \end{cases} \quad (4.2)$$

where a and b are constant parameters.

For the explosion amplitude for each firework, the equation is given by:

$$A_i = \hat{A} \times \frac{f(x_i) - y_{min} + \varepsilon}{\sum_{i=1}^n f(x_i) - y_{min} + \varepsilon} \quad (4.3)$$

where $\hat{A}$ is the maximum explosion amplitude and y_{min} represents the best value of the objective function $y_{min} = \min(f(x_i))$ ($i = 1, \dots, n$).

In the present work, these control parameters equations are preserved in the proposed combinatorial adaptation. As for sparks generation, the mechanism is slightly different. This latter is discussed in the next section.

4.1.3 Next-generation selection

After sparks generation, many new locations are obtained. Thus, a selection strategy must be defined to prepare for the next explosion. Among n current fireworks locations,

only the best one is selected for the next explosion. As for the newly generated sparks' locations, a stochastic mechanism is used for choosing $n-1$ locations for the next explosion generation. The probability p to choose a spark location x_i is defined by:

$$p(x_i) = \frac{R(x_i)}{\sum R(x_i)} \quad (4.4)$$

This probability is based on the distance between two locations x and y . For this proposed Fireworks Algorithm, the distance between two locations is evaluated as a subtraction of the objective values of the concerned locations. $R(x_i)$ is then defined as follows:

$$R(x_i) = \mathring{a} \sum_{j \in K} d(x_i, x_j) = \mathring{a} \sum_{j \in K} |f(x_i) - f(x_j)| \quad (4.5)$$

4.2 A Fireworks Algorithm for the total weighted tardiness problem

Many real-world optimization problems are naturally combinatorial problems. The most challenging among them are those classified as NP-Hard problems, like routing, manufacturing cell formation, frequency assignment, or scheduling problems. The main goal of the latter ones is the assignment of tasks (jobs) to single or multiple machines to minimize one or many objective functions.

To our best knowledge, the *FA* algorithm hasn't been yet widely used for combinatorial optimization, especially for scheduling problems. Thus, we introduce, in this section, a Fireworks Algorithm adaptation to unconstrained permutation-based scheduling problems. To test its performance, the proposed algorithm is applied to the single-machine total weighted tardiness problem.

4.2.1 SMTWT: Problem definition

In engineering and computer science, the single machine total weighted tardiness problem (*SMTWT*) [101] is a widely studied scheduling problem that is typed as *NP-Hard*. *SMTWT* considers n jobs (or tasks) to be processed sequentially on a single machine. Each job i is characterized by a processing time P_i , representing the necessary time needed by the machine to process the job i , a due date d_i indicating the date by which a job i should be completed, and a nonnegative weight w_i representing the relative importance of job i . The tardiness of job j is then defined as:

$$T_j = \max(C_j - d_j, 0) \quad \text{where} \quad C_j = \sum_{i=1}^j P_i \quad (4.6)$$

C_j denotes the completion time of a job j . All jobs are assumed to be available for

processing at time *zero*. The purpose of *SMTWT* is to find a job sequence minimizing the total weighted tardiness given by:

$$T = \sum_{i=1}^n w_i T_i \quad (4.7)$$

For more clarity, let us illustrate the problem in the following example. Consider the scheduling of aircrafts' takeoffs at an airport. Assuming that the airport has only one landing strip, the planes are waiting for taking off, and there are no time constraints between takeoffs. The processing machine can then model the airport runway, j models the task, P_j models the time for an aircraft to take off from the runway, d_j models the maximum time before which an aircraft should take off to be on time, and w_j models the penalty to be paid by the company if the flight arrives late at its destination. The goal would be to minimize the total delay induced by the sequencing of flights, thus, implicitly reduce the total paid penalties by airline companies.

Of course, many extended variants of *SMTWT* were developed such as Single machine scheduling with time-dependent linear deterioration and rate-modifying maintenance [102], and Two-agent single machine scheduling with forbidden intervals [103], and many others. This contribution focuses on the basic *SMTWT*.

Several algorithms have been used to tackle *SMTWT*. Exact algorithms such as branch and bound techniques [93], and methods like greedy heuristics [94] have been proposed, but they have encountered some issues in obtaining optimality, sometimes even in small-scale problems. In response to this, many optimization techniques notably nature-inspired techniques, have been proposed to tackle these issues such as Particle Swarm Optimization [95], Ant Colony Optimization [96], Genetic algorithms [97], Learning Automata [98], Evolutionary Algorithms [99] and many others.

4.2.2 A Discrete Fireworks Algorithm for SMTWT Problem

4.2.2.1 Solution encoding

A linear vector-based representation is used to represent solutions for an *SMTWT* problem of size N (Number of Jobs). Each integer in the vector represents the job number to be sequenced.

<i>1</i>	<i>2</i>	<i>3</i>	<i>...</i>	<i>N-2</i>	<i>N-1</i>	<i>N</i>
----------	----------	----------	------------	------------	------------	----------

Figure 4.2 Representation of a solution

- **Neighborhood structure**

For *SMTWT*, three neighborhood structures of a solution s are the most used in the literature.

- **Insertion neighborhood**

Given a solution s , insertion neighborhood is the resulting set of solutions wherein each solution; a job j in s is reinserted in a different position.

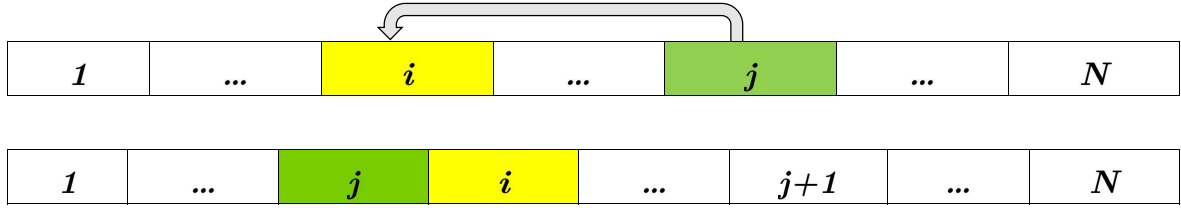


Figure 4.3 Insertion neighbor solution

- **Interchange neighborhood**

This structure represents the resulting set of solutions where each solution s is obtained by swapping a job i , and a job j .

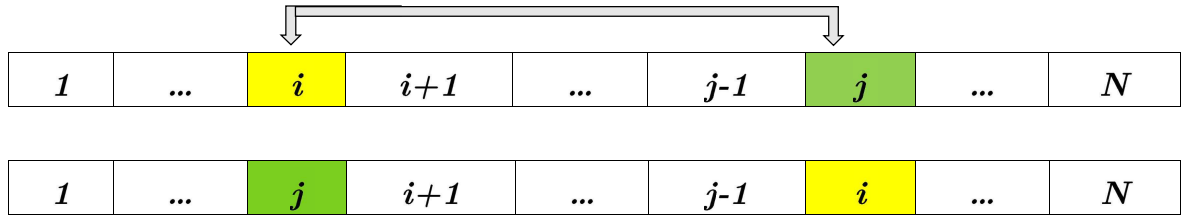


Figure 4.4 Interchange neighbor solution

- **Twist neighborhood**

The twist neighborhood of a solution s is the resulting set of solutions where given job i and job j , the whole sub-list of jobs in a sequence, starting from job i to job j , is reversed.

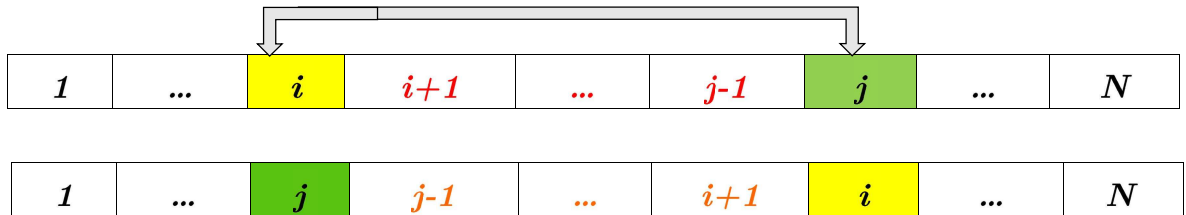


Figure 4.5 Twist neighbor solution

4.2.2.2 Sparks generation

Diversification and intensification phases are crucial for a good cover of the search space and may affect the algorithm performance negatively if they are not well designed. As for the original algorithm, two search stages are used. The first stage consists of an explorative phase, where input locations are widely explored using special operators as shown in *Figure 4.6*. By mimicking the fireworks explosion and using control parameters defined in equations (4.2) and (4.3), sparks' locations are obtained using *Algorithm 4.1* where d is the problem dimension representing the number of jobs to be sequenced.

Algorithm 4.1 *Exploration spark*

- Calculate the firework amplitude A_i using equation 4.1.3
 - For each spark do:
 - Initialize spark with the current location x_i
 - Select the sequence S with length A_i of the first elements of x_i
 - Generate the displacement h in $A_i < h < d$ where h is a random value
 - Re-insert S in x_i at the position h
 - End For
-

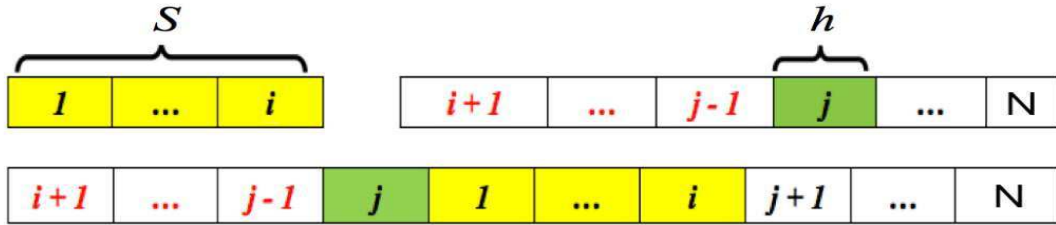


Figure 4.6 Exploration spark example

The second stage implements an exploitation search phase. Indeed, the sparks of the input fireworks locations are generated using a twist operator that builds neighbor solutions. The twist operation is rarely used in scheduling problems in comparison with the interchange and insertion operators, but when the explorative and exploitation phases are combined, the twist decent search seems to perform well, leading to good results that are demonstrated in the next section. *Algorithm 4.2* is used to generate sparks at this stage.

Algorithm 4.2 *Specific Exploitation spark*

- Initialize spark sp with the current firework location x_i
 - Calculate $z = \text{round}(d \times \text{rand}(0,1))$
 - While $z > 0$ do
 - Generate the best spark st using the twist descent
 - if $f(st) - f(sp) < 0$
 - $sp = st$
 - End If
 - $z = z-1$
 - End While
-

4.2.2.3 DFA pseudo-code

Algorithm 4.3 summarizes the complete process of the discrete Fireworks Algorithm for minimizing the total weighted tardiness.

Algorithm 4.3 *Discrete Fireworks Algorithm*

- Initialize randomly n fireworks locations
 - While stopping criteria is false do:
 - For each firework x_i do
 - Calculate the number of sparks using equation 4.2
 - Obtain locations of exploration sparks using Algorithm 4.1
 - End For
 - For each firework x_i do
 - Generate specific exploitation spark using Algorithm 4.2
 - End For
 - Select the best current fireworks location and keep it for the next explosion;
 - Select $n-1$ locations among all generated sparks locations for the next explosion using equation 4.4;
 - End While
-

4.2.3 Variable Neighborhood Search

Initially proposed by Mladenović and Hansen [103], the *Variable Neighborhood Search* “VNS” is used to extend the search process on multiple neighborhood structures to improve the current solution. In general, the Variable Neighborhood Search using

insertion and interchange operators has a high potential for the *SMTWT* problem. Thus, the proposed *DFA* is hybridized with a *VNS* decent to increase the performance.

4.2.4 Results and discussion

To evaluate the performance of the proposed method, a benchmark set of randomly generated instances, available via the ORLIB library, is used. The benchmark is classified into three classes: 40, 50, and 100 jobs. Each class contains 125 instances. The optimal solutions for instances with 40 and 50 jobs are known, while for 100 jobs only the best-known solutions are given. A comparison between the proposed *discrete Fireworks Algorithm* and *discrete PSO*, *Unified PSO* [104], *Ant Colony Optimization* [105], and a hybrid *Simulated Annealing* [106] is provided. The algorithm parameters are fixed in the *Table 4.1*.

Table 4.1 DFA initial parameter values

<i>Parameter</i>	<i>Value</i>
<i>Stop Criteria</i>	<i>50 iterations</i>
<i>a</i>	<i>0.2</i>
<i>b</i>	<i>0.8</i>
$\hat{A}$	<i>Instance Size $\times$ 0.8</i>
<i>m</i>	<i>Instance Size $\times$ 0.5</i>

Table 4.2 below shows *DFA* performance on finding optimal solutions without a *VNS* local search. For 40 jobs instances, the proposed algorithm was able to find 102 optimal solutions out of 125 in less than one second with a 2.3 Ghz processor, 50% are found in less than half a second, and more than 32% are found in less than 0.1 seconds. It is important to mention that the maximum CPU time allowed was 30 seconds per instance, and the presented results are from a random run and not the best run. In the *Table 4.3*, the maximum and minimum percent deviation of results obtained from instances where *DFA* could not find the optimal solutions shows that the largest deviation is less than 12% for both 40 and 50 jobs instances. Also, more than 78% of instances have a deviation percent less than 2% from optimums as illustrated in *Figure 4.6* and *Figure 4.7*. Furthermore, encouraging results are demonstrated in the comparison shown in *Table 4.4* where the proposed method outperforms *PSO* and *UPSO*. In *Table 4.2.4*, *DFA* shows good performance for the 40 jobs instances, but once

the problem dimension is augmented to 50 jobs, a performance decrease is noticed, and the Hybrid Simulated Annealing appears to perform better.

Table 4.2 DFA results for reached optimums

<i>Algorithm</i>	<i>Jobs</i>	<i>Optimal solutions found</i>	<i>Mean Time (seconds)</i>
<i>DFA</i>	<i>40</i>	<i>102</i>	<i>0.83</i>
	<i>50</i>	<i>67</i>	<i>2.72</i>

Table 4.3 DFA results for unreached optimums

	<i>40 Jobs</i>	<i>50 Jobs</i>
<i>Instance size</i>	<i>125</i>	<i>125</i>
<i>Unreached optimums (%)</i>	<i>18.4</i>	<i>46.4</i>
<i>Largest percent deviation (%)</i>	<i>10.85</i>	<i>11.8</i>
<i>Smallest percent deviation (%)</i>	<i>0.01</i>	<i>0.005</i>
<i>Average percent deviation (%)</i>	<i>1.65</i>	<i>1.52</i>

Table 4.4 Comparison results with some Particle Swarm Optimization variants

<i>Jobs</i>	<i>Algorithm</i>	<i>Optimums found (%)</i>	<i>Population Size</i>
<i>40</i>	<i>DFA</i>	<i>81.6%</i>	<i>10</i>
	<i>PSO</i>	<i>56.3%</i>	<i>400</i>
	<i>UPSO</i>	<i>65.8%</i>	<i>400</i>
<i>50</i>	<i>DFA</i>	<i>53.6%</i>	<i>10</i>
	<i>PSO</i>	<i>42.2%</i>	<i>500</i>
	<i>UPSO</i>	<i>43.7%</i>	<i>500</i>

Although the results are very promising, the real challenge would be to find all the best-known solutions in all instances. That's why a hybridized version of the proposed method with a *variable neighborhood descent* is implemented by replacing the twist decent in *Algorithm 4.2* with a *VNS* decent search. The results show that the proposed method can find all the best-known solutions on all the 40 and 50 jobs' instances in a mean time less than one and four seconds respectively in all runs. Thus, the association

of the exploitation capacity of the VNS search with the explorative technique of the proposed method is successful.

Table 4.5 Comparison results with ant colony and simulated annealing

<i>Jobs</i>	<i>Algorithm</i>	<i>Reached optimums</i>
<i>40</i>	<i>DFA</i>	<i>102</i>
	<i>Hybrid SA</i>	<i>91</i>
	<i>ACO-AVG</i>	<i>57</i>
<i>50</i>	<i>DFA</i>	<i>67</i>
	<i>Hybrid SA</i>	<i>73</i>
	<i>ACO-AVG</i>	<i>50</i>

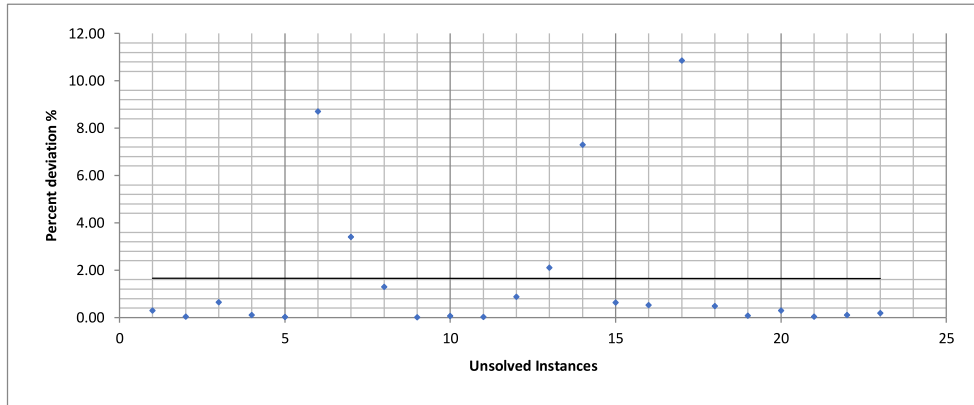


Figure 4.6 Percent deviation of unreachable optimums on 40 jobs instances

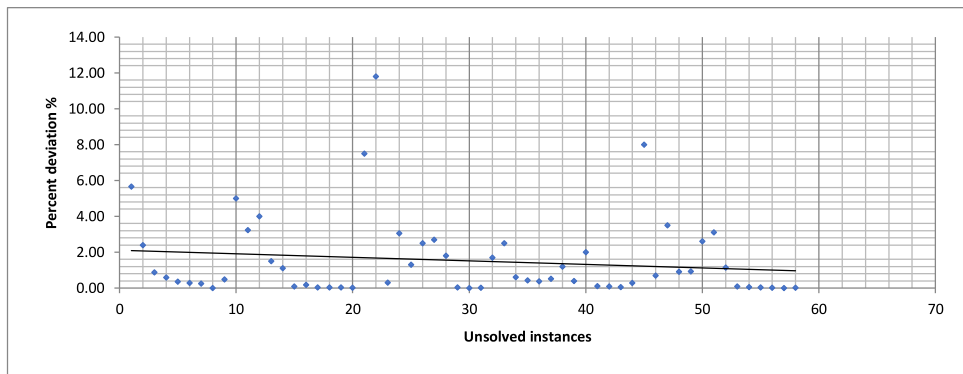


Figure 4.7 Percent deviation of unreachable optimums on 50 jobs instances

4.2.5 Conclusion and perspectives

In this study, *DFA*, a novel adaptation of the Fireworks Algorithm, to the single machine total weighted tardiness problem is proposed. The mimicking of fireworks explosion through sparks generation is carefully respected as it is in the original Fireworks Algorithm design. Also, Adequate operators are used for the *SMTWT* problem to navigate through the search space using two types of sparks: Exploration sparks and Exploitation sparks.

To prove its performance, *DFA* is tested with benchmark instances of the single machine total weighted tardiness problem. The obtained results demonstrate high efficiency in comparison with other optimization techniques used to solve *SMTWT*. Nevertheless, using a *VNS* search to find all the best-known solutions can be seen as a performance issue concerning the exploitation stage due to the high complexity. Therefore, a fine-tuning mechanism is under investigation to ameliorate the proposed method for a future extension to the single machine total weighted tardiness with setup times.

4.3 A Fireworks Algorithm for the Big Data Optimization problems

4.3.1 In the age of Big Data

During the last decade, accessing and sharing information became so easy which made the information flow grow significantly. This growth has been pushed further by the introduction of connected objects. The data can be generated massively from medical sensors to traffic data, social media, and many others. In 2014, data scientists revealed that we create every two days as much data as we did from the beginning of time until 2003 [107]. Indeed, many fields are more and more connected to the information flow or share data in social media. Thus, many challenging issues have been raised with traditional storage databases and processing techniques [108], because they couldn't support the big size and the high heterogeneity of collected data which we call today "Big Data". The National Institute of Standards and Technology (NIST) states that "Big Data consists of extensive datasets; primarily in the characteristics of volume, variety, velocity, and/or variability (3v); that require a scalable architecture for efficient storage, manipulation, and analysis" [109]. Certainly, Big Data brought many challenges concerning scalable architectures capable of handling and analyzing the huge amount of data. More specifically, these architectures should rely on efficient optimization algorithms to perform rapid intelligent processing and data analysis of highly voluminous data.

Furthermore, Big Data can be found everywhere in the industry. Indeed, several real-world applications dealing with optimizing and analyzing Big Data are emerging in different domains such as Cyber Security, logistics, healthcare, and many others. In this contribution, the electroencephalography cleaning process [110] is considered. Indeed, the brain's electrical activities are captured by the mean of EEG signals. One of the main challenges lies in determining true EEG brain signals in the captured data, hence, the interpretation accuracy becomes more reliable. Another critical challenge is the processing time as this cleaning process requires real-time performance. In the Optimization of Big Data 2015 competition [111], the problem is mathematically modeled as a Big Data optimization problem (BigOpt). The dimensionality of this problem can be very high as the number of handled variables can reach the product of the length of EEG signals by the number of channels used in the capture operation. Besides dealing with such high dimensionality, the optimization solving algorithm should also have a rapid convergence time as EEG signals require real-time processing. Also, BigOpt can be seen as a multi-objective optimization problem, hence, the algorithm should also reveal a well distributed Pareto front with many solutions. Consequently, a “Fireworks Algorithm” Framework (FAF) is developed, proposing a single objective Fireworks Algorithm (SOFWA) with a modified search mechanism for a consistent adaptation on Big Data problems, notably Big Optimization of Signals (BigOpt). SOFWA is then extended to handle multi-objective BigOpt. Thus, a new Multi-Objective Fireworks Algorithm (MOFWA) is obtained. To test the framework performance, FAF is used to optimize six datasets of the single and multi-objective BigOpt problem formulations. The proposed dimension of the datasets is ranging in 1024, 3072 and 4864. The datasets have a size of 20 KB in its binary form, and 0.5 MB in text form. The goal of the considered Big Optimization problem is to optimize and process large-scale data every second. The datasets will increase in size to reach 12MB as explained in the competition website [111]. Additionally, the data is bounded in the range $[-8, 8]$. However, no time constraints have been imposed on the current datasets. In the rest of this chapter, we provide a detailed description of the BigOpt problem and related works. Right after, the details of the proposed framework for Big Optimization problems and the experimental results are reported.

4.3.2 BigOpt Problem: Background & related works

Most Noninvasive Brain-Computer Interfaces (BCI) measure brain activity using special sensors placed on the head to record Electroencephalogram (EEG) signals that

allow neurobiologists to monitor the brain's electric activity [112]. Recent advances focus on analyzing and correctly interpreting the EEG signals. On one hand, they deliver essential information used by specialists to understand several interesting neurobiological disorders of great importance. On the other hand, EEG signals can be interpreted and translated to computer tasks which open wide doors to integrate BCIs into potential applications in the public markets, notably in the gaming industry, Virtual Reality, or military programs. However, achieving such tasks is faced with several difficulties. A BCI is characterized to be highly time-dependent and captured signals may vary on a milliseconds scale. Also, using many sensors implicate different accuracy rates because their manufacture is not the same. Also, captured signals are noised by non-brain electrical sources which affect further the accuracy and increase the processing time. These difficulties can define the 3v characterizing the Big Data concept, which leads to considering the EEG signals as an instance of Big Data.

Although several methods to monitor and record brain activity have been proposed, the EEG signal is still the most used because of its convenience to the patients and its economic cost [114]. The challenge in EEG signals analysis lies in successfully recovering true brain signals of a given activity that is noised by some artifacts due to both internal and external factors. While external factors can be summarized in non-brain electrical activity sources, the internal factors concern involuntary movements such as eyes blinking or undesirable imagined movements that generate an undesirable electrical brain activity. To remove these artifacts, some techniques are used such as the Independent Component Analysis method [115]. The reconstruction is supposed to be performed in real-time during the sensors' recording. Considering the high dimensionality of the signals, the reconstruction becomes a time-consuming task. To automatize the artifact cleaning process, the problem is mathematically expressed as a Big Data optimization problem.

Let X a matrix of dimension $N \times M$, where N is the number of inter-dependent time series (signals), and M is their respective length. Assume S , an $N \times M$ matrix with N independent time series of length M such that:

$$X = A . S \quad (4.8)$$

where A is a linear transformation matrix of $N \times N$ dimension. The problem is to decompose S into two matrices S_1 and S_2 of the same dimensionality of S such that:

$$S = S_1 + S_2 \quad (4.9)$$

$$X = A . S_1 + A . S_2 \quad (4.10)$$

The Pearson correlation matrix C between S_I and X is given by:

$$C = \frac{cov(X, A.S_1)}{var(X).var(A.S_1)} \quad (4.11)$$

where $cov(.)$ is the covariance matrix and $var(.)$ is the variance.

The goal is to find S_I similar to S where the distance between the two matrices is as minimal as possible. Also, the off-diagonal elements of C should be minimized while the on-diagonal elements have to be maximized.

This mathematical expression gave two formulations of the problem. The first one is a multi-objective optimization problem that minimizes two functions f_1 and f_2 such that: Given S , X and A , find S_1 which optimizes:

$$\min f_1 = \frac{1}{M.N} \sum_{i,j} (S_{ij} - S_{1,ij})^2 \quad (4.12)$$

$$\min f_2 = \frac{1}{N^2-N} \sum_{i,j \neq i} (C_{ij}^2) + \frac{1}{N} \sum_i (1 - C_{ii})^2 \quad (4.13)$$

The second is a single-objective optimization problem that optimizes:

$$f = \min (f_1 + f_2) \quad (4.14)$$

To solve BigOpt, many techniques have been used. For instance, an adaptive configuration of differential evolution algorithms (ACDE) was introduced in [116]. The idea proposed the run of three variants of DEA in parallel, then the best performing variant is automatically selected by using a selection strategy, to evolve the population until the evolutionary process reaches its end. To increase the exploitation capabilities of ACDE, the interior point approach is used as a local search method. This approach iteratively helps the algorithm search by generating one or more interesting search directions. The same work was ameliorated and extended to a differential evolution framework by Elsayed et al. [117]. They mainly focused on analyzing and tuning the differential evolution parameters of the used variants. The interior points method is applied only on the best-performing variant. They compared their results with each performing variant alone, notably DECC-DG [118], JADE [119] and SHADE [120]. DECC-DG is a differential evolution technique using an automatic decomposition strategy to uncover the underlying interaction structure of the decision variables. JADE is an adaptive DEA with an external archive and SHADE is an enhanced version of JADE. Goh et al. [121] investigated the performance of evolutionary algorithms such as MOEA/D and NSGA II to solve BigOpt. They showed that the standard

implementation of existing algorithms is extremely slow. Zhang et al. [122] proposed a multi-agent genetic algorithm with a redesigned competition and self-learning operators to solve the single objective BigOpt problem. The new operators are combined with crossover and mutation operators to simulate the learning behavior of agents. The study showed that the new self-learning operators can find quickly decreasing directions using a heuristic strategy. Zhang et al. [123] also proposed a decomposition-based multi-objective memetic algorithm with an embedded gradient local search for the multi-objective form of the BigOpt problem. The method exploits the gradient information and the decomposition method to design the local search operator.

4.3.3 Fireworks Algorithm Framework for BigOpt

A Fireworks Algorithm-based framework “FAF” is described. The framework proposes a Single Objective Fireworks Algorithm “SOFWA” and a Multi-Objective Fireworks Algorithm “MOFWA” for the Big Optimization problems. SOFWA is an adaptation of the basic Fireworks Algorithm to BigOpt where a modified search mechanism is adopted. MOFWA is a novel multi-objective algorithm based on the Fireworks search model. The framework initialization and the technical details of SOFWA and MOFWA are detailed in the next paragraphs.

4.3.3.1 Framework initialization

In FAF, a vector of decision variables is used as a solution representation scheme where each variable value is encoded by a real number. The size of each vector is the number of dimensions in the decision space. The goal of BigOpt is to find a matrix S_1 similar to S . The similarity is measured using the second norm distance d_2 .

$$d_2(S_1, S) = 0 \quad \Rightarrow \quad S_1 \equiv S \quad (4.15)$$

The initialization process is then facing two choices. Random initialization of solutions with uniformly distributed values in the decision variables ranges or initialize vectors using the values of S the optimal solution of the minimization problem of the first objective function. The experimental tests show convergence in both cases. However, the second choice has 15 times faster convergence than the random initialization.

To bring some diversity, the Fireworks are initialized with the values of S added to a vector ε_v of the same size. ε_v is initialized with random values smaller than a predefined constant ε . `Vector(rand(-1,1))` returns a vector of size equal to the number of the decision space dimensions initialized with random values in $[-1, 1]$ using `rand(.)`.

Algorithm 4.4 Fireworks Initialization

- Initialize the number of Fireworks: *popSize*
 - Initialize ε
 - Calculate $\varepsilon_v = \text{Vector}(\text{rand}(-1,1)) \times \varepsilon$
 - For $j : 1 \dots \text{popSize}$ do:
 - Create a Firework x_j
 - Initialize current location of the Firework x_j : $x_j = S + \varepsilon_v$
 - End For
-

4.3.3.2 SOFWA: A Single-Objective Fireworks Algorithm

- **Design**

SOFWA for BigOpt is mainly based on the Fireworks model reported in *Figure 4.1.1* where three phases are defined. Fireworks set-off, sparks generation and locations selection of the next generation of explosions.

- **Set-off**

SOFWA uses a modified mechanism for the fireworks set off. Contrarily to the original Fireworks Algorithm, SOFWA uses a fixed number of sparks S_i generated by each firework after the explosion which does not depend on the fitness function. This allows achieving two goals. The first one is to deliver explorative capacities to SOFWA such that solutions with poor quality can evolve equally with the ones having good quality. This can enlarge the search space coverage to discover new promising regions. As temporal complexity performances are critical in a Big Data context, the second goal aims to control the total number of evaluations performed by the algorithm.

Algorithm 4.5 Fireworks Set-Off

- Initialize $\hat{A}$
 - Initialize the number of sparks S_n to generate for all firework
 - For each Firework x_i do:
 - Calculate the amplitude A_i of Firework x_i using equation 4.1.3
 - End For
-

Indeed, the number of evaluations becomes constant which makes the algorithm performances concerning temporal complexity easily predictable. The amplitude

parameter calculation is similar to the one in the basic Fireworks Algorithm. The pseudo-code of the set-off phase is given in *Algorithm 4.5*.

- **Sparks generation**

SOFWA uses only one type of sparks. To obtain a new spark, a random integer is uniformly generated to represent the number of dimensions to modify in order to calculate the location of the spark. At this stage, both the exploration and the exploitation of the current space region is performed. Contrarily to the basic Fireworks Algorithm, the modified dimensions are not pre-selected in SOFWA, only the number of dimensions is known. That means a dimension can be modified many times depending on the random distribution used which gives an exploitation behavior if the number of dimensions is relatively small or an explorative behavior otherwise.

Algorithm 4.6 A Spark generation of a Firework

- Initialize a Spark location y with the current Firework location x
 - Calculate the displacement h_i using the current firework amplitude:

$$h_i = A_i \cdot \text{rand}(-1,1)$$
 - Generate uniformly a random k : $1 < k < d$ where d is the number of dimensions
 - For $j : 1 \dots k$ do:
 - Select a random dimension t
 - $y_t = x_t + h_i$
 - End For
-

- **Next-generation selection**

In SOFWA, the next generation of explosions are triggered from and nearby one of the best-found locations. This allows consistent exploitation of the current region and brings some balance to the search process. The parameter α is used to control the selection of the next location, where the Sparks are either penalized or promoted to evolve at the expense of the older Fireworks. The parameter ε_p is calculated during the framework initialization.

Algorithm 4.7 Selection of the next generation of Fireworks

- *Select the best Firework location F_{best}*
- *Select the best Spark location SP_{best}*
- *Initialize the next generation starting location*

$$ST_{next} = \begin{cases} F_{best} & \text{if } SP_{best} > \alpha \cdot F_{best} \\ SP_{best} & \text{Otherwise} \end{cases}$$

- *For each next-generation Firework x_i do:*
 - $x_i = ST_{next} + \varepsilon_v$
 - *End For*
-

- **Detailed Pseudo-code**

Algorithm 4.8 SOFWA Algorithm

- *Initialize the set of Fireworks using Algorithm 4.4*
 - *Initialize the best location Sol_{best}*
 - *While (Stop Criteria is not satisfied) do:*
 - *Set-off Fireworks using Algorithm 4.5*
 - *For each Firework x_i do:*
 - *For $j : 1 \dots k$ do:*
 - *Generate a Spark using Algorithm 4.6*
 - *End for*
 - *End for*
 - *Evaluate all Fireworks and generated Sparks*
 - *Update the best location Sol_{best}*
 - *Select the next generation of Fireworks locations using Algorithm 4.7*
 - *End while*
-

k is the fixed number of sparks of each firework, defined during the Set-Off.

4.3.3.3 MOFWA: Multi-Objective Fireworks Algorithm

- **Design**

MOFWA for BigOpt is an extension of SOFWA to cover multi-objective optimization. The search phases such as Fireworks set-off, sparks generation, and next-generation selection are performed as follows.

- **Set-off**

In MOFWA, each objective defines a fireworks type. For each firework type, a fixed number of post-explosion sparks is defined. Each Firework amplitude is calculated using the objective fitness function associated with the concerned Firework type.

Algorithm 4.9 Fireworks Set-Off

- Initialize $\hat{A}$
 - Initialize the number of sparks S'_{fi} to generate for each Fireworks type
 - For each Firework x_i do:
 - Calculate the amplitude A_i of Firework x_i using the current Fireworks type objective function in equation 4.3
 - End For
-

- **Sparks generation**

Algorithm 4.10 A Normal or Special Spark generation of a Firework

- Initialize a Spark location y with the current Firework location x
 - Calculate the displacement h_i using the current Firework amplitude:
$$h_i = \begin{cases} A_i \cdot \text{rand}(-1,1) & \text{if Normal Spark} \\ A_i \cdot \text{GaussianRand}(-1,1) & \text{if Special Spark} \end{cases}$$
 - Generate uniformly a random $k : 1 < k < d$
 - For $j : 1 \dots k$ do:
 - Select a random dimension t
 - $y_t = x_t + h_i$
 - End For
-

MOFWA uses the same SOFWA mechanism to generate sparks called normal sparks. Each objective Firework generates its sparks by modifying its location dimensions using a uniform distribution. To bring more diversity to the search as the basic Fireworks Algorithm suggests, another sparks generation process is defined using a Gaussian distribution. Sparks generated in this process are called special sparks. The sparks generation process is given in *Algorithm 4.10*.

- **Next-generation selection**

After the evaluation of the newly generated sparks, the Pareto set is updated by adding directly the new non-dominated solutions. The rest of dominated sparks are filed in a list and sorted decreasingly from the best to the worst fitness value of a given objective function. For this case of BigOpt, the list is sorted using the second objective f_2 fitness value. Afterward, depending on the defined size of the initial population, 25% of the new locations are uniformly selected from the current Pareto set, 25% are selected from the first half of the list, and 50% from the last half. This selection strategy ensures diversity preservation during the next stages of the search. the selection strategy pseudo-code is given in *Algorithm 4.11*.

Algorithm 4.11 Selection of the next generation of Fireworks

- Initialize Table C with all Sparks and Fireworks locations
 - Sort C using f_2 fitness function values
 - Construct the new generation of Fireworks locations:
 - Randomly select 25% from the Pareto Set
 - Select 25% from the first elements of C
 - Select 50% starting from the last element of C
 - End Construction
-

- **Detailed Pseudo-code**

Algorithm 4.12 MOFWA Algorithm

- *Initialize the set of Fireworks using Algorithm INIT*
 - *Initialize the Pareto Set PS*
 - *While (Stop Criteria is not satisfied) do:*
 - *Set F1 type Fireworks and F2 type Fireworks*
 - *Set-off Fireworks using Algorithm 4.3.1M*
 - *For each Firework x_i do:*
 - *For $j : 1 \dots S'_{f_i}$ do:*
 - *Generate a Normal Spark using Algorithm 4.3.2M*
 - *Generate a Special Spark using Algorithm 4.3.2M*
 - *End for*
 - *End for*
 - *Evaluate all Fireworks and generated Sparks*
 - *Update the Pareto Set SP*
 - *Select the next generation of Fireworks locations using Algorithm 4.3.3M*
 - *End while*
-

S'_{f_i} is the number of sparks. Each Firework type has a fixed number of sparks defined during the Set-Off.

4.3.4 Experimental results & discussion

To validate the proposed framework, six datasets of different sizes are used for both single and multi-objective problems. The datasets contain six different EEG records with several difficulty levels. Indeed, the datasets can be classified into three classes A, B, and C. Class A contains D4 and D4N datasets. They represent EEG records collected using six channels (electrodes), four of them record EEG signals while the other two simulate the artifacts with a time delay. The signals are sampled at 256 Hz. Similarly, class B (C) contains D12 and D12N (D19 and D19N). It represents data collected using 18 (25) sources of signals. 12 (19) sources are real EEG data, and the others are used as artifacts. More details on data generation are given in [124]. One dataset from each class is noised (Instance name ends with *N*). Table 4.6 summarizes the properties of the used datasets.

The experiments are executed on a laptop with a *2.3 GHz* CPU and *8 Go* of RAM. No parallel computing has been performed. To avoid the “random effect”, experiments are

repeated ten times for the multi-objective problem and thirty times for the single objective problem.

Table 4.6 BigOpt instance details

<i>Dataset instance</i>	<i>D4</i>	<i>D4N</i>	<i>D12</i>	<i>D12N</i>	<i>D19</i>	<i>D19N</i>
<i>N° of channels</i>	<i>4</i>	<i>4</i>	<i>12</i>	<i>12</i>	<i>19</i>	<i>19</i>
<i>Data length</i>	<i>256</i>	<i>256</i>	<i>256</i>	<i>256</i>	<i>256</i>	<i>256</i>
<i>Size</i>	<i>1024</i>	<i>1024</i>	<i>3072</i>	<i>3072</i>	<i>4864</i>	<i>4864</i>
<i>Noise</i>	<i>No</i>	<i>Yes</i>	<i>No</i>	<i>Yes</i>	<i>No</i>	<i>Yes</i>

For the single-objective problem, the minimum fitness value obtained by SOFWA is considered as a measure of quality. Afterward, SOFWA is compared with the most recent approaches for BigOpt that are mostly based on differential evolution. For the multi-objective problem, the hyper-volume indicator [125] and a distance score from baseline [111] are selected as quality metrics. The comparison is done with the recently proposed approaches found in the literature. To present a coherent comparison study, the hyper-volume indicator obtained by all reported methods is calculated using the same reference point (1.01, 22.34). After thorough experiments, the framework parameters are initialized as given in *Table 4.7*.

Table 4.7 Framework parameters initial values

<i>Parameters</i>	<i>SOFWA</i>	<i>MOFWA</i>
<i>Stop Criteria</i>	<i>10000 iterations</i>	<i>2000 iterations</i>
ϵ	1×10^{-5}	1×10^{-5}
α	<i>1</i>	-
<i>popSize</i>	<i>2</i>	<i>4</i>
S'_{f1}	-	<i>2</i>
S'_{f2}	-	<i>2</i>
S_n	<i>2</i>	<i>2</i>
$\hat{A}$	<i>0,02</i>	<i>0,02</i>

4.3.4.1 Results of SOFWA for BigOpt

For SOFWA, *Table 4.8* reports the obtained results. The scored best (*Best*) and averages (*Avg.*) of the found solution fitness are good, but the more important is the standard deviation (*Stdev*) that is in the order of 10^{-6} , which shows high stability of the algorithm convergence. To clearly illustrate the convergence stability, *Figure 4.8* represents the recorded convergence during a random run.

SOFWA is compared to the Adaptive Configuration of Differential Evolution (ACDE) [116], Automated DE Framework (ADEF) [117], A cooperative co-evolution DE with differential grouping algorithm (DECC-DG) [118], adaptive DE algorithm with external archive (JADE) [119] and its improved version Success-history based adaptive DE (SHADE) [120]. Results of the baseline algorithm based on NSGAII are also included. *Table 4.9* summarizes a comparison where the best-found fitness (*Best*), Average fitness (*Avg.*), fitness standard deviation (*Stdev*), and average running time (*Time (s)*) of all methods are reported. The comparison shows that SOFWA not only outperforms the DE techniques but also scores the best deviations regarding convergence stability. SOFWA obtains better solutions and averages in a faster time. For instance, ADEF and ACDE seem to get close averages to SOFWA in D4 and D4N datasets. However, SOFWA is four times faster than ACDE and 1.5 times faster than ADEF. Similarly, the same performance is shown in D12 and D12N datasets.

Table 4.8 Results of SOFWA in all datasets

<i>Instance</i>	<i>D4</i>	<i>D4N</i>	<i>D12</i>	<i>D12N</i>	<i>D19</i>	<i>D19N</i>
<i>Best</i>	<i>6,11E-02</i>	<i>5,90E-02</i>	<i>2,02E-03</i>	<i>1,91E-03</i>	<i>2,65E-03</i>	<i>2,70E-03</i>
<i>Avg.</i>	<i>6,11E-02</i>	<i>5,90E-02</i>	<i>2,03E-03</i>	<i>1,92E-03</i>	<i>2,65E-03</i>	<i>2,70E-03</i>
<i>Stdev</i>	<i>3,47E-06</i>	<i>3,58E-06</i>	<i>2,09E-06</i>	<i>2,31E-06</i>	<i>2,66E-06</i>	<i>2,37E-06</i>
<i>NFFES</i>	<i>4,00E+04</i>	<i>4,00E+04</i>	<i>4,00E+04</i>	<i>4,00E+04</i>	<i>4,00E+04</i>	<i>4,00E+04</i>
<i>Time (s)</i>	<i>5,88E+00</i>	<i>5,80E+00</i>	<i>2,71E+01</i>	<i>2,70E+01</i>	<i>6,01E+01</i>	<i>6,01E+01</i>

However, in D19 and D19N where the size of the datasets is very large, SOFWA records the best performance over all other methods with a large difference.

Table 4.9 Comparison of SOFWA with several Differential Evolution techniques $\mathcal{E}$ NSGAI.

<i>Instance</i>		<i>D4</i>	<i>D4N</i>	<i>D12</i>	<i>D12N</i>	<i>D19</i>	<i>D19N</i>
SOFWA	Best	6.11E-02	5.90E-02	2.02E-03	1.91E-03	2.65E-03	2.70E-03
	Avg.	6.11E-02	5.90E-02	2.03E-03	1.92E-03	2.65E-03	2.70E-03
	Stdev	3.47E-06	3.58E-06	2.09E-06	2.31E-06	2.66E-06	2.37E-06
	Time	5.88E+00	5.80E+00	2.71E+01	2.70E+01	6.01E+01	6.01E+01
ACDE	Best	6.13E-02	5.93E-02	3.61E-02	3.63E-02	6.28E-02	7.02E-02
	Avg.	6.14E-02	5.93E-02	5.18E-02	5.09E-02	2.15E-01	1.36E-01
	Stdev	3.01E-06	3.51E-06	9.60E-03	1.11E-02	7.49E-02	5.16E-02
	Time	2.50E+01	2.18E+01	4.26E+01	4.27E+01	6.53E+01	6.53E+01
ADEF	Best	6.13E-02	5.93E-02	2.22E-03	2.14E-03	6.21E-03	6.54E-03
	Avg.	6.13E-02	5.93E-02	2.39E-03	2.28E-03	7.61E-03	7.88E-03
	Stdev	2.97E-06	1.87E-06	8.73E-05	8.25E-05	7.48E-04	7.45E-04
	Time	8.75E+00	8.33E+00	3.23E+01	3.19E+01	6.55E+01	6.58E+01
DECC-DG	Best	3.47E+00	8.24E+00	6.96E+00	7.15E+00	2.66E+02	2.69E+02
	Avg.	3.82E+00	8.89E+00	7.16E+00	7.39E+00	2.70E+02	2.73E+02
	Stdev	2.26E-01	3.18E-01	9.86E-02	8.10E-02	2.05E+00	2.16E+00
JADE	Best	9.58E-01	9.53E-01	2.04E+00	2.15E+00	2.49E+00	2.46E+00
	Avg.	1.10E+00	1.11E+00	2.32E+00	2.32E+00	2.73E+00	2.66E+00
	Stdev	8.37E-02	7.71E-02	1.10E-01	1.24E-01	1.11E-01	1.02E-01
SHADE	Best	7.71E-01	7.98E-01	1.81E+00	1.85E+00	2.29E+00	2.29E+00
	Avg.	8.97E-01	9.07E-01	2.09E+00	2.08E+00	2.45E+00	2.45E+00
	Stdev	6.51E-02	5.04E-02	9.39E-02	8.81E-02	9.52E-02	9.33E-02
NSGAI	Avg.	1.87E+00	1.74E+00	2.93E+00	2.82E+00	3.19E+00	3.17E+00

For a clear visualization of the reported results, *Figure 4.9* illustrates the obtained averages of SOFWA, ACDE, and ADEF. Concerning the computational time, although SOFWA was run on a less performant computer than the other methods, the obtained computational time is very satisfying as illustrated in *Figure 4.10*.

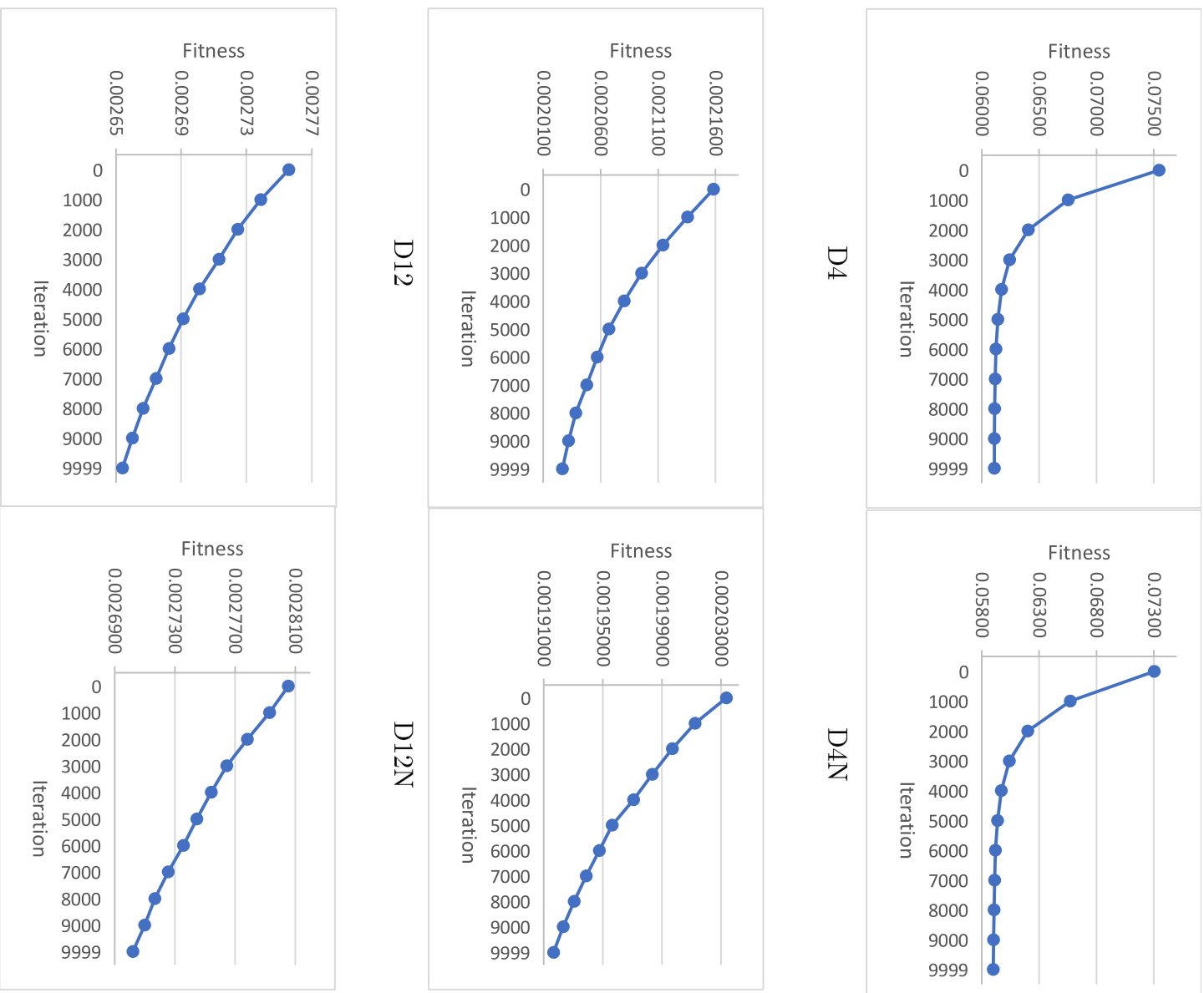


Figure 4.8 SOFWA convergence in a single run for all datasets

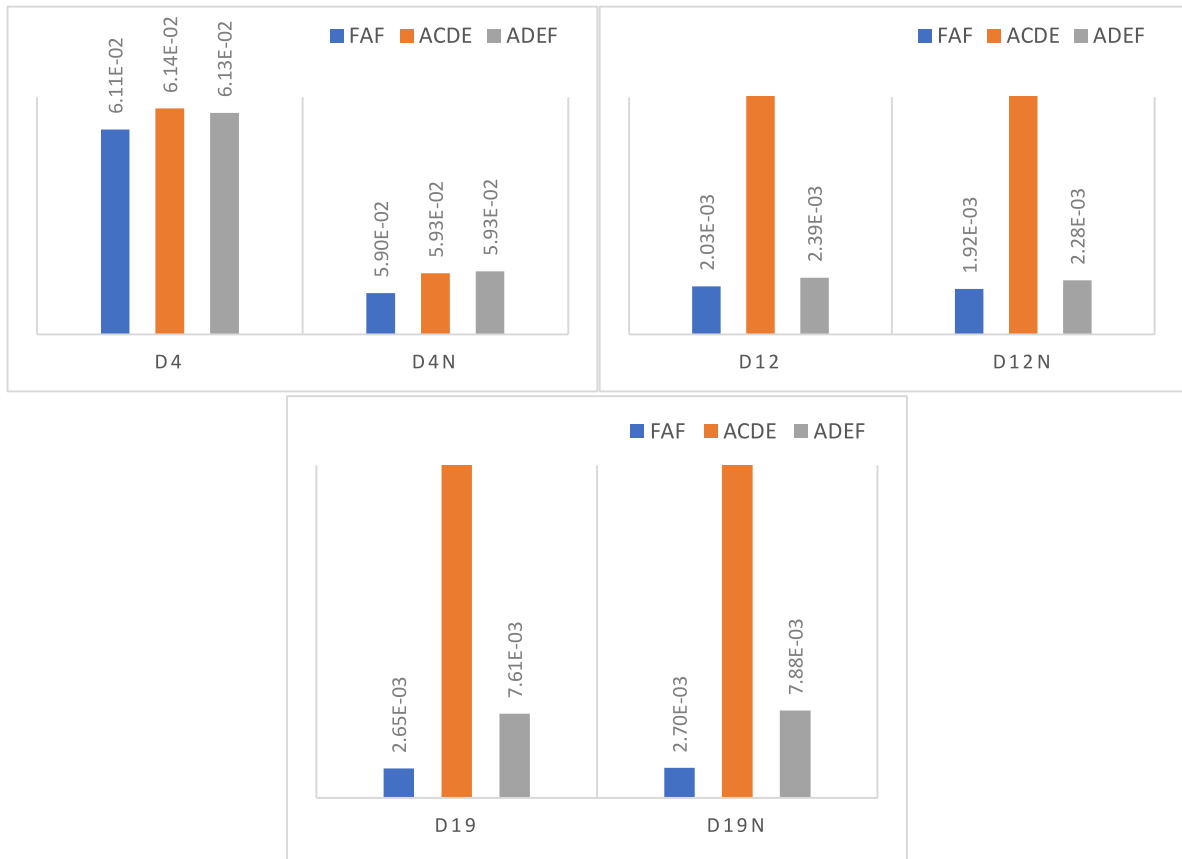


Figure 4.9 Comparison of the average quality between FAF, ACDE, and ADEF

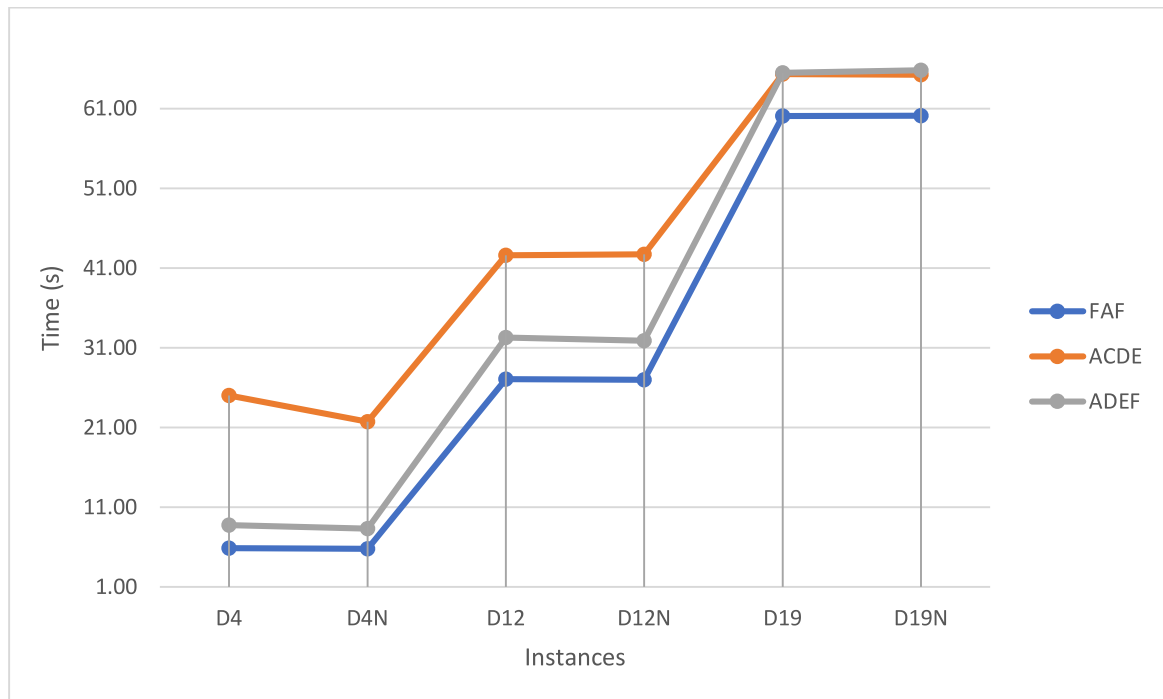


Figure 4.10 Comparison of the computational time between SOFWA, ACDE, and ADEF

4.3.4.2 Results of MOFWA for BigOpt

For the multi-objective problem, MOFWA obtains well-distributed Pareto fronts representing a large number of feasible solutions, as shown in *Figure 4.11*. The average scores (*Avg. Score*) and hyper-volumes (*Avg. HV*) are very encouraging. Also, the standard deviations (*Stdev. HV* and *Stdev. Score*) obtained of both metrics are low which means acceptable convergence stability. The number of fitness function evaluations (*NFFES*) on each dataset is also measured. The results are summarized in *Table 4.10*.

Table 4.10 Computational results of MOFWA in BigOpt datasets

<i>Instance</i>	<i>D4</i>	<i>D4N</i>	<i>D12</i>	<i>D12N</i>	<i>D19</i>	<i>D19N</i>
<i>Best HV</i>	2.253E+01	2.253E+01	2.256E+01	2.256E+01	2.256E+01	2.256E+01
<i>Avg. HV</i>	2.253E+01	2.253E+01	2.256E+01	2.256E+01	2.256E+01	2.256E+01
<i>Stdev. HV</i>	1.494E-03	1.657E-03	9.000E-05	6.000E-05	3.000E-05	6.403E-05
<i>Best Score</i>	2.244E+01	2.179E+01	2.312E+01	2.374E+01	2.337E+01	2.198E+01
<i>Avg. Score</i>	2.241E+01	2.177E+01	2.312E+01	2.374E+01	2.336E+01	2.198E+01
<i>Stdev. Score</i>	1.354E-02	1.280E-02	4.188E-03	3.080E-03	2.490E-03	2.855E-03
<i>Avg. Time (s)</i>	1.107E+01	1.106E+01	4.714E+01	4.730E+01	1.042E+02	1.037E+02
<i>NFFES</i>	6.400E+04	6.400E+04	6.400E+04	6.400E+04	6.400E+04	6.400E+04

Table 4.11 reports a comparison study of MOFWA with ACDE, ADEF, and MOEA/D with DE [117,126] where best score (Best), average score (Avg.), score standard deviation (Stdev), average time (Avg. Time), and the number of fitness function evaluations (NFFES) are given. Using the distance score metric, MOFWA outperforms ACDE and MOEA/D – DE. The results obtained by ADEF are also competitive where a comparable performance to MOFWA is noticed on datasets bigger than D4 and D4N, with a small difference in favor of MOFWA having a smaller standard deviation. As for D4 and D4N, ACDF outperforms all methods including MOFWA with a slight difference. Concerning the computational time, MOFWA is faster than ACDE. ADEF seems to be faster than MOFWA even when it performs fewer fitness function evaluations.

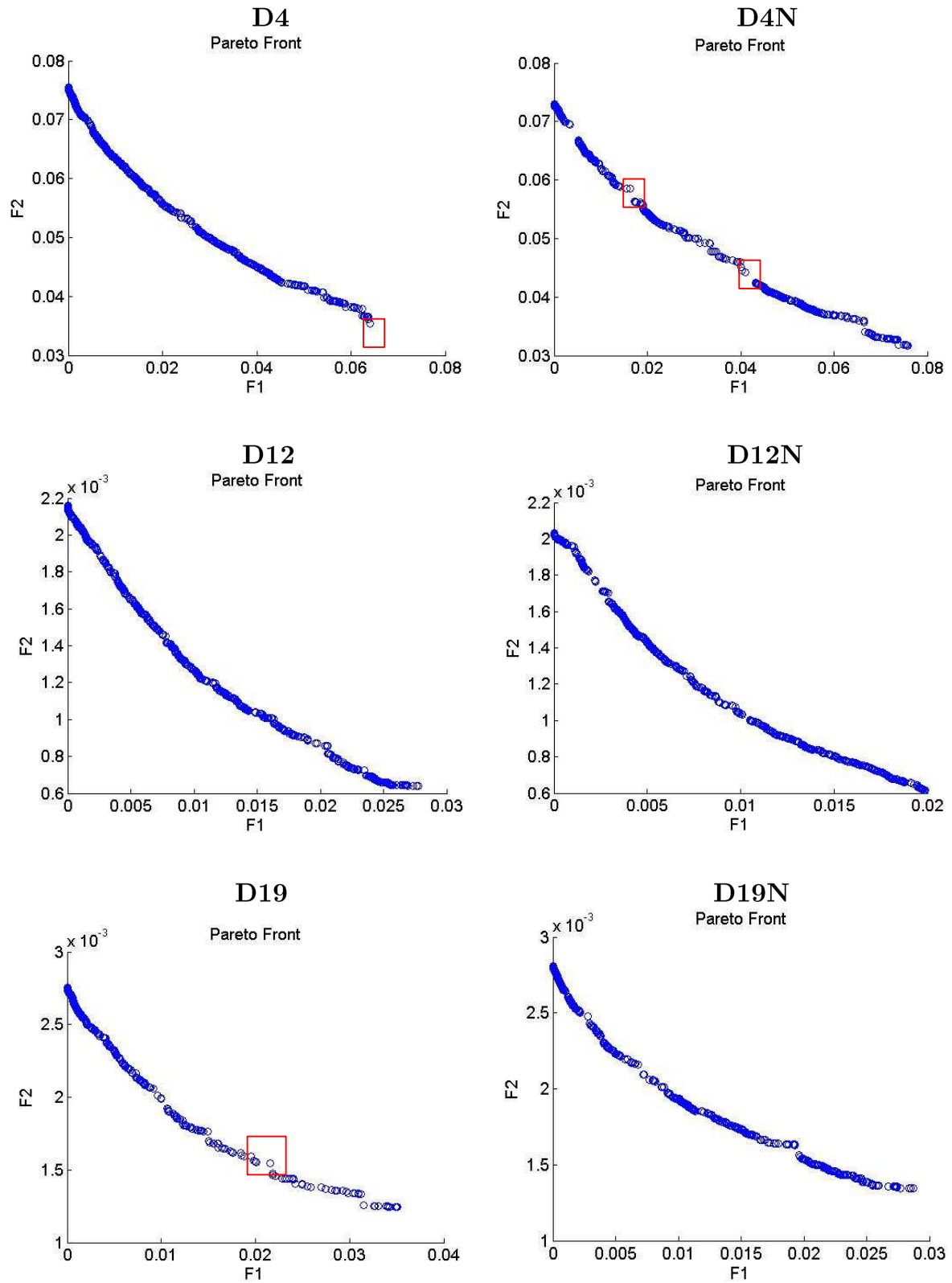


Figure 4.3.4. Pareto Fronts of MOFWA in all datasets.

Table 4.11 Comparison of results: MOFWA, ACDE, ADEF, and MOEA/D

<i>Instance</i>		<i>D4</i>	<i>D4N</i>	<i>D12</i>	<i>D12N</i>	<i>D19</i>	<i>D19N</i>
MOFWA	Best	2.244E+01	2.179E+01	2.312E+01	2.374E+01	2.337E+01	2.198E+01
	Avg.	2.241E+01	2.177E+01	2.312E+01	2.374E+01	2.336E+01	2.198E+01
	Stdev	1.354E-02	1.280E-02	4.188E-03	3.080E-03	2.490E-03	2.855E-03
	Time (s)	1.107E+01	1.106E+01	4.714E+01	4.730E+01	1.042E+02	1.037E+02
	NFFES	6.400E+04	6.400E+04	6.400E+04	6.400E+04	6.400E+04	6.400E+04
ACDE	Best	2.252E+01	2.187E+01	2.310E+01	2.372E+01	2.325E+01	2.191E+01
	Avg.	2.251E+01	2.186E+01	2.308E+01	2.370E+01	2.314E+01	2.186E+01
	Stdev	1.355E-02	1.191E-02	7.932E-03	7.189E-03	7.671E-02	3.108E-02
	Time(s)	3.196E+01	3.158E+01	6.403E+01	6.354E+01	1.002E+02	1.019E+02
	NFFES	1.00E+05	1.00E+05	1.00E+05	1.00E+05	1.00E+05	1.00E+05
ADEF	Best	2.25E+01	2.19E+01	2.31E+01	2.37E+01	2.34E+01	2.20E+01
	Avg.	2.25E+01	2.19E+01	2.31E+01	2.37E+01	2.34E+01	2.20E+01
	Stdev	5.84E-03	7.39E-03	1.13E-03	8.99E-04	2.76E-03	1.99E-03
	Time(s)	1.23E+01	1.25E+01	4.10E+01	4.07E+01	7.12E+01	7.18E+01
MOEA/D-DE	Avg.	7.78E+00	7.81E+00	2.74E+00	3.57E+00	3.15E+00	1.69E+00

The distance score is a relative metric regarding the quality of the obtained Pareto front because of the absence of a true Pareto front. That's why the hyper-volume indicator is also used as a quality measure. *Figure 4.12* shows a comparison of the obtained hyper-volume averages for MOMA/D, MOEA /D - Local, NSGA III [127], ADEF, and MOFWA. This latter scores the best results in terms of hyper-volume in

all instances. Besides, the number of fitness function evaluations is very competitive on larger datasets.

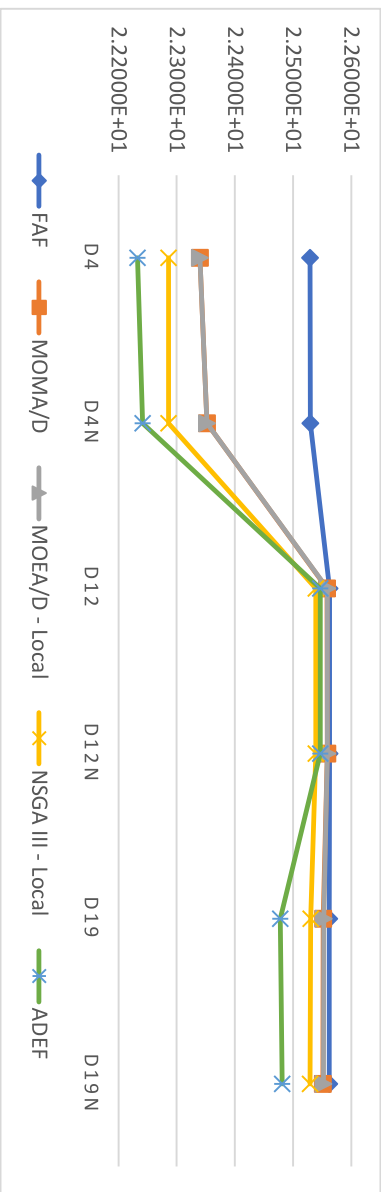


Figure 4.12 Comparison of the obtained hyper-volumes on all datasets

Figure 4.13 shows that MOFWA performs more fitness evaluations than the other methods in D4 and D4N. However, once the size of datasets gets bigger, the number of fitness evaluations for all other methods increases to become bigger than NFEES obtained by MOFWA. For example, NSGA III performs around 30% more evaluations in D12, D12N, D19, and D19N. Similarly, MOEA/D and MOMA/D score approximately 22% more evaluations than MOFWA.

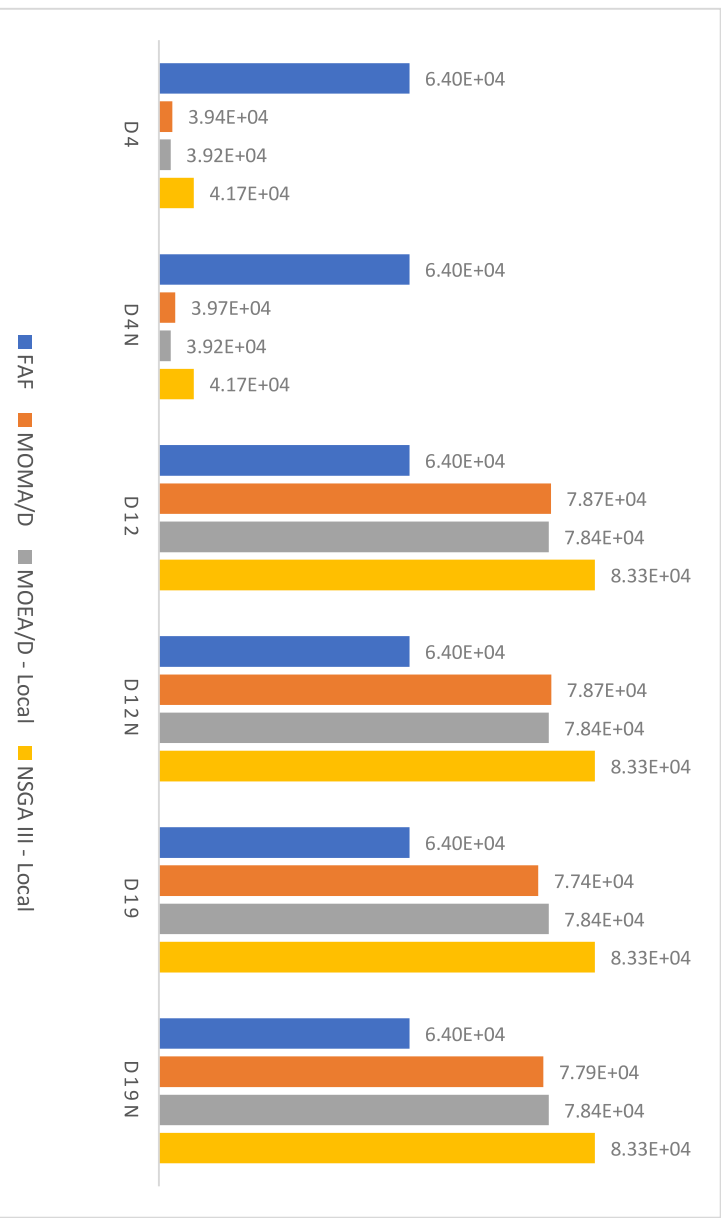


Figure 4.13 Fitness function evaluations comparison

4.3.5 Discussion & Insights

FAF shows interesting performance in single-objective optimization. The initialization phase plays an important role. Starting the search nearby the data matrix S can be a good choice in the case the amount of noise is low and thus the variation on S is small. Also, starting FAF with totally random solutions does not affect severely the convergence. This latter scenario is tested, and the results show that FAF is getting approximately the same solutions averages with 120% extra fitness function evaluations. These extra evaluations can be reduced by using a dedicated local search phase. However, this phase is not implemented in the model to avoid increasing the complexity. Even though, the single objective optimization can provide good results if FAF is driving a search relying mainly on minimizing the distance between S_i and S . Nevertheless, performing a search that focuses on the Pearson correlation optimization part is very important especially in highly noised data. This is clearly illustrated in the multi-objective optimization of BigOpt.

As stated before, FAF gets competitive results on multi-objective optimization. In fact, the Pareto fronts are well distributed in the fitness space. However, *Figure 4.14* shows that Pareto fronts obtained by ADEF partially dominate those obtained by FAF. Nevertheless, the hypervolume indicator and the score from baseline metrics (D12, D12N, D19; D19N) conclude that FAF is more performant. This can be explained by the high number of solutions that FAF can provide in the Pareto set. In other words, all the solutions found by ADEF dominate only few solutions of FAF. Hence, the FAF Pareto front is covering a bigger surface than ADEF, and consequently has a better hyper-volume indicator. *Table 4.11* shows the average number of solutions represented in the Pareto fronts of FAF and ADEF.

Like the single objective optimization, the lack of a dedicated local search effect can be seen in the obtained Pareto fronts. As shown by the little red rectangles in *Figure 4.11* and *Figure 4.14*, some small discontinuities in the Pareto fronts revealing that FAF fails to drive an extensive search over these areas. In fact, an effective implementation of a local search for BigOpt should take into consideration the rapid velocity of the processed data. Intentionally, a dedicated local search phase is not added to avoid an increase in complexity and to preserve the parallel design of FAF. Actually, a parallel GPU implementation of FAF is being designed. Thus, FAF needs a highly parallelized local search mechanism which is under investigation.

Table 4.3.6 Comparison of the average Pareto set sizes between MOFWA and ADEF

<i>Instance</i>	<i>D4</i>	<i>D4N</i>	<i>D12</i>	<i>D12N</i>	<i>D19</i>	<i>D19N</i>
<i>FAF</i>	468.1	479.2	362.7	348.7	363.6	418.3
<i>ADEF</i>	75.1	74.43	81.2	84.1	93.7	90.6

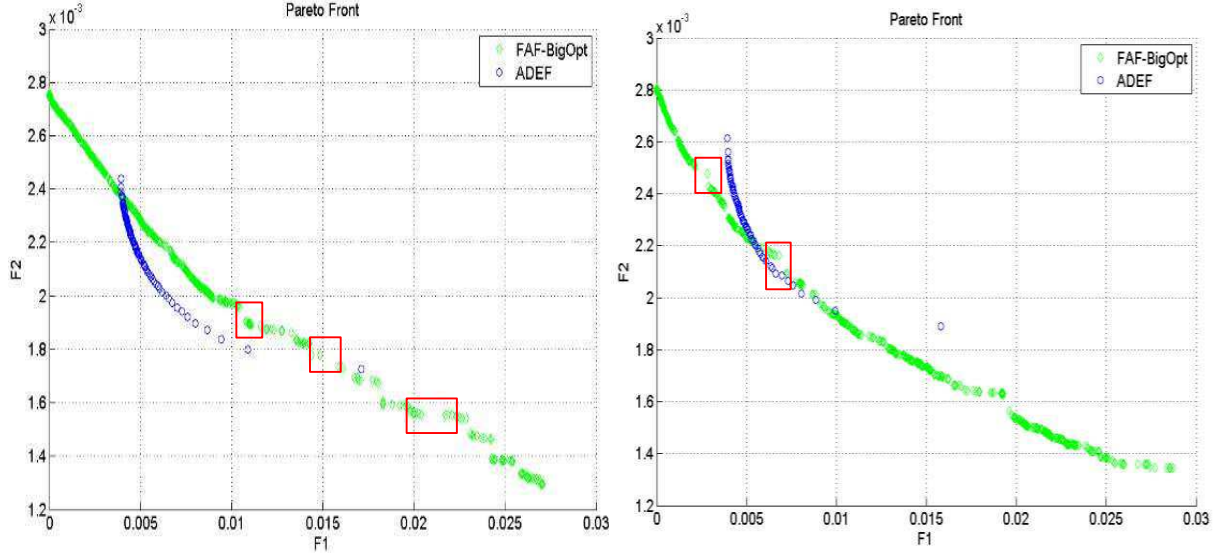


Figure 4.14 MOFWA and ADEF obtained Pareto fronts in D19 (Left) and D19N (Right)

4.3.6 Conclusion

This part of the chapter introduces a new Fireworks Algorithm based Framework (FAF) for Big Data optimization. In the proposed approach, two algorithms for single and multi-objective Big Data optimization problems are designed. To validate the framework performance, FAF is set to optimize the EEG data cleaning process as an instance of a Big Data optimization problem. The obtained results exhibit a competitive performance of the FAF algorithms in comparison to the recent state-of-the-art algorithms. The general behavior of FAF is also analyzed, concluding that a parallel local tuning mechanism can ameliorate considerably the obtained Pareto fronts on the multi-objective optimization.

Chapter V

SAPIAS: Towards Automation

5.1 Introduction: A “Meta-View” of Limitations

The Metaheuristics is an active research field for fifty years now. Many important advances have been achieved in several hard problems from transport, logistics, medicines, and many others. Comparing the public adoption of metaheuristics as a branch of artificial intelligence with another branch like neural networks, metaheuristics are still far from being considered as a ready-to-use product. Thus, a “why” question is imposed. Kendall in [128] has studied this question demonstrating the poor adoption of the industrial/commercial sectors of such solving techniques. One reason the study states is that the underlying benchmarking models are not quite relevant and less descriptive of the real-world problem being studied, leading to difficulties in successfully applying them commercially at a larger scale. Another possible reason we believe could be the ignorance of their potential in the optimization field as seen by non-experts. Indeed, metaheuristics are pointed out as untrusted non-deterministic stochastic procedures rather than proven solving tools. This can be seen by comparing the language used in the artificial intelligence community (AIC) and mathematical optimization community (MOC) where the term “solution of a given problem” is referred to as being a feasible solution for AIC, while the MOC refers to it as being the proven optimal solution. Additionally, the research efforts driven by an abusive interpretation of the No Free Lunch (NFL) theorem [129], has deviated the main goal of the research field, precisely by misusing the nature inspiration part which at the same time represents one of the main advantages and central aspects of metaheuristics. Two clear examples are given in [130] and [131]. Consequently, two opinions are confronted. The first one defends the nature inspiration originality despite the similarity with other algorithms, while the other requires a novelty in the engineered search mechanism.

As the majority of metaheuristics follow a modular composition scheme, two approaches can be exploited to reach a compromise between the two opinions. One approach is the Automatic Algorithm Configuration (AAC) [132]. Although this approach can be applied to any algorithm by automating its parameter tuning, AAC becomes more interesting when it comes to exploiting the modular aspect of metaheuristics. Indeed, most of metaheuristics follow a generic algorithmic scheme resumed in three interdependent conceptual components: *Initialize*, *Ameliorate*, and *Select*. This makes it easier for AAC to tune not only a set of parameters for a metaheuristic but can also define a grammar to generate many variants of the given metaheuristic and can go beyond for a hybridization with other algorithms without the need of a natural phenomenon inspiration. Hence, the main goal of using AAC for

metaheuristics is to find the best configuration of parameters and components to solve a set of problems. This configuration should have the best overall performance. However, the overall performance means a compromised good and poor performance in all instances, which is insufficient as a metric, especially if we target a large-scale adoption as discussed earlier. Also, this is undesirable for real-world scenarios as the real-world instances' landscapes are not controlled.

Another approach is the Per-instance Algorithm Selection (PiAS) approach [133]. Given a set of problem instances, a set of features that can characterize them, a set of algorithms, in our case metaheuristics, and a set of performance metrics to measure algorithm performance, PiAS tries to identify the best performing algorithm for each instance based on a prebuilt prediction model that binds the features to the algorithms' performance. However, this approach faces many limitations. For instance, the performance of a PiAS system is deeply influenced by factors like the accuracy of the underlying prediction model, feature space correlation, or incomplete performance information for new problems during the training phase. Another example of limitations is the information lack of the real performance of all non-predicted algorithms on a new problem instance during the prediction decision phase.

In this chapter, we describe an effort to address such limitations. A cooperative platform concept labeled as “the SAPIAS concept” is proposed. SAPIAS is composed of a self-adaptive online Per-instance Algorithm Selection system and an offline Automatic Algorithm Configuration system, working together to deliver the most accurate performance. A first analysis of the proposed concept shows that its adoption at a broad level can help to close the “why” gap mentioned earlier, by proposing a two-mode solving system where the online mode deals with problems solving while the offline mode deals with the system upgrade and maintenance ensuring its scalability.

5.1.2 Automatic Algorithm Configuration

First of all, an Algorithm Configuration problem can be formally defined as follows: Given a parameterized algorithm A with possible parameter settings P , a set of training problem instances I and a performance metric $m: I \times P \rightarrow IR$. The Algorithm Configuration problem is to find a parameter configuration p that optimizes m across the instances in I . *Figure 5.1* gives the flow-chart of automatic configuration.

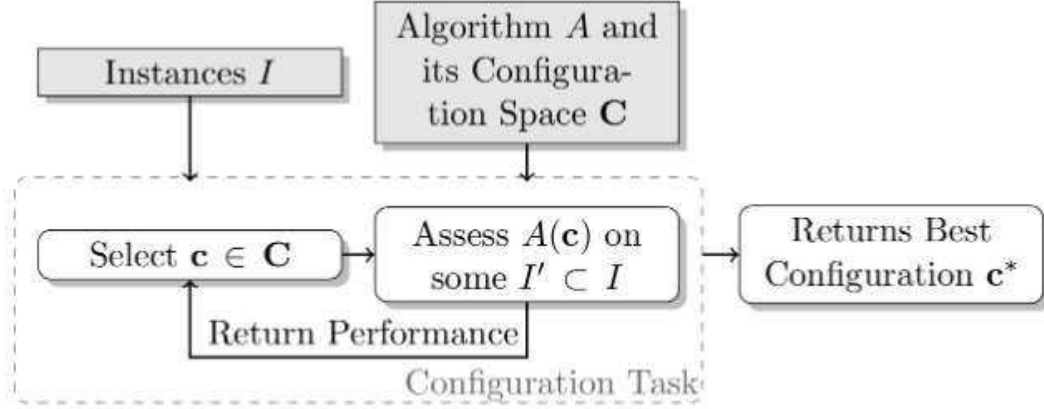


Figure 5.1 Algorithm Configuration

To tackle the Algorithm Configuration problem, many approaches have emerged. One approach uses combinations of relevant methods from the Design of Experiments (DOE) field such as response surface methodology [135], partial designs, local searches, and ANOVA [136,137]. Another approach applies racing procedures [138] based on an elitist selection among many configurations of which a small number of promising candidates are selected. This election is performed using sequential statistical testing [139]. The initial candidate configurations for a race are generated using random initializations, problem-specific knowledge, or DOE techniques. Among racing techniques, iterated racing is distinguished. It uses a sampling model which is iteratively updated during the race. A well-known implementation of this approach is *iRace* [133].

These approaches evaluate the configurations by explicitly using the instances of the problem at hand. Thus, the task becomes time-consuming when the instances are large. To mitigate this issue, another approach uses surrogate models to predict the quality of a configuration without running it on the actual instance(s). Afterward, the performance of the most promising configurations is assessed using the real instances and the prediction model is updated accordingly. Popular implementations of this approach are Sequential Parameter Optimization (SPOT) [140] and the Sequential Model-based Algorithm Configuration (SMAC) [141].

5.1.3 Per-instance Algorithm Selection

Formally, a Per-instance Algorithm Selection problem can be defined as follows [132]. Given a set I of problem instances, a distribution D over I , a space of algorithms A , and a performance measure $m: I \times A \rightarrow \mathbb{R}$, the Per-instance Algorithm Selection problem aims to find a mapping $s: I \rightarrow A$ that optimizes the performance measure

achieved by running the selected algorithm $s(i)$ for instance i across instances in I , drawn from the distribution D . To find such a mapping, a mathematical representation of an instance called an instance-features vector is used to describe the common characteristics of every instance. Using this vector, a mapping can be established between an instance and the algorithms using machine learning techniques. Indeed, a predictor of the best choice is provided as a function of the features vector provided. This choice can be adjusted depending on the desired strategy to solve. The most natural strategy would be the selection of one algorithm whose performance is optimal over the algorithm space as in [142]. For example, a simple classification model can be trained to attribute a group of instances to a given algorithm. Alternatively, a regression model can be trained to produce a prediction by recording the performance of each algorithm on each instance. Nevertheless, other strategies can be developed such as a schedule of algorithms in which the solving time budget is effectively distributed over a subset of chosen algorithms. This strategy appears to be more efficient as it allows the selection to diversify the solving methods in case of a poor prediction accuracy induced by machine learning techniques as opposed to the first strategy which copes only with one solving method.

Although PiAS seems to be a promising approach to tackle optimization problems using the different state-of-the-art metaheuristics, it induces many weaknesses to the solving process. Most of them are inherited from machine learning space. Indeed, since the produced model is not dynamically updated nor upgraded during the production phase when unseen instances are being processed, the problems of overfitting and underfitting can severely influence the performance of the system. Other problems arise in the feature extraction phases in case of incomplete information or biased sampling. To address these weaknesses, we propose the use of a hybrid system combining AAC and PiAS systems to dynamically update the prediction output of the Algorithm Selection and simultaneously monitor and prepare the prediction model for a future upgrade in case of a gradually decreasing performance.

5.2 SAPIAS: A Self Adaptive Per-Instance Algorithm Selection

The general concept of the proposed Self-Adaptive Per-Instance Algorithm Selection concept, labeled as the “SAPIAS” concept, is composed of four important cooperating layers. The first two layers are supposed to be implemented for an online execution mode and the last two layers for an offline mode as shown in the SAPIAS flow chart in *Figure 5.2*.

The first layer (Prediction layer) defines a PiAS system. This is the classical task where, based on a pre-built prediction model, a schedule of algorithms is selected to optimize a new given instance. Once the optimization process is achieved, the predicted schedule and the solution found are passed to the second layer called the Advisor layer. In the second layer (Advisor Layer), two operations are respectively triggered. The first one is using a control block that implements a low-budget “Prediction Enhancer” algorithm to optimize the new instance. The goal here is not to check the prediction accuracy in the precedent layer, but rather to detect if an improvement of the output solution of the predicted algorithm can be performed. Indeed, the Prediction Enhancer implements all the improvement strategies used for all algorithms present in the algorithm space used by the prediction model in the first layer. Afterward, if the solution is enhanced, the Prediction Enhancer identifies the algorithm(s) to which the enhancing improvement strategy(s) belongs. The second operation is then performed by grouping the detected algorithms in a schedule allowing them to contribute to the solving process along with the original algorithm once this latter is predicted again. Right after, the final solution is returned and the set of active algorithms in the schedule is passed to the next layer in the offline mode. The third layer (AAC Layer) aims to find an algorithm by automatically configuring different components used by the set of algorithms received. For this reason, a Bi-Phase Automatic Algorithm Configuration is established. In the first phase, an algorithm configuration is performed at the components level. Indeed, an algorithmic framework is constructed first, then an automatic configuration is performed to find a well-performing configuration. Besides the technical implementation of such a configuration, a very important point is that this configuration is evaluated using only the new problem instance at hand. Afterward, if a successful configuration is found, the next phase performs an extensive parameter tuning, however this time, all problem instances available are used. The parameter tuning should ensure in the first place a good parameter configuration for the new instance and then seek for others that perform better in other problem instances as well. Subsequently, a performance set for the new algorithm is generated by mapping its performance to each instance. The goal of the fourth layer is to merge this performance set into the performance space, automatically update the algorithm space, and rebind the outperformed algorithms to the new algorithm so that it becomes ready to use at the online mode instead of the outperformed algorithms.

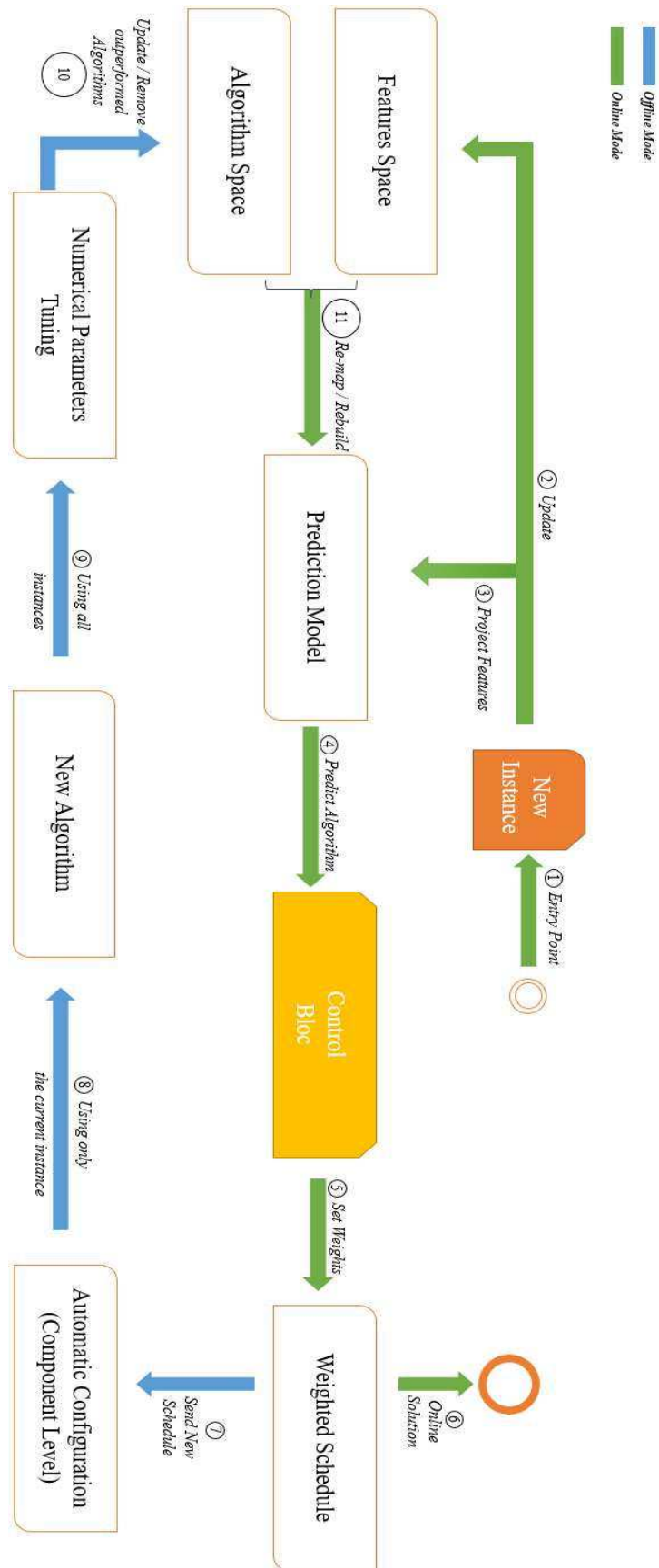


Figure 5.2: The SAPIAS concept flow-chart

5.2.1 Prediction Layer: Defining the Per-instance Algorithm Selection

In this layer, we introduce the Prediction layer mainly based on a PiAS system. Let's consider the following assumptions on the space of algorithms A of size n in which:

- An algorithm A_i is implemented using the modular scheme: Initialize module IN_i , Ameliorate module AM_i , and Select module SE_i as shown in *Figure 5.3*.
- Modules of an algorithm can be used in a standalone mode. This means every module can be executed by providing adequate inputs and without the need to run all of the algorithms' modules.
- Each module has its parameters and the exchanged information between modules includes solutions and their respective fitness.

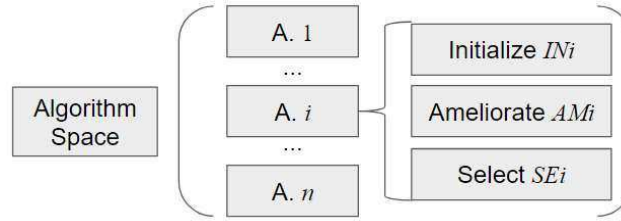


Figure 5.3 Algorithm Configuration

As shown in Figure 5.3, if A_i is a Genetic Algorithm for example, the three modules would be an Initialization module, an Offspring Generation module, and a Selection module, each of them operates on a set of solutions called population. Using the extracted features from pre-available problem instances with the performance space drawn from each algorithm performance over all instances, a prediction model can be built. Upon the arrival of a new instance I_i , this model will be used to select an algorithm A_i that is likely to solve better this instance based on the features projection results. The selected algorithm is then communicated to the next layer.

5.2.2 Advisor Layer: Setting up the solving process

At this layer, a Control Block is implemented where an internal schedule of algorithms is initiated such that:

$$SKDL(A_i) = \alpha_0 A_0 + \dots + \alpha_i A_i + \dots + \alpha_n A_n \quad (5.1)$$

where

$$\begin{cases} \alpha_i = 1 \\ \alpha_{j \neq i} = 0 \\ i, j \in [0, n] \end{cases}$$

for the initial prediction of the algorithm A_i .

This block mainly controls the execution of the solving process. Indeed, using a fixed time budget α_i initially set to 1, the predicted algorithm A_i is set to optimize the new instance I_i by generating an initial solution S_{init} to produce a solution S_i as follows:

$$S_i = A_i(S_{init}) \quad (5.2)$$

$$S_{final} = AM_n \circ AM_{n-1} \circ \dots \circ AM_1(S_i) \quad (5.3)$$

Next, the solution S_i is passed through all the Ameliorate modules implemented in the algorithm space A to identify one or more amelioration strategies that can enhance S_i . In the case where no improvement is detected, S_i is returned as a final solution of the instance I_i , otherwise, the solution S_{final} is returned and the internal schedule $SKDL(A_i)$ is updated for the algorithm A_i where the time budget α_i is decreased and the time budgets for the algorithms, where the improvements have been detected, are increased.

$$C = \{A_i / \alpha_i > 0\} \quad (5.4)$$

Once done, the set of algorithms C is passed to the next layer in the offline mode where an automatic configuration is conducted. The next time A_i is predicted, the optimization process is conducted by the schedule of algorithms $SKDL$ rather than A_i alone.

5.2.3 AAC Layer: Two-Phase Automatic Algorithm Configuration

In this layer, a bi-phase AAC is performed. Given the set of algorithms C received from the previous layer, the first AAC phase starts building new algorithms by hybridizing different algorithms' components to generate an algorithm B that has similar or better performance than the one recorded by $SKDL$ in the online mode. It is also important to note that the set of training instances I is reduced to only the new instance I_i . This is particularly performed for three reasons.

The first is that the current component used for the configuration surely contributes to optimizing I_i as they were communicated from $SKDL$, and thus we need them to produce an algorithm. The second reason is that we are particularly interested to have a working algorithm for this new instance as the predicted algorithm was not able alone to provide the best performance. Thus, future instances that would be considered closer

to I_i will have a better solving algorithm. The third reason is concerning the simplicity of the process, as starting a full configuration task including the component level and the numerical level with a large training set of instances would certainly make the task more complex and time-consuming.

Once a successful algorithm is achieved, the second *AAC* phase starts tuning the components parameters of the new algorithm B . In this phase, the whole training set of instances I is used. The main goal here is to record the performance set of B , then see if it can outperform other existing algorithms in other instance landscapes.

5.2.4 Update Layer: Online Update of Performance Space

After the recorded performance set of B is received, the existing performance space is updated, allowing the removal of the binding of algorithms where the new algorithm performs better in the respective instances. Once this operation is completed, the algorithm B is made available to the prediction layer immediately at the online mode instead of the replaced algorithms. Consequently, the control block in the second layer is notified to reset the schedules. Also, in this layer, the prediction model is also updated by regenerating a new one using extracted features from all instances including the newly received instances. This allows us to track the variation in the features characterizing the problem at hand rather than being limited to the initial set of features. This update is ruled using an update threshold monitoring the number of new instances or algorithms being (or waiting to be) made available to the online mode.

5.3 Questions and challenges

SAPIAS conceptually answers to some limitations found in Algorithm Selection and Automatic Algorithm Configuration by transforming the research questions being investigated for both approaches. Some of the remaining questions and their transformation in the SAPIAS concept are as follows.

- *What is the actual performance of the unpredicted algorithms?*

This question is one of the fundamental questions in PiAS particularly when for an unseen instance, no information is provided concerning the real performance of the available algorithms, only a similarity metric at the features level is used to determine which algorithm to run. However, it's not always safe to use this similarity metric as the prediction model can have a very tight confidence interval to decide the algorithm choice. This question is transformed to become:

- *Can unpredicted algorithms enhance further our prediction output performance?*

By answering this question, we do not change the prediction model paradigm, but the focus is mainly dedicated to enhancing the final solution which, after all, is the main goal. If the answer is yes, we can use this information to bring post-processing of the prediction task using the schedule strategy.

- *When can we have a satisfying instance sampling?*

This question is also inherited from machine learning where the model fitting problems like underfitting and overfitting are usually encountered. In SAPIAS, the question is transformed to be:

- *Can we have online learning of the problem structure to establish a feature-component relationship?*

The answer to this question is implemented in the post prediction schedule strategy which plays the role of an online corrector when the prediction is not accurate enough. Indeed, by modifying the weights in SKDL of each algorithm, SAPIAS can indirectly follow the evolution of the problem features evolution.

- *Is the time budget efficiently consumed?*

This question also tries to address the metric on which an algorithm choice is made. Indeed, a PiAS system may predict an algorithm where the Anytime Optimization performance can be very poor. In SAPIAS, the question is addressed differently:

- *Can we reallocate the budget efficiently?*

Using the schedule strategy, the algorithm weights are adjusted also using the anytime optimization metrics. This can be achieved by receiving feedbacks from the AAC layer in the offline mode where predicted algorithms having poor anytime performance are re-tuned and refined even if the control block does not detect an additional Ameliorate module.

- *How to upgrade the performance of a PiAS system with new algorithms?*

This is also one of the problems for PiAS systems where the insertion of new algorithms requires a rebuilding of the prediction model. This is a critical operation especially for online PiAS systems. The question is transformed as follows.

- *Can we configure efficient algorithms using the components we already have?*

Indeed, using the power of the AAC system implemented in the offline mode, the algorithm space can be dynamically updated with new algorithms using the available components of the algorithm space that are more likely to be suitable for the problem instance at hand. So, it's a per-instance targeted algorithm space update rather than

an integration of a new algorithm where the real performance on unseen instances is to be assessed.

- *Which components for which configuration? Why?*

In an AAC system, a big problem is encountered when the algorithm components' space is very large. Indeed, the configuration task becomes very complex computationally. Additionally, configuring the components for large-size instances using an overall performance metric makes the task more expensive and less effective. To alleviate this, the question is reformulated as follows.

- *Can the algorithms' components space be reduced?*

Having the ability to reduce the space will make it easier for PiAS to build a more accurate prediction model. It would also help at the configuration phase where the number of components can be controlled. This is exactly the purpose of the control block, where the detection of the Amelioration block is performed to assess the configuration task with concise components that would successfully and rapidly achieve. This is also implemented at the configuration level in the offline mode, where an unsuccessful components configuration is not suited by a numerical parameter tuning. Whereas, a successful one can give a new algorithm that performs well on the problem instance at hand, and then generalize the test to other instances to see whether this new algorithm can outperform existing algorithms and thus replace them. Consequently, the algorithm space is reduced.

Furthermore, the SAPIAS concept tries to deliver an autonomous solving system by exploiting the key benefits of AAC and PiAS systems. In fact, AAC ensures the evolvability of the PiAS system by providing it with new algorithms for unseen instances, and likewise, PiAS feeds AAC with a set of algorithms where their configuration procedure is more confident to produce a better algorithm.

However, many challenges can be identified that faces the real implementation of such a concept. For example, the "Amelioration modules" standalone behavior at the control block is not guaranteed to be similar to the behavior within the original algorithm. For an evolutionary algorithm for example, the use of such amelioration blocks is easy, however, for a swarm intelligence algorithm, particularly Particle Swam Optimization, this could be complicated because information like local and global best solutions are missing. It is also noticed that the currently designed control block does not make use of the Initialization and the Selection modules for complexity reasons. As a remedial to this situation, a learning phase is designed on top of the algorithm selection training

phase, and/or the performance space construction phase, to simulate the contribution of these modules to the performance of each algorithm. It is also planned in the midterm, to replace the amelioration modules in the control block with equivalent highly trusted Estimation Distribution Algorithms that will be used to draw solutions directly. Another issue is the choice of the adopted grammar for the first AAC phase on the component level i.e., Top/Down, Bottom/Up, or Fixed Grammar. This is a very important decision in the case of the hybridization of two different strategies. for example, it is easier to integrate the evolutionary process into the swarm intelligence algorithmic scheme but not the opposite. Another question is how much time budget the control block should be allowed. This is very critical in low-budget situations where the objective function evaluation is very expensive. This could be statistically solved by building models during the performance space construction or the training phases of the Algorithm Selection.

5.4 Conclusion

In this chapter, a description of a new concept of using metaheuristics is proposed. This concept, namely SAPIAS, uses a co-evolved Algorithm Selection and Automatic Algorithm Configuration systems to deliver the most advanced performance of the solving procedure. The SAPIAS concept is intended to be implemented as a community effort and be widespread and ready to use by industrial sectors to advance the evolution of the metaheuristics field more concisely and beneficially. By adopting this concept, the research trend that uses the nature-inspired aspect of metaheuristics to provide deprecated or renamed models is expected to be widely reduced. It is also expected that the SAPIAS concept would bring more research directions together to fill in the gap between theory and practice. Theoretical research can benefit from the analysis of the components' behavior implemented in the control block and link it with the dynamic feature space evolution. This latter is enriched using real-world instances coming from the industry, and consequently used to propose more accurate surrogate models or “pseudo-provable” parameter values for such components. Alongside this, practical research can benefit also by exploiting the capability of using and generating as many algorithm variants as possible being assisted with an underlying learning-based selection system.

General Conclusion

In this thesis, we contributed to the efforts of conceiving and developing novel metaheuristic techniques to solve NP-Hard and Big Data optimization problems.

Firstly, a new nature-inspired algorithm for unconstrained combinatorial optimization problems. The main concept of LISA is based on lightning observations and different lightning simulation models defined in the literature.

In LISA, atmospheric charged particles are considered as nodes being added, following the local potential and electric field intensity, to a breakdown or an upward charged leader based on their positions either near the cloud “NC” or near the ground “NG”. Once the breakdown leader meets up with the upward leader, a complete lightning path is formed to allow a cloud to ground discharge. The lightning path allowing the maximum discharge amount is considered as the optimal solution found. The breakdown leader and the upward leader construction spaces are then updated, simulating the electric current properties by exchanging positions of some NC particles with NG particles, and a new generation of upward and breakdown leaders is constructed. To prove its performance, LISA is applied to optimize the traveling salesman problem instances. The results on small and medium scale instances show high efficiency in comparison with other optimization techniques. As a perspective, the next step is to model a proper local tuning mechanism for LISA to enhance its explorative capacity for large-scale dimensions, and also extend LISA to constrained and multi-objective combinatorial optimization.

Secondly, we presented a Discrete Fireworks Algorithm DFA, as a novel adaptation of the original Fireworks Algorithm, to the single machine total weighted tardiness problem SMTWT. In this study, the fireworks explosion logic through sparks generation is preserved using two types of sparks; Exploration sparks and Exploitation sparks.. The obtained results are very competitive in comparison with other optimization techniques used to solve SMTWT. As a perspective, a fine-tuning mechanism is under investigation to ameliorate the exploitation phase capability for a future extension to the single machine total weighted tardiness with setup times.

Afterward, a study for solving big data optimization problems using the fireworks algorithm is emphasized. Indeed, we introduced a new Fireworks Algorithm Framework (FAF) for Big Data optimization. In the proposed approach, two algorithms for single and multi-objective Big Data optimization problems are designed respectively. To validate the framework performance, FAF is set to optimize the EEG data cleaning process as an instance of a Big Data optimization problem. The obtained results exhibit a competitive performance of FAF in comparison to the recent state-of-the-art

algorithms. Currently, a parallel implementation of FAF with a parallel local search method is under investigation. As a future work, the use of dimension reduction techniques is intended to be integrated into FAF to lower the computational complexity. Other highly constrained Big Data optimization problems are also of interest.

Finally, a description of a new concept for using metaheuristics is proposed by integrating, machine learning techniques such as Automatic algorithm configuration and per-instance algorithm selection approaches. This concept, namely SAPIAS, uses a co-evolved Algorithm Selection and Automatic Algorithm Configuration systems to deliver a more advanced solving process. By linking an offline mode AAC system and an online mode AS system using four layers, AS feeds AAC with a set of algorithmic components where their configuration procedure is more confident to produce a better algorithm. This latter is made available for the AS system to solve more accurately new unseen instances. This design answers some of the limitations known in both systems when used separately. It is expected that SAPIAS will deliver an independent behavior of the Algorithm Selection where the algorithm space is dynamically updated ensuring evolvability of the system. Using the amelioration blocks, not only the set of AAC algorithms to be detected is identified but also an enhanced output solution is found. SAPIAS also provides an indirect silent monitoring of the prediction model accuracy by using the schedules strategy, which helps to redress any model fitting problems of the Algorithm selection system. Moreover, a quicker AAC procedure is proposed by dividing the configuration task into two parts. The first one deals with the components' configuration, which upon success, the second part will conduct a numerical parameter tuning. It is also expected that the SAPIAS would bring more research directions together to fill in the gap between theory and practice. Actually, theory research can benefit from the analysis of the components' behavior implemented in the control block and link it with the feature space evolution, which is mainly updated using real world instances, in order to propose more accurate surrogate models or evidence- backed parameter values for such components. Alongside with this, practical research can benefit also by exploiting the capability of using and generating as many algorithms variants as possible while being assisted with an underlying learning-based selection system.

References

- [1] Rao, S. S. (2019). *Engineering optimization: theory and practice*. John Wiley & Sons.
- [2] Pardalos, P. M., & Romeijn, H. E. (Eds.). (2013). *Handbook of global optimization* (Vol. 2). Springer Science & Business Media.
- [3] Cook, S. (2006). The P versus NP problem. *The millennium prize problems*, pp. 87-104.
- [4] Flood, M. M. (1956). The traveling salesman problem. *Operations research*, 4(1), pp. 61-75.
- [5] Woeginger, G. J. (2003). Exact algorithms for NP-hard problems: A survey. In *Combinatorial optimization—eureka, you shrink!* pp. 185-207. Springer, Berlin.
- [6] Hussain, K., Salleh, M. N. M., Cheng, S., & Shi, Y. (2019). Metaheuristic research: a comprehensive survey. *Artificial Intelligence Review*, 52(4), pp. 2191-2233.
- [7] El Aferni, A., Guettari, M., & Tajouri, T. (2020). Mathematical model of Boltzmann's sigmoidal equation applicable to the spreading of the coronavirus (Covid-19) waves. *Environmental Science and Pollution Research*, pp. 1-9.
- [8] Talbi, E. G. (2009). *Metaheuristics: from design to implementation* (Vol. 74). John Wiley & Sons.
- [9] Glover, F. (1975). Improved linear integer programming formulations of nonlinear integer problems. *Management Science*, 22(4), pp. 455-460.
- [10] Jünger, M., Liebling, T. M., Naddef, D., Nemhauser, G. L., Pulleyblank, W. R., Reinelt, G. & Wolsey, L. A. (Eds.). (2009). *50 Years of integer programming 1958-2008: From the early years to the state-of-the-art*. Springer Science & Business Media.
- [11] Voorneveld, M. (2003). Characterization of Pareto dominance. *Operations Research Letters*, 31(1), pp. 7-11.
- [12] Gehlenborg, N., & Wong, B. (2012). Heat maps.

- [13] He, Z., & Yen, G. G. (2015). Visualization and performance metric in many-objective optimization. *IEEE Transactions on Evolutionary Computation*, 20(3), pp. 386-402.
- [14] Dantzig, G. B. (1951). Maximization of a linear function of variables subject to linear inequalities. *Activity analysis of production and allocation*, 13, pp. 339-347.
- [15] Nocedal, J., & Wright, S. J. (2006). Sequential quadratic programming. *Numerical optimization*, pp. 529-562.
- [16] Nelder, J. A., & Mead, R. (1965). A simplex method for function minimization. *The computer journal*, 7(4), pp. 308-313.
- [17] Boyd, S., & Mattingley, J. (2007). Branch and bound methods. *Notes for EE364b, Stanford University*.
- [18] Bellman, R. E., & Dreyfus, S. E. (2015). Applied dynamic programming. Princeton university press.
- [19] Williamson, D. P., & Shmoys, D. B. (2011). The design of approximation algorithms. Cambridge university press.
- [20] Sörensen K., Sevaux M., Glover F. (2018) A History of Metaheuristics.
- [21] Goldberg, D. E., Korb, B., & Deb, K. (1989). Messy genetic algorithms: Motivation, analysis, and first results. *Complex systems*, 3(5), pp. 493-530.
- [22] Goldberg, D. E., & Smith, R. E. (1987, October). Nonstationary Function Optimization Using Genetic Algorithms with Dominance and Diploidy. In *ICGA* pp. 59-68.
- [23] Han, K. H., & Kim, J. H. (2002). Quantum-inspired evolutionary algorithm for a class of combinatorial optimization. *IEEE transactions on evolutionary computation*, 6(6), pp. 580-593.
- [24] Levenick, J. R. (1991). Inserting Introns Improves Genetic Algorithm Success Rate: Taking a Cue from Biology. In *Proceedings of the Fourth International Conference on Genetic Algorithms*.

- [25] Bean, J. C. (1994). Genetic algorithms and random keys for sequencing and optimization. *ORSA journal on computing*, 6(2), pp. 154-160.
- [26] Hoai, N. X., McKay, R. I., & Essam, D. (2006). Representation and structural difficulty in genetic programming. *IEEE Transactions on evolutionary computation*, 10(2), pp. 157-166.
- [27] Ho, Y. C., & Pepyne, D. L. (2002). Simple explanation of the no free lunch theorem of optimization. *Cybernetics and Systems Analysis*, 38(2), pp. 292-298.
- [28] Beheshti, Z., & Shamsuddin, S. M. H. (2013). A review of population-based meta-heuristic algorithms. *Int. J. Adv. Soft Comput. Appl*, 5(1), pp. 1-35.
- [29] Lones, M. (2011) Sean Luke: essentials of metaheuristics. *Genet Program Evolvable Mach* 12, pp. 333–334
- [30] Van Laarhoven, P. J., & Aarts, E. H. (1987). Simulated annealing. In *Simulated annealing: Theory and applications* pp. 7-15. Springer, Dordrecht.
- [31] Glover, F., & Laguna, M. (1998). Tabu search. In *Handbook of combinatorial optimization* pp. 2093-2229. Springer, Boston, MA.
- [32] Salhi, S. (2002). Defining tabu list size and aspiration criterion within tabu search methods. *Computers & Operations Research*, 29(1), pp. 67-86.
- [33] Holland, J. H. (1992). Genetic algorithms. *Scientific american*, 267(1), pp. 66-73.
- [34] Pavai, G., & Geetha, T. V. (2016). A survey on crossover operators. *ACM Computing Surveys (CSUR)*, 49(4), pp. 1-43.
- [35] Zhong, J., Hu, X., Zhang, J., & Gu, M. (2005, November). Comparison of performance between different selection strategies on simple genetic algorithms. In *International conference on computational intelligence for modelling, control and automation and international conference on intelligent agents, web technologies and internet commerce (CIMCA-IAWTIC'06)* Vol. 2, pp. 1115-1121. IEEE.

- [36] Koza, J. R. (1995, November). Survey of genetic algorithms and genetic programming. In Wescon conference record pp. 589-594. WESTERN PERIODICALS COMPANY.
- [37] Burnet, F. M. (1957). A modification of Jerne's theory of antibody production using the concept of clonal selection. *Australian J. Sci.*, 20(3), pp. 67-9.
- [38] Iba, H., & Noman, N. (2016). *Evolutionary Computation in Gene Regulatory Network Research*. John Wiley & Sons
- [39] De Castro, L. N., & Timmis, J. (2002). *Artificial immune systems: a new computational intelligence approach*. Springer Science & Business Media.
- [40] Cutello, V., Nicosia, G., & Pavone, M. (2006, April). Real coded clonal selection algorithm for unconstrained global optimization using a hybrid inversely proportional hypermutation operator. In *Proceedings of the 2006 ACM symposium on Applied computing* pp. 950-954. ACM
- [41] De Castro, L. N., & Von Zuben, F. J. (2002). Learning and optimization using the clonal selection principle. *IEEE transactions on evolutionary computation*, 6(3), pp. 239-251.
- [42] De Castro, L. N., & Timmis, J. (2002, May). An artificial immune network for multimodal function optimization. In *Proceedings of the 2002 Congress on Evolutionary Computation. CEC'02 (Cat. No. 02TH8600) Vol. 1*, pp. 699-704. IEEE.
- [43] López-Ibáñez, M., Stützle, T., & Dorigo, M. (2018). *Ant Colony Optimization: A Component-Wise Overview*.
- [44] Chen, J., Xin, B., Peng, Z., Dou, L., & Zhang, J. (2009). Optimal contraction theorem for exploration–exploitation tradeoff in search and optimization. *IEEE Transactions on Systems, Man, and Cybernetics-Part A: Systems and Humans*, 39(3), pp. 680-691.

- [45] Dorigo, M., Maniezzo, V., & Coloni, A. (1996). Ant system: optimization by a colony of cooperating agents. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, 26(1), pp. 29-41.
- [46] Dorigo, M., & Gambardella, L. M. (1997). Ant colony system: a cooperative learning approach to the traveling salesman problem. *IEEE Transactions on evolutionary computation*, 1(1), pp. 53-66.
- [47] Stützle, T., & Hoos, H. H. (2000). MAX-MIN ant system. *Future generation computer systems*, 16(8), pp. 889-914
- [48] Kennedy, J., & Eberhart, R. (1995, November). Particle swarm optimization. In *Proceedings of ICNN'95-International Conference on Neural Networks Vol. 4*, pp. 1942-1948. IEEE.
- [49] Reynolds, R. G. (1994, February). An introduction to cultural algorithms. In *Proceedings of the third annual conference on evolutionary programming* pp. 131-139. River Edge, NJ: World Scientific.
- [50] Tereshko, V., & Loengarov, A. (2005). Collective decision making in honey-bee foraging dynamics. *Computing and Information Systems*, 9(3).
- [51] Karaboga, D., Gorkemli, B., Ozturk, C., & Karaboga, N. (2014). A comprehensive survey: artificial bee colony (ABC) algorithm and applications. *Artificial Intelligence Review*, 42(1), pp. 21-57.
- [52] Mitchell, D., Selman, B., & Levesque, H. (1992, July). Hard and easy distributions of SAT problems. In *AAAI Vol. 92*, pp. 459-465.
- [53] Yildiz, B. S., Lekesiz, H., & Yildiz, A. R. (2016). Structural design of vehicle components using gravitational search and charged system search algorithms. *Materials Testing*, 58(1), pp. 79-81.
- [54] L. Benameur, J. Alami, A. El Imrani, "A new discrete particle swarm model for the frequency assignment problem", *International Conference on Computer Systems and Applications (AICCSA) 2009*, pp. 139-144.

- [55] Collins, F. S., Patrinos, A., Jordan, E., Chakravarti, A., Gesteland, R., & Walters, L. (1998). New goals for the US human genome project: 1998-2003. *Science*, 282(5389), pp. 682-689.
- [56] Johnson, O., & Liu, J. (2006). Source Code for Biology and Medicine. *Source Code for Biology and Medicine*, 1, 3.
- [57] Mills, B., Unrau, D., Pentelow, L., & Spring, K. (2010). Assessment of lightning-related damage and disruption in Canada. *Natural hazards*, 52(2), pp. 481-499.
- [58] Uman, M. A. (2001). The lightning discharge. Courier Corporation. pp. 8-9
- [65] Kim, T., Sewall, J., Sud, A., & Lin, M. C. (2007). Fast simulation of laplacian growth. *Computer Graphics and Applications, IEEE*, 27(2), pp. 68-76.
- [59] Gurevich, A. V., & Zybin, K. P. (2005). Runaway breakdown and the mysteries of lightning. *Phys. Today*, 58(5), pp. 37-43.
- [60] Gomes, C., & Cooray, V. (2001). Leader fields due to a lightning channel with an inclined upper section. *Sri Lankan Journal of Physics*, 2, pp. 31-40.
- [61] Aslan, B. C. (2007). Continuous Approach to the Lightning Discharge (Doctoral dissertation, University of Florida).
- [62] Niemeyer, L., Pietronero, L., & Wiesmann, H. J. (1984). Fractal dimension of dielectric breakdown. *Physical Review Letters*, 52(12), pp. 1033-1036.
- [63] Perera, M. D. N., & Sonnadara, D. U. J. (2012). Research Article Fractal nature of simulated lightning channels. *Sri Lankan Journal of Physics*, 13(2), pp. 09-25.
- [64] Hoffman, K. L., Padberg, M., & Rinaldi, G. (2013). Traveling salesman problem. In *Encyclopedia of Operations Research and Management Science*, pp. 1573-1578. Springer US.
- [65] Zhou, M. X. (2014). Classic Lagrangian may not be applicable to the traveling salesman problem. arXiv preprint arXiv:1405.1298.

- [66] Dorigo, M., & Gambardella, L. M. (1997). Ant colony system: a cooperative learning approach to the traveling salesman problem. *Evolutionary Computation, IEEE Transactions on*, 1(1), pp. 53-66.
- [67] Stützle, T., & Hoos, H. H. (2000). MAX-MIN ant system. *Future generation computer systems*, 16(8), pp. 889-914.
- [68] Le Louarn, F. X., Gendreau, M., & Potvin, J. Y. (2004). GENI ants for the traveling salesman problem. *Annals of Operations Research*, 131(1-4), pp. 187-201.
- [69] Guo, P., & Liu, Z. (2011, July). Moderate ant system: An improved algorithm for solving TSP. In *Natural Computation (ICNC), 2011 Seventh International Conference on* Vol. 2, pp. 1190-1196. IEEE.
- [70] Escario, J. B., Jimenez, J. F., & Giron-Sierra, J. M. (2015). Ant colony extended: experiments on the travelling salesman problem. *Expert Systems with Applications*, 42(1), pp. 390-410.
- [71] Albayrak, M., & Allahverdi, N. (2011). Development a new mutation operator to solve the traveling salesman problem by aid of genetic algorithms. *Expert Systems with Applications*, 38(3), pp. 1313-1320.
- [72] Paul, P. V., Moganarangan, N., Kumar, S. S., Raju, R., Vengattaraman, T., & Dhavachelvan, P. (2015). Performance analyses over population seeding techniques of the permutation-coded genetic algorithm: An empirical study based on traveling salesman problems. *Applied Soft Computing*, 32, pp. 383-402.
- [73] Karaboga, D., & Gorkemli, B. (2011, June). A combinatorial artificial bee colony algorithm for traveling salesman problem. In *Innovations in Intelligent Systems and Applications (INISTA), 2011 International Symposium*, pp. 50-53. IEEE.
- [74] Othman, Z. A., Srour, A. I., Hamdan, A. R., & Ling, P. Y. (2013). Performance water flow-like algorithm for TSP by improving its local search. *International Journal of Advancements in Computing Technology*, 5(14), pp. 126.

- [75] Ouaarab, A., Ahiod, B., & Yang, X. S. (2014). Discrete cuckoo search algorithm for the travelling salesman problem. *Neural Computing and Applications*, 24(7-8), pp. 1659-1669.
- [76] Ouaarab, A., Ahiod, B., & Yang, X. S. (2014). Random-key cuckoo search for the travelling salesman problem. *Soft Computing*, 19(4), pp. 1099-1106.
- [77] Osaba, E., Yang, X. S., Diaz, F., Lopez-Garcia, P., & Carballedo, R. (2016). An improved discrete bat algorithm for symmetric and asymmetric Traveling Salesman Problems. *Engineering Applications of Artificial Intelligence*, 48, pp. 59-71.
- [78] Zhou, Y., Luo, Q., Chen, H., He, A., & Wu, J. (2015). A discrete invasive weed optimization algorithm for solving traveling salesman problem. *Neurocomputing*, 151, pp. 1227-1236.
- [79] Chen, S. M., & Chien, C. Y. (2011). Solving the traveling salesman problem based on the genetic simulated annealing ant colony system with particle swarm optimization techniques. *Expert Systems with Applications*, 38(12), pp. 14439-14450.
- [80] Mahi, M., Baykan, Ö. K., & Kodaz, H. (2015). A new hybrid method based on Particle Swarm Optimization, Ant Colony Optimization and 3-Opt algorithms for Traveling Salesman Problem. *Applied Soft Computing*, 30, pp. 484-490.
- [81] Elloumi, W., El Abed, H., Abraham, A., & Alimi, A. M. (2014). A comparative study of the improvement of performance using a PSO modified by ACO applied to TSP. *Applied Soft Computing*, 25, pp. 234-241.
- [82] Geng, X., Chen, Z., Yang, W., Shi, D., & Zhao, K. (2011). Solving the traveling salesman problem based on an adaptive simulated annealing algorithm with greedy search. *Applied Soft Computing*, 11(4), pp. 3680-3689.
- [83] Créput, J. C., & Koukam, A. (2009). A memetic neural network for the Euclidean traveling salesman problem. *Neurocomputing*, 72(4), pp. 1250-1264.
- [84] Avşar, B., & Aliabadi, D. E. (2015). Parallelized neural network system for solving Euclidean traveling salesman problem. *Applied Soft Computing*, 34, pp. 862-873.

- [85] Reinelt, G. (1995). TSPLIB 95 documentation. University of Heidelberg
- [86] Geng, X., Chen, Z., Yang, W., Shi, D., & Zhao, K. (2011). Solving the traveling salesman problem based on an adaptive simulated annealing algorithm with greedy search. *Applied Soft Computing*, 11(4), pp. 3680-3689
- [87] Saenphon, T., Phimoltares, S., & Lursinsap, C. (2014). Combining new fast opposite gradient search with ant colony optimization for solving travelling salesman problem. *Engineering Applications of Artificial Intelligence*, 35, pp. 324-334.
- [88] Marinakis, Y., Marinaki, M., & Dounias, G. (2011). Honey Bees Mating Optimization Algorithm for the Euclidean traveling salesman problem. *Information Sciences*, 181(20), pp. 4684-4698.
- [89] Dowlatshahi, M. B., Nezamabadi-Pour, H., & Mashinchi, M. (2014). A discrete gravitational search algorithm for solving combinatorial optimization problems. *Information Sciences*, 258, pp. 94-107.
- [90] Johnson, D. S., McGeoch, L. A., Rego, C., & Glover, F. (2001). 8th DIMACS implementation challenge. URL <http://www.research.att.com/~dsj/chtsp.f>
- [91] Hoos, H.H., Stutzle, T., *Stochastic Local Search: Foundations and Applications*, Morgan Kaufmann Publishers, 2005.
- [92] S. Lin, Computer solutions of the traveling salesman problem, *Bell Systems Technical Journal* 44 (1965) pp. 2245–2269.
- [93] Babu, P., Peridy, L., & Pinson, E. (2004). A branch and bound algorithm to minimize total weighted tardiness on a single processor. *Annals of Operations Research*, 129(1-4), pp. 33-46.
- [94] Palominos, P., Araya, P., & Silva, P. (2012). A new heuristic for the single machine total weighted tardiness problem.
- [95] Tasgetiren, M. F., Sevkli, M., Liang, Y. C., & Gencyilmaz, G. (2004, June). Particle swarm optimization algorithm for single machine total weighted tardiness

- problem. In Proceedings of the 2004 Congress on Evolutionary Computation (IEEE Cat. No. 04TH8753) Vol. 2, pp. 1412-1419. IEEE.
- [96] Dorigo, M., Caro, G. D., & Gambardella, L. M. (1999). Ant algorithms for discrete optimization. *Artificial life*, 5(2), pp. 137-172.
- [97] Ferrolho, A., & Crisóstomo, M. (2007, May). Single machine total weighted tardiness problem with genetic algorithms. In 2007 IEEE/ACS International Conference on Computer Systems and Applications pp. 1-8. IEEE.
- [98] Sabamoniri, S., Asghari, K., & Hosseini, M. J. (2012). Solving single machine total weighted tardiness problem using variable structure learning automata. *International Journal of Computer Applications*, 56(1).
- [99] Liu, W., & Yan, F. (2012, October). On the population diversity control of evolutionary algorithms for production scheduling problems. In 2012 IEEE Fifth International Conference on Advanced Computational Intelligence (ICACI) pp. 468-473. IEEE.
- [100] Tan, Y. (2015). *Fireworks algorithm*. Heidelberg, Germany: Springer, 10, 978-3.
- [101] Maheswaran, R., & Ponnambalam, S. G. (2003). An investigation on single machine total weighted tardiness scheduling problems. *The International Journal of Advanced Manufacturing Technology*, 22(3-4), pp. 243-248.
- [102] Rustogi, K., & Strusevich, V. A. (2015). Single machine scheduling with time-dependent linear deterioration and rate-modifying maintenance. *Journal of the Operational Research Society*, 66(3), pp. 500-515.
- [103] Feng, Q., Li, S. S., & Shang, W. P. (2015). Two-agent single machine scheduling with forbidden intervals. *Applied Mathematics-A Journal of Chinese Universities*, 30(1), pp. 93-101.
- [103] Hansen, P., & Mladenović, N. (2005). Variable neighborhood search. In *Search methodologies* (pp. 211-238). Springer, Boston, MA.

- [104] Parsopoulos, K. E., & Vrahatis, M. N. (2006, December). Studying the performance of unified particle swarm optimization on the single machine total weighted tardiness problem. In *Australasian Joint Conference on Artificial Intelligence* pp. 760-769. Springer, Berlin, Heidelberg.
- [105] Aljanaby, A., & Ku-Mahamud, K. R. (2011). Interacted multiple ant colonies optimization approach for the single machine total weighted tardiness problem.
- [106] Nearchou, A. C. (2004, June). Solving the single machine total weighted tardiness scheduling problem using a hybrid simulated annealing algorithm. In *2nd IEEE International Conference on Industrial Informatics, 2004. INDIN'04. 2004* pp. 513-516. IEEE.
- [107] Garlasu, D., Sandulescu, V., Halcu, I., Neculoiu, G., Grigoriu, O., Marinescu, M., & Marinescu, V. (2013, January). A big data implementation based on Grid computing. In *Roedunet International Conference (RoEduNet), 2013 11th* pp. 1-4. IEEE.
- [108] Han, J., Haihong, E., Le, G., & Du, J. (2011, October). Survey on NoSQL database. In *Pervasive computing and applications (ICPCA), 2011 6th international conference on* pp. 363-366. IEEE.
- [109] PWG, N. B. D. (2015). *NIST Big Data Interoperability Framework. Vol1, Definitions* pp. 4-12.
- [110] Walls-Esquivel, E., Vecchierini, M. F., Héberlé, C., & Wallois, F. (2007). Electroencephalography (EEG) recording techniques and artefact detection in early premature babies. *Neurophysiologie Clinique/Clinical Neurophysiology*, 37(5), pp. 299-309.
- [111] <http://www.husseinabbass.net/BigOpt.html>. Accessed: 25 April 2016
- [112] Curran, E. A., & Stokes, M. J. (2003). Learning to control brain activity: a review of the production and control of EEG components for driving brain-computer interface (BCI) systems. *Brain and cognition*, 51(3), pp. 326-336.

- [113] Murthy, J. M. K. (2003). Some problems and pitfalls in developing countries. *Epilepsia*, 44(s1), pp. 38-42.
- [114] Parvinnia E, Sabeti M, Zolghadri Jahromi M, Boostani R (2014) Classification of EEG Signals using adaptive weighted distance nearest neighbor algorithm. *J King Saud Univ Comput Inf Sci* 26(1): pp. 1-6
- [115] Goh SK, Abbass HA, Tan KC, Al-Mamun A (2015) Decompositional independent component analysis using multi-objective optimization. *Soft Computing*, pp 1-16
- [116] An adaptive configuration of differential evolution algorithms for big data. In *Evolutionary Computation (CEC)*, 2015 IEEE Congress, pp. 695-702. IEEE. DOI: 10.1109/CEC.2015.7256958
- [117] Elsayed, S., & Sarker, R. (2016). Differential evolution framework for big data optimization. *Memetic Computing*, pp. 1-17.
- [118] Omidvar M, Li X, Mei Y, Yao X (2014) Cooperative co-evolution with differential grouping for large scale optimization. *IEEE Trans Evolut Comput* 18(3), pp. 378-393.
- [119] Zhang J, Sanderson AC (2009) Jade: adaptive differential evolution with optional external archive. *IEEE Trans Evolut Comput* 13(5), pp. 945-958
- [120] Tanabe R, Fukunaga A (2013a) Evaluating the performance of shade on cec 2013 benchmark problems. In: *IEEE congress on evolutionary computation*, pp. 1952-1959. doi:10.1109/CEC.2013. 6557798
- [121] Goh, S. K., Tan, K. C., Al-Mamun, A., & Abbass, H. A. (2015, May). Evolutionary big optimization (BigOpt) of signals. In *Evolutionary Computation (CEC)*, 2015 IEEE Congress, pp. 3332-3339. IEEE.
- [122] Zhang, Y., Zhou, M., Jiang, Z., & Liu, J. (2015, May). A multi-agent genetic algorithm for big optimization problems. In *Evolutionary Computation (CEC)*, 2015 IEEE Congress, pp. 703-707. IEEE.

- [123] Zhang, Y., Liu, J., Zhou, M., & Jiang, Z. (2016). A multi-objective memetic algorithm based on decomposition for big optimization problems. *Memetic Computing*, 8(1), pp. 45-61.
- [124] <http://www.husseinabbass.net/AuxDataNTools.pdf>. Accessed: 25 April 2016
- [125] Zitzler, E., & Thiele, L. (1998, September). Multiobjective optimization using evolutionary algorithms—a comparative case study. In *Parallel problem solving from nature—PPSN V* pp. 292-301. Springer Berlin Heidelberg
- [126] Qingfu Z, Wudong L, Hui L (2009) The performance of a new version of moea/d on cec09 unconstrained mop test instances. In: *IEEE congress on evolutionary computation*, pp. 203–208
- [127] Zhang, Y., Liu, J., Zhou, M., & Jiang, Z. (2016). A multi-objective memetic algorithm based on decomposition for big optimization problems. *Memetic Computing*, 8(1), pp. 45-61.
- [128] Kendall, G. 2018. Is Evolutionary Computation evolving fast enough? *IEEE Computational Intelligence Magazine*, 13, 2, pp. 42-51.
- [129] Ho, Y. C., & Pepyne, D. L. 2002. Simple explanation of the no free lunch theorem of optimization. *Cybernetics and Systems Analysis* 38, 2, pp. 292-298.
- [130] Weyland, D. 2015. A critical analysis of the harmony search algorithm—How not to solve sudoku. *Operations Research Perspectives*, 2, pp. 97-105.
- [131] Camacho-Villalón, C. L., Dorigo, M., & Stützle, T. 2018, October. Why the Intelligent Water Drops Cannot Be Considered as a Novel Algorithm. In *International Conference on Swarm Intelligence*. Springer, Cham. pp. 302-314.
- [132] Manuel López-Ibáñez, Jérémie Dubois-Lacoste, Leslie Pérez Cáceres, Thomas Stützle, and Mauro Birattari. The irace package: Iterated Racing for Automatic Algorithm Configuration. 2016. *Operations Research Perspectives*. 3, pp. 43–58.
- [133] Bischl, Bernd; Kerschke, Pascal; Kotthoff, Lars; Lindauer, Marius; Malitsky, Yuri; Fréchet, Alexandre; Hoos, Holger; Hutter, Frank; Leyton-Brown, Kevin;

- Tierney, Kevin; Vanschoren, Joaquin. 2016. ASlib: A benchmark library for algorithm selection. *Artificial Intelligence*. 237, pp. 41–58. arXiv:1506.02465. doi:10.1016/j.artint.2016.04.003
- [134] Preuss, M. (2015). *Multimodal optimization by means of evolutionary algorithms*. Springer International Publishing.
- [135] Coy SP, Golden BL, Runger GC, Wasil EA. Using experimental design to find effective parameter settings for heuristics. *J Heuristics* 2001 7(1), pp. 77–97.
- [136] Ridge E, Kudenko D. Tuning the performance of the MMAS heuristic. In: Stützle T, Birattari M, Hoos HH, editors. *International workshop on engineering stochastic local search algorithms (SLS 2007)*, volume 4638 of *lecture notes in computer science*. Heidelberg, Germany: Springer; 2007. pp. 46–60.
- [137] Ruiz R, Maroto C. A comprehensive review and evaluation of permutation flowshop heuristics. *Eur J Oper Res* 2005;165(2), pp. 479–94.
- [138] Maron O, Moore AW. The racing algorithm: model selection for lazy learners. *Artif Intell Res* 1997;11(1–5), pp. 193–225.
- [139] Birattari M, Stützle T, Paquete L, Varrentrapp K. A racing algorithm for configuring metaheuristics. In: Langdon WB, editor. *Proceedings of the genetic and evolutionary computation conference, GECCO 2002*. San Francisco, CA: Morgan Kaufmann Publishers; 2002. pp. 11–18.
- [140] López-Ibáñez, M., Dubois-Lacoste, J., Cáceres, L. P., Birattari, M., & Stützle, T. (2016). The irace package: Iterated racing for automatic algorithm configuration. *Operations Research Perspectives*, 3, pp. 43–58.
- [140] Bartz-Beielstein T, Lasarczyk C, Preuss M. Sequential parameter optimization. In: *Proceedings of the 2005 congress on evolutionary computation (CEC 2005)*. Piscataway, NJ: IEEE Press; 2005. pp. 773–80.
- [141] Hutter F, Hoos HH, Leyton-Brown K. Sequential model-based optimization for general algorithm configuration. In: Coello Coello CA, editor. *Learning and intelligent*

optimization, 5th international conference, LION 5, volume 6683 of lecture notes in computer science. Heidelberg, Germany: Springer; 2011. pp. 507–23.

[142] Nudelman, E., Leyton-Brown, K., Andrew, G., Gomes, C., McFadden, J., Selman, B., & Shoham, Y. (2003). Satzilla 0.9, solver description. International SAT Competition, pp. 25.

[143] EL MAJDOULI, Mohamed Amine & EL IMRANI Abdelhakim Ameer. Lightning Inspired Search Algorithm: Introduction & application to the traveling salesman problem. In : Proceedings of 10th International Conference on Intelligent Systems: Theories and Applications (SITA). IEEE, 2015. pp. 1-6.

[144] EL MAJDOULI, Mohamed Amine et EL IMRANI, Abdelhakim Ameer. Discrete fireworks algorithm for single machine scheduling problems. In : International Journal of Applied Metaheuristic Computing (IJAMC), 2016, vol. 7, no 3, pp. 24-35.

[145] EL MAJDOULI Mohamed Amine, BOUGRINE Saad, RBOUH Ismail & EL IMRANI Abdelhakim Ameer. Fireworks Algorithm for single objective big optimization of signals. In : Proceedings of IEEE/ACS 13th International Conference of Computer Systems and Applications (AICCSA). IEEE, 2016. pp. 1-7.

[146] EL MAJDOULI Mohamed Amine, BOUGRINE Saad, RBOUH Ismail, El Benani Bouazza & EL IMRANI Abdelhakim Ameer. Fireworks algorithm framework for Big Data optimization. In : Memetic computing, 2016, vol. 8, no 4, pp. 333-347.

[147] EL MAJDOULI Mohamed Amine, BOUGRINE Saad, RBOUH Ismail & EL IMRANI Abdelhakim Ameer. A comparative study of the EEG signals big optimization problem using evolutionary, swarm and memetic computation algorithms. In : Proceedings of the genetic and evolutionary computation conference companion. 2017. pp. 1357-1364.

[148] EL MAJDOULI, Mohamed Amine. SAPIAS concept: towards an independent self-adaptive per-instance algorithm selection for metaheuristics. In : Proceedings of the Genetic and Evolutionary Computation Conference Companion. 2019. pp. 1474-1477.

[149] The ORLIB library, <http://people.brunel.ac.uk/~mastjjb/jeb/orlib/whinfo.html>,
accessed: 17/09/21.

Résumé :

Ce mémoire décrit des travaux de recherche menés pour la conception de nouvelles métaheuristiques et leurs applications aux problèmes d'optimisation NP-Difficiles et Big Data. En effet, quatre contributions majeures sont apportées. La première contribution présente un nouveau modèle de recherche inspiré de la nature appelé « Lightning-Inspired Search Algorithm ». Le modèle utilise un mécanisme constructif, similaire au modèle de la formation naturelle de la foudre, qui favorise la diversité pendant la recherche pour éviter une convergence prématurée. Les expériences réalisées sur le problème du voyageur de commerce montrent un comportement très compétitif de l'algorithme. La deuxième contribution présente une adaptation de l'algorithme « Fireworks Algorithm (FA) » aux problèmes d'ordonnancement à une seule machine. L'algorithme proposé modifie le mécanisme d'exploration et d'exploitation de la phase de génération d'étincelles. Les résultats expérimentaux sont satisfaisants par rapport aux algorithmes de l'état de l'art. La troisième contribution concerne l'application de FA à des problèmes d'optimisation de big data. L'algorithme original FA est revisité pour améliorer sa vitesse de convergence tout en préservant sa robustesse. Les résultats montrent que la solution proposée surpasse toutes les méthodes proposées pour BigOPT. La dernière contribution introduit de nouvelles idées sur le développement des métaheuristiques, en introduisant des techniques de configuration automatique d'algorithmes et de sélection d'algorithmes par instance, pour une utilisation plus intelligente des métaheuristiques dans les applications futures.

Mots-clés : Métaheuristiques, Problèmes d'optimisation NP-difficiles, Fireworks Algorithm, BigOPT, SMTWT

Abstract:

This work describes an effort towards the conception of novel metaheuristics and their application to NP-hard and big data optimization problems. Indeed, four major contributions are reported in this thesis. The first contribution introduces a new nature-inspired search model called "Lightning Inspired Search Algorithm". The model uses a constructive mechanism, similar to the pattern of the natural lightning formation, that promotes diversity during the search to avoid premature convergence. The experiments performed over the traveling salesman problem show a very competitive behavior of the algorithm. The second contribution presents an adaptation of the Fireworks Algorithm to single machine scheduling problems. The proposed algorithm redesigns the exploration and exploitation mechanism implemented by the spark's generation phase. The experimental results are satisfactory in comparison to state-of-the-art algorithms. The third contribution concerns the application of the Fireworks Algorithm to big data optimization problems. The original fireworks algorithm is revisited to enhance the speed of convergence while preserving its robustness. The results show that the proposed solution outperforms all proposed methods for BigOPT. The last contribution offers a new design pattern for using metaheuristics based on automatic algorithm configuration and per-instance algorithm selection techniques for a smarter usage of metaheuristics in future applications.

Keywords: Metaheuristics, NP-Hard problems, Fireworks Algorithm, BigOPT, SMTWT.

Année Universitaire : 2020/2021