PEOPLE'S DEMOCRATIC AND REPUBLIC OF ALGERIA
MINISTRY OF HIGH EDUCATION AND SCIENTIFIC RESEARCH

N° Série : ……/2024

Kasdi Merbah Ouargla University

Faculty of New Technologies of Information and Communication
Department of computer science and information technologies

DISSERTATION

To obtain master's degree

Option: Fundamental computing

Presented by:

ATTAB Ismail, GHERIANI Mohammed Aymene

-TITLE-

Integration of Machine Learning into Local Search Meta-heuristics for the Sports Scheduling Problem

Publicly defended on: 24/06/2024

Jury:

President:	MESSAID Abd Elsselam	Univ. Ouargla
Examiner:	MECHALIKH Charafeddine	Univ. Ouargla
Supervisor:	KHELIFA Meriem	Univ. Ouargla

College year: 2023/24

Acknowledgement

Above all, I thank ALLAH, the Almighty, for providing me with the bravery and patience to finish my dissertation.

First, I'd like to thank my supervisors, Ms. KHELIFA Meriem, for her invaluable patience and feedback during this project. for her insightful comments.

I would be remiss in not mentioning the faithful and truthful teachers, for their full dedication for the sake of students, and their guidance in my journey in the Kasdi Merbah Ouargla University's department of computer science and information technology.

Last but not least, I express my heartfelt gratitude to the jury members Mr. MESSAID Abd Elsselam and MR. MECHALIKH Charafeddine for their tolerance and taking the time to assess my work.

Dedication

It’s an honor to be able to express my gratitude for the people I love the most.

I wholeheartedly dedicate this work to my beloved mother KERRI Zohra, she has made me who I am today and did nothing less than everything to make sure that I am loved, save, supported, guided and most of all successful, my inspiration who never fail to encourage me morally, spiritually, and emotionally…

To my dear father ATTAB Abdelkader for his absolute support and tremendous effort in raising me all this time.

To my cornerstone and brother BOUAICHA Mohamed Abdelhai.

To my brothers, IMADEDDIN, ABDESSALAM, Med AYMEN and ZOUBIR.

To my sisters, MERIEM, HALA, HALIMA and ZINA.

To my beloved family and friends, all by his name. Specially BOUAICHA Mohamed Abdelhai for his invaluable advice over the past few years, and my dear friend BENHAMIDA Iskander may ALLAH grant him heaven.

Finally, to all who helped me during my journey and made it special and full of memories.

ATTAB Ismail

Dedication

It’s an honor to express my gratitude to the people I love the most.

I dedicate this dissertation to my loving parents, whose unconditional love, unwavering support, and sacrifices have been the cornerstone of my journey. Your belief in me has fueled my determination, and your guidance has shaped my character. This accomplishment is as much yours as it is mine. Thank you for being my pillars of strength and for instilling in me the values of perseverance and resilience.

To my brothers Imad, Ramy and Amir.

To my entire family and all my friends.

To Virage Sud Mouloudeen Who lifts my spirits every week.

To myself because I suffered a lot lo live this day

GHERIANI Mohamed Aymene

Abstract

This thesis addresses the **Traveling Tournament Problem (TTP)**, an **NP-hard combinatorial optimization** challenge in sports scheduling. The TTP is crucial in the sports community due to its impact on league management budgets, where suboptimal schedules can lead to significant financial losses. To tackle this problem, we propose an innovative approach that integrates **machine learning** and **deep learning** techniques to enhance the performance of **Stochastic Local Search (SLS)** and **Biogeography-Based Optimization (BBO)** algorithms. It is well known that the main challenge in the **evolutionary algorithms** is to set the best parameters. we employ a deep learning model for parameter tuning. Additionally, we proposed a novel approach to incorporate reinforcement learning within the SLS algorithm during the exploitation phase to enhance its performance. The computational experiments indicate that our method produces competitive and promising results, showcasing its potential for effective TTP optimization.

Key words: Traveling Tournament Problem (TTP), NP-hard combinatorial optimization, machine learning, deep learning, Stochastic Local Search (SLS), Biogeography-Based Optimization (BBO), evolutionary algorithms.

الملخص

تتناول هذه الاطروحة مشكلة السفر في البطولات (TTP) وهي مشكلة تحسين تركيبية صعبة في جدولة الرياضات. يحتل TTPأهمية كبيرة ضمن المجتمعات الرياضية، حيث يمكن أن تتسبب التحسينات غير المثلى في خسائر مالية في ميزانية إدارة الدوريات. لمعالجة هذه المشكلة، نقترح نهجًا مبتكرًا يدمج التعلم الآلي وتقنيات التعلم العميق لتعزيز أداء خوارزميات البحث المحلي العشوائي (SLS) والتحسين القائم على الجغرافيا الحيوية (BBO). من المعروف أن التحدي الرئيسي في الخوارزميات التطورية هو تحديد أفضل الإعدادات. نستخدم نموذج التعلم العميق لضبط الإعدادات. بالإضافة إلى ذلك، اقترحنا نهجًا جديدًا لدمج التعلم المعزز ضمن خوارزمية SLS أثناء مرحلة الاستغلال لتحسين أدائها. تشير التجارب الحسابية إلى أن طريقتنا تنتج نتائج تنافسية وواعدة، مما يعرض إمكاناتها لتحسين TTP الفعال.

الكلمات المفتاحية:

مشكلة السفر في البطولات , مشكلة تحسين تركيبية صعب , التعلم الآلي, التعلم العميق, خوارزميات البحث المحلي العشوائي (SLS), والتحسين القائم على الجغرافيا الحيوية (BBO)

Résumé

Cette thèse aborde le problème du voyage dans les tournois (TTP), un défi d'optimisation combinatoire NP difficile dans la programmation sportive. Le TTP est crucial dans la communauté sportive en raison de son impact sur les budgets de gestion de la ligue, où les horaires sous-optimaux peuvent entraîner des pertes financières importantes. Pour résoudre ce problème, nous proposons une approche innovante qui intègre des techniques d'apprentissage automatique et d'apprentissage profond pour améliorer les performances des algorithmes de recherche locale stochastique (SLS) et d'optimisation basée sur la biogéographie (BBO). Il est bien connu que le principal défi des algorithmes évolutifs est de définir les meilleurs paramètres. Nous utilisons un modèle d'apprentissage profond pour le réglage des paramètres. De plus, nous avons proposé une nouvelle approche pour intégrer l'apprentissage par renforcement dans l'algorithme SLS pendant la phase d'exploitation afin d'améliorer ses performances. Les expériences informatiques indiquent que notre méthode produit des résultats compétitifs et prometteurs, démontrant son potentiel pour une optimisation efficace du TTP

Mot clés :

problème du voyage dans les tournois, d'optimisation combinatoire NP difficile, apprentissage automatique, apprentissage profond, recherche locale stochastique, d'optimisation basée sur la biogéographie

Table of contents

List of figures:

General introduction

Context:

Human life is filled with challenges and difficulties; however, humans have the unique ability to learn from past experiences and continuously improve to enhance their quality of life and achieve goals more efficiently. This evolution and progress are evident across various fields, including engineering, economics, finance, management, sports, and operations research. One significant aspect of this progression is the ability to solve optimization problems, which involves finding the most efficient or effective way to allocate resources, make decisions, or design systems. Optimization is crucial in decision-making and problem-solving as it evaluates different options to identify the best course of action. It also plays a vital role in improving processes and systems, reducing costs, and increasing efficiency.

Modern optimization techniques leverage algorithms and computer programs to handle large and complex problems, allowing for faster and more accurate optimization. These tools have significantly contributed to the growth of optimization across many fields. There are various methods to solve optimization problems, including exact methods and approximate algorithms. Exact methods yield optimal solutions but often require significant time, making them unsuitable for solving the most complex problems (NP-hard problems). Approximate algorithms, including heuristic and meta-heuristic approaches, can provide high-quality solutions within a reasonable time, though they do not guarantee globally optimal solutions.

In the context of sports scheduling, the Traveling Tournament Problem (TTP) is a prominent NP-hard combinatorial optimization challenge. The TTP aims to create a double round-robin tournament schedule that minimizes the total distance traveled by teams while adhering to specific constraints. This problem is crucial in the sports community as suboptimal schedules can lead to significant financial losses in league management budgets.

To tackle the TTP, we propose an innovative approach that integrates Biogeography-Based Optimization (BBO) and Stochastic Local Search (SLS) algorithms enhanced by machine learning and deep learning techniques. Our methodology begins with generating initial solutions and defining neighborhood structures. The BBO algorithm is adapted to the TTP by implementing these structures and swap methods. Additionally, we employ a deep learning

model for parameter tuning and incorporate reinforcement learning within the SLS algorithm to further optimize the scheduling process.

Our proposed methodology has demonstrated promising results in optimizing tournament schedules, reducing total travel distance, and adhering to the specific constraints of the TTP. The integration of deep learning for parameter tuning and reinforcement learning within the SLS algorithm has shown to be effective in addressing the complexities of the TTP.

Looking forward, our future research will focus on enhancing the efficiency and scalability of our solutions, particularly in tackling larger TTP instances. We plan to explore alternative strategies to improve the effectiveness of our approach, such as refining the machine learning models and investigating other evolutionary algorithms. Additionally, we aim to expand the applicability of our methods to other domains, further bridging the gap between traditional optimization techniques and modern machine learning approaches.

Problematic:

Although optimization has advanced considerably nowadays, it is always receptive to new better methods and algorithms, The performance of an algorithm considered effective today may be deemed inadequate in the future, especially given the rapid evolution of artificial intelligence (AI) across various domains. This continuous progress in AI presents an opportunity to leverage its capabilities to enhance optimization techniques. Add to that, There is a constant need to propose new solutions and refine existing ones to address the ever-growing and complex set of optimization challenges.

Objective:

The primary objective of this research is to enhance the performance of Stochastic Local Search (SLS) and Biogeography-Based Optimization (BBO) algorithms in solving the TTP by incorporating machine learning and deep learning techniques. This hybrid approach aims to balance exploration and exploitation of the search space, thereby improving the overall solution quality.

Organization of the Thesis:

To comprehensively present our work, the thesis is structured as follows:

- Chapter 1: Provides a review of the main definitions and fundamental issues related to optimization problems.

- Chapter 2: Provides a review of the main definitions of the machine learning

- Chapter 3: Describes the Traveling Tournament Problem (TTP), the main focus of this study, and gives an overview of past approaches to solving the TTP.

- Chapter 4: Describes The Biogeography-Based Optimization.

- Chapter 5: Details the proposed approach for solving the TTP, including the integration of machine learning and deep learning techniques to enhance SLS and BBO algorithms.

- Chapter 6: Highlights the experiments conducted and compares the results of our approach with the best results available in the literature.

Through this structured approach, the thesis aims to contribute to the field of optimization by offering a novel method for solving the TTP, demonstrating its efficacy through rigorous experimentation and comparison with existing methods.

CHAPITRE I: Combinatorial optimization

1 Introduction

The inherent complexity of real-world systems, processes, equipment, and devices necessitates the development of optimization methodologies. Even seemingly simple systems often require theoretical models that incorporate approximations, time-dependent parameters, or random variations. These imperfections are unavoidable but crucial for predicting the optimal operating conditions that maximize a specific performance metric. At best, such models can only provide an approximation of the system's proximity to the desired optimum. Optimization techniques become essential tools in this scenario. They enable exploration of the local operational region and prediction of parameter adjustments necessary to drive the system towards its optimal state [1].

Optimization lies at the heart of any problem that necessitates decision-making. The core of decision-making involves selecting from a set of available alternatives. This selection process is guided by the pursuit of the "best" choice. An objective function, also known as a performance index, serves as the measure of merit for these alternatives. Optimization theory and methods strive to identify the optimal alternative based on the defined objective function.

The field of optimization has witnessed a surge in interest in recent years, primarily driven by the rapid advancements in computer technology. These advancements encompass the development and widespread availability of user-friendly software, high-speed and parallel processing capabilities, and artificial neural networks. A compelling illustration of this phenomenon is the proliferation of optimization software tools [2].

In this chapter, we will cover the basic concepts and fundamental points of combinatorial optimization. This includes:

- The concept of optimization: We will establish a clear understanding of what it means to optimize a system or process.
- Identifying problems that require optimization: We will explore real-world scenarios where optimization techniques become crucial for achieving the best possible outcome.
- Various solution methods: We will delve into a variety of approaches used to solve optimization problems in combinatorial optimization.

- Complexity classes: Finally, we will learn how to classify optimization problems based on their inherent computational difficulty.

2 Problems optimization

2.1 Introduction to problems optimization

Search space: a finite or infinite set of solutions used to find an optimal or near-optimal solution by a search algorithm [3].

Exploration: During the intensification phase in combinatorial optimization, the search algorithm focuses its exploration on the local neighborhood of a promising solution identified within the search space [4].

Exploitation: In optimization algorithms, exploitation refers to the strategy of iteratively refining solutions within the vicinity of previously explored regions of the search space. This approach aims to enhance the quality of existing solutions but can lead to premature convergence at local optima, thereby hindering the discoveries of globally optimal solutions [4].

Global optimum: A feasible solution is considered a global optimum if there is no other viable solution within the entire viable region that has a strictly better objective function value. A global optimum can be manifested as a global maximum (the value of the highest objective function) or a global minimum (the lowest objective function value) [5].

Local optimum: a local optimal solution is a feasible solution where the objective function achieves a good value (minimum or maximum) compared to its immediate neighbors [6].

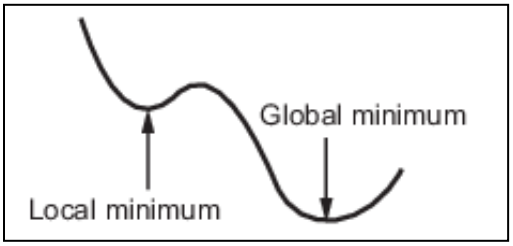


Figure I.1: Local vs. Global Optima [7

Combinatorial optimization is a subdomain of discrete optimization that involves identifying an optimal element in a finite set of candidate solutions. These candidate solutions are called feasible solutions, and the element with the most favorable objective function value is designated as the optimal solution [8].

optimization is the minimization or maximization of a function subject to constraints on its variables. We use the following notation: x is the vector of variables, also called unknowns or parameters; f is the objective function, a (scalar) function of x that we want to maximize or minimize; - ci are constraint functions, which are scalar functions of x that define certain equations and inequalities that the unknown vector x must satisfy [9].

Using this notation, the optimization problem can be written as follows:

$$\min_{x\in\mathbb{R}} f(x) \text{ subject to } \begin{cases} ci(x) = 0, i\epsilon\varepsilon \\ ci(x) \geq 0, i\epsilon\Gamma \end{cases}$$

There are two main classes for optimization problems:

2.1.1 Constrained optimization

A mathematical framework for decision-making in the field of mathematical optimization, limited optimization problems focus on identifying the optimal (minimum or maximum) value of an objective function. However, this quest is not boundless. A set of constraints restricts the feasible

region, dictating acceptable solutions. In simpler terms, these constraints define a set of viable options that meet certain pre-determined conditions. Common constraint formulas include inequalities, such as $x \geq 0$ (representing non-negative values for variable x) and equals, expressed as $Ax = b$ (where x is a vector of decision variables, A is a matrix representing coefficients, and b is a constant vector). To solve these restricted optimization problems, there are several established methods, including the simplex method (both its algebraic and matrix-based forms) and graphical methodology. These methodologies provide a systematic approach to navigate the feasible region and locate the optimal solution that adheres to the defined constraints.

2.1.2 Unconstrained optimization

Unconstraint optimization problems, characterized by the absence of constraints (i.e., &= I = Ø in the equation (1.1)), occur naturally in many practical applications. This category includes scenarios in which decision variables operate freely across the area. Even in situations where inherent limitations (natural constraints) might exist for variables, it may be strategically wise to ignore them under certain conditions. This is especially true if these constraints have demonstrated no impact on the optimal solution and do not prevent the optimization algorithms chosen.

Furthermore, optimization problems without constraints can be strategically constructed by reformulating these problems. In such reformulations, the initial constraints are replaced by criminal clauses incorporated into the objective function. These criminal conditions effectively discourage breaches of the original constraints, guiding the process of optimization toward solutions that implicitly adhere to the underlying limitations. This approach takes advantage of well-developed theoretical frameworks and solution methodologies available for problem-free optimization to solve problems with inherent constraints.

2.1.3 Constrained optimization VS unconstrained optimization

Now, consider the beam shown in Figure. It is subjected to moments at the ends and a load P at the center and rests on a Winkler tensionless elastic foundation of modulus K. The corresponding deformation pattern is shown in Figure, where two contact regions and one central noncontact region are observed. Here, t_i and t_f constitute the additional variables of the problem.

Figure I.2 shows the influence of the foundation stiffness on the behavior of the beam, where the lateral deflection along the beam axis is given for different values of the $k = KL^4/EI$. Once more, there is good agreement between the results obtained by the dual-constrained numerical formulation and the Ritz method, in which the same displacement field (5.1) is adopted. Under unilateral constraints, the beam displacement w increases in the central region while the contact regions between the bodies decrease.

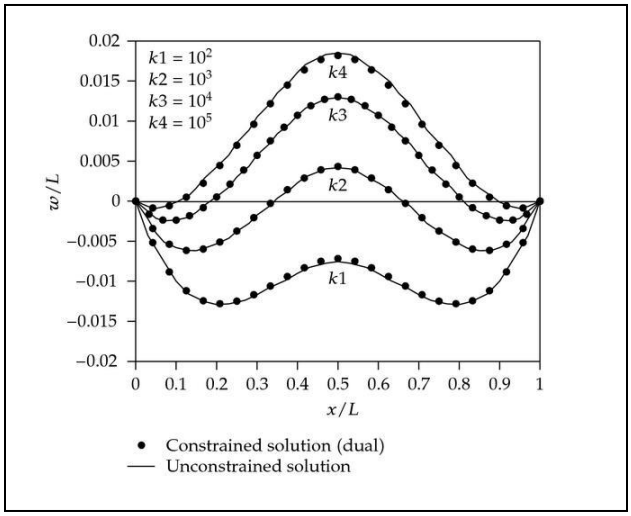


Figure I.2: Deformed shape of the beam for several values of the foundation stiffness.

2.2 Penalty function

Methods of penalty: This section explores methods for approximating restricted optimization issues to non-restrictive ones. This approach allows us to leverage established research techniques to solve problems without constraints, in order to face problems with limitations. Approximation depends on the modification of the objective function. Methods of penalty: Here, a term is added to the objective function that significantly penalizes violations of constraints. The extent of the penalty increases as the degree of violation of the coercion increases. This effectively discourages the research process from venturing into areas that are not practical.

The penalty method works directly in the n-dimensional space of the decision variables, where n represents the number of variables and m is the number of constraints. While the following

discussion focuses on methods of penalty. Their main distinction lies in how they influence the research process to move toward a solution that adheres to constraints [10].

Consider the problem

where f is continuous function on $\mathcal{R}^n$and S is a constraint set in $\mathcal{R}^n$. In

most applications S is defined explicitly by a number of functional

$$Minimize\ \{f(x): x \in S\} \qquad (I.1)$$

constraints, but in this section the more general description can be handled. The idea of a penalty function method is to replace the problem by an unconstrained approximation of the form.

$$Minimize\ \{f(x) + cP(x)\} \qquad (I.2)$$

where c is a positive constant and P is a function on $\mathcal{R}^n$satisfying (i) $P(x)$ is continuous,

(ii) $P(x) \geq 0$ for all $x \in \mathcal{R}^n$, and (iii) $P(x) = 0$ if and only if $x \in S$.

Here, each unsatisfied constraint influences x by assessing a penalty equal to the square of the violation. These influences are summed and multiplied by c, the penalty parameter. Of course, this influence is counterbalanced by $f(x)$. Therefore, if the magnitude of the penalty term is small relative to the magnitude of $f(x)$, the minimization of (c, x) will almost certainly not result in an x that would be feasible for the original problem. However, if the value of c is made suitably large, the penalty term will exact such a heavy cost for any constraint violation that the minimization of the augmented objective function will yield a feasible solution.

The function $cP(x)$ is illustrated in Fig for the one-dimensional case with g1 (x) = b – x and g2(x) = x-a. For large c it is clear that the minimum point of problem will be in a region where P is small. Thus, for increasing c it is expected that the corresponding solution points will approach the feasible region S and, subject to being close, will minimize f. Ideally then, as c→∞ the solution point of the penalty problem will converge to a solution of the constrained problem.

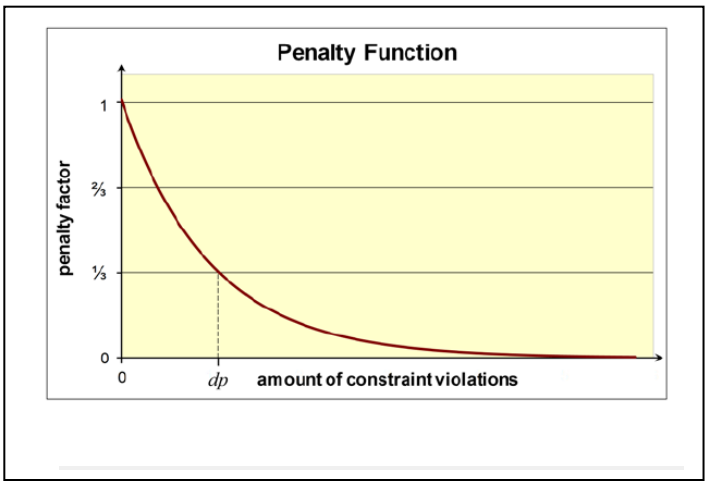


Figure I.3: Relationship Between Penalty Factor and Amount of Constraint Violations in Penalty Function

Selecting the Penalty Parameter: is to start with a relatively small value of c and an infeasible (exterior) point. This will ensure that no steep valleys are present in the initial optimization of (c, x). Subsequently, we will solve a sequence of unconstrained problems with monotonically increasing values of c chosen so that the solution to each new problem is "close" to the previous one. This will preclude any major difficulties in finding the minimum of (c, x) from one iteration to the next.

To implement this strategy, let $\{ck\}, k = 1, 2, \ldots$ be a sequence tending to infinity such that $ck > 0$ and $ck + 1 > ck$. Now for each k we solve the problem.

$$Minimize\ \{(c_k, \boldsymbol{x}): \boldsymbol{x} \in \mathcal{R}^n\} \qquad \text{(I.3)}$$

To obtain $\mathbf{x}^k$, the optimum It is assumed that problem has a solution fr all positive values of c_k. This will be true, for example, if $\theta(\mathrm{c}, \mathrm{x})$ increases without bounds as $||\mathbf{x}|| \rightarrow \infty$.

A simple implementation known as the sequential unconstrained minimization technique (SUMT), is given below.

Initialization Step: Select a growth parameter $\eta > 1$, a stopping parameter $\varepsilon > 0$, and an initial value of the penalty parameter c_0.

Choose a starting point x_0 that violates at least one constraint increases without bounds as $||\mathbf{x}|| \rightarrow \infty$. And formulate the augmented objective function (c_0, x). Let k=1.

Iterative Step: Starting from $\mathbf{x}^{k-1}$, use an unconstrained search technique to find the point that minimizes $\theta(c_{\mathrm{k\_1}}, \mathrm{x})$. Call it $\mathbf{x}^k$ and determine which constraints are violated at this point.

Stopping Rule: If the distance between $\boldsymbol{x}^{k-1}$ and $\mathbf{x}^k$ is smaller than ε $(i.e., ||\boldsymbol{x}^{k-1} - \boldsymbol{x}^k|| \leq \varepsilon)$ or the difference between two successive objective functions values is smaller than ε, stop with $\mathbf{x}^k$

An estimate of the optimal solution. Otherwise, put $c^k \leftarrow \eta c^{k-1}$ formulate the new $\theta(c_{\mathrm{k\_1}}, \mathrm{x})$ based on which constraints are violated at $\boldsymbol{x}^k$ put $k \leftarrow k + 1$ and return to the iterative step.

2.3 Computational Complexity

2.3.1 Complexity of Algorithms

The effective execution of algorithms relies on two crucial resources: time and space. In the context of algorithmic analysis, time complexity refers to the number of basic operations (or steps) it needs to solve a problem of the size n. (often representing the input size). This complexity is usually determined by a worst-case analysis, taking into account the maximum number of steps required for any size n entry.

The main objective of computational complexity analysis is not to obtain a precise number of steps for each algorithm. Instead, it aims to establish an asymptotic link in the number of stages. This limitation describes the behavior of the runtime of the algorithm as the input size increases by an infinite size. Big-O notation, a common tool in algorithmic analysis, facilitates this asymptotic analysis. It provides a way to express the upper limit on the number of steps an algorithm takes based on the input size, ignoring constants and lower-order terms. As a result, the Big-O rating

allows a comparative evaluation of different algorithms to solve the same problem, highlighting their relative effectiveness based on their growth of runtime relative to input size [11].

2.3.2 Big-O notation

An algorithm has a complexity $f(n) = O(g(n))$ if there exist positive constants n0 and c such that $\forall n > n0, f(n) \leq c \cdot g(n)$. In this case, the function $f(n)$ is upper bounded by the function g(n). The Big-O notation can be used to compute the time or the space complexity of an algorithm [11].

2.3.3 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:

$$p(n) = ak \cdot nk + \cdots + aj \cdot nj + \cdots + a1 \cdot n + a0$$

where $ak > 0$ and $aj \geq 0, \forall 1 \leq j \leq k - 1$. The corresponding algorithm has a polynomial complexity of $O(nk)$.

2.3.4 Exponential-time algorithm

An algorithm is classified as exponential time if its time complexity can be expressed as $O(c^n)$, where:

c is a positive real constant strictly greater than 1 (denoting the base of the exponential term).
n represents the size of the problem (often measured by the number of variables or constraints).
This phenomenon is often referred to as the "combinatory explosion".
In practical applications, waiting for centuries to solve a problem is simply not feasible. The use of an exhaustive research algorithm (which has an exponential complexity) to solve a problem of a certain size would require an impracticably long time, exceeding the estimated age of the universe.

Big-θ notation

An algorithm has a complexity f (n) = _(g(n)) if

there exist positive constants n0, c1, and c2 such that $\forall n > n0$, c1 · $g(n) \leq f(n) \leq c2 \cdot$ g(n). The complexity of the algorithm $f(n)$ is lower bounded by the function $g(n)$.

This analysis often begins with the determination of the Big-O notation, which provides an upper limit on the time complexity of an algorithm as the size of the problem increases. This initial evaluation can then be refined by calculating the Big-θ rating, which captures both th upper and lower limits, providing a more accurate characterization of the algorithm's time complexity. However, the most specific limit is provided by the Big-Omega notation, which defines the exact lower limit on the complexity of time.

2.3.5 Asymptotic analysis

Which encompasses these notations, focuses on the rate of growth of the time complexity of an algorithm depending on the size of the problem. This analysis provides a theoretical framework for comparing algorithms according to their complexity in the worst case, enabling researchers to evaluate how much an algorithm evolves with the increase in the size of the problem. It is important to note that asymptotic analysis focuses on theoretical performance assurances and does not predict the exact execution time for a specific problem instance. In fact, the execution time of an algorithm is strongly influenced by the nature of the input data. For a more comprehensive understanding, the analysis of the average complexity of cases can be performed, although it is often a more difficult task.

Here are some common Big O notations and their corresponding growth rates:

- O (1) - Constant Time Complexity:

The algorithm's running time remains constant regardless of the input size. It indicates that the algorithm has a fixed number of operations and does not depend on the input.

- O (log n) - Logarithmic Time Complexity:

The algorithm's running time increases logarithmically with the input size. These algorithms typically divide the problem into smaller subproblems and solve them iteratively.

- O(n) - Linear Time Complexity:

The algorithm's running time scales linearly with the input size. Each element of the input is processed once, resulting in a direct relationship between the input size and the running time.

- O(n log n) - Linearithmic Time Complexity:

The algorithm's running time grows at a rate slightly higher than linear. It is often seen in sorting algorithms like merge sort and quicksort.

- O(n^2) - Quadratic Time Complexity:

The algorithm's running time grows quadratically with the input size. It commonly occurs in nested loops where each element needs to be compared with every other element.

- O(2^n) - Exponential Time Complexity:

The algorithm's running time grows exponentially with the input size. These algorithms are highly inefficient and become infeasible for large inputs.

2.4 Complexity of Problems

2.4.1 Introduction

In the field of computational complexity theory, the difficulty of a problem is often measured by the complex of the most effective algorithm that solves it. A problem is considered to be treatable, or easy to solve, if there is an algorithm that can find a solution within a polynomial timeframe relative to the input size of the problem. Polynomial time refers to an algorithm whose execution time increases proportionally to a polynomic function of the input size. Conversely, a problem is classified as intractable, or difficult, if no algorithm can guarantee a solution in polynomial time regardless of the size of the input. This distinction between treatable and untreatable problems is a fundamental concept of complexity theory.

It is important to note that complexity theory mainly focuses on decision problems. A decision problem is characterized by a definitive yes or no answer. By analyzing the complexity of solving

these decision-making problems, complexity theory highlights the inherent difficulty of larger problem classes.

2.4.2 Complexity Classes

A fundamental concept of the theory of computational complexity is the classification of problems into complexity classes. Each complexity class encompasses a set of solved problems with a specific amount of computer resources. Two important classes are P and NP (as illustrated in Figure).

2.4.2.1 The class P (polynomial)

represents all the decision problems solved by a deterministic Turing machine in polynomic time. We define an algorithm (determinist in this context) as polynomial for a decision problem A if its complexity in the worst case is limited by a polynomic function p(n), where n means the size of the input instance (I). Essentially, P encompasses problems for which a known polynomial time algorithm exists to come up with a solution. Therefore, P problems are considered to be relatively "easy" to solve from a computational point of view.

2.4.2.2 The complexity class NP (Nondeterministic Polynomial Time)

encompasses all the decision problems for which a solution can be verified in polynomial time by a deterministic algorithm. A key feature of NP problems is the existence of a non-determinist algorithm [14]. Non-determinist algorithms differ from their deterministic counterparts by incorporating "choice points". At these points, the algorithm can be branched into several possible continuations without any predetermined selection of the path to follow. The algorithm uses a set of fundamental primitives:

Choice: This primitive proposes a potential solution, often called "oracle" or "certificate". Check: This primitive checks the proposed solution (certificate) in polynomial time, determining whether it leads to a positive or negative answer for the decision problem. Successful: If the primary test confirms a positive response, the algorithm successfully solves the problem.

Failure: If the control primitive does not give a positive answer, the algorithm does not solve the problem.

The basic concept of NP lies in the relationship between choice and control primitives. If the primitive choice proposes a valid solution that, after verification by the primary control, leads to a "yes" answer, and the control can be carried out in polynomial time, then the problem is considered to be in NP. In simpler terms, NP problems are those where a solution can be effectively verified (polynomial time) by a deterministic algorithm, even though finding the solution itself could be computationally more difficult.

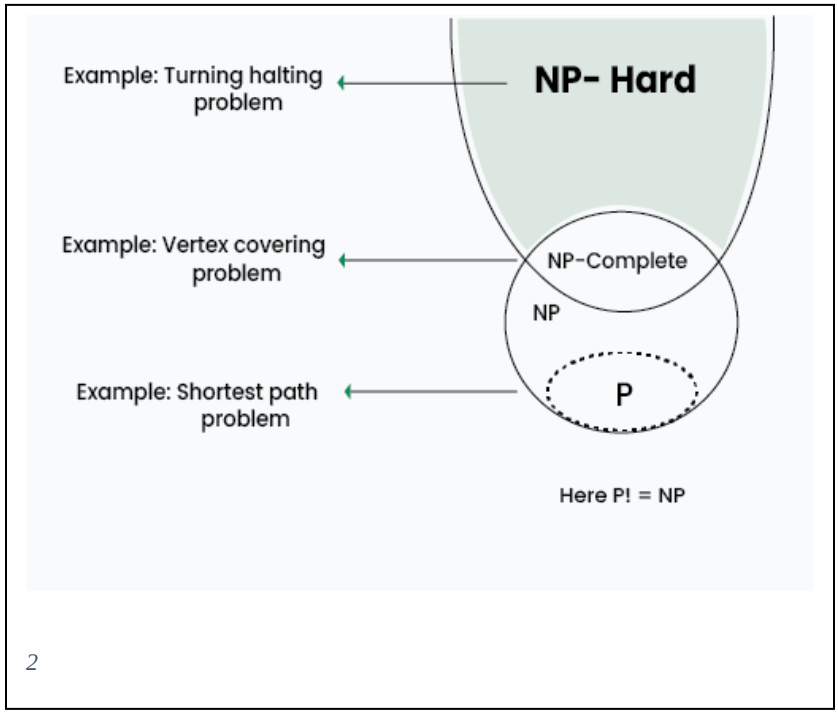


Figure I.4: Classification of Computational Problems: P, NP, NP-Complete, and NP-Hard

2.4.2.3 Example of Nondeterministic algorithm for the 0–1 knapsack problem

The 0–1 knapsack decision problem can be defined as follows. Given a set of N objects. Each object O has a specified weight and a specified value. Given a capacity, which is the maximum total weight of the knapsack, and a quota, which is the minimum total value that one wants to get. The 0–1 knapsack decision problem consists in finding a subset of the objects whose total weight is at most equal to the capacity and whose total value is at least equal to the specified quota.

2.4.2.4 The P vs NP Question

The question of whether P = NP is widely recognized as one of the most important unresolved problems in the theory of computational complexity. The answer to this question has profound implications for our understanding of effective algorithms and problem-solving approaches. Intuitively, class P (problems solved by deterministic algorithms in polynomial time) is known to be a subset of NP (problems solvable by nondeterministic algorithms in polynomial time). This relationship (P NP) is well established (Fig. 1.6). However, the P NP critical hypothesis, which states that there are problems in NP that are intrinsically more difficult than those in P, remains an open question.

2.4.3 Reducibility and NP-Completeness

A key concept in the theory of complexity is the reducibility of polynomial time. A decision problem A is said to be polynomial reduced to another problem B if there is a polynomic time function that transforms any input instance for A (called IA) into a corresponding instance of B. (denoted IB). The result (positive or negative) of A on IA must strictly match the result of B on IB. A decision problem A is classified as NP-complete if it belongs to class NP (solvable by a non-deterministic polynomial-time algorithm) and also has the property that all other NP problems are polynomial reducible to A. Figure 1.6 visually represents the relationships between P, NP and NP-complete problems.

2.4.4 The Meaning of NP-Completeness

The meaning of NP completeness lies in the following: if a polynomial-time determinist algorithm is discovered for even one single NP-complete problem, then all NP problems can be solved effectively (in polynomial time). This highlights the importance of NP completeness in identifying intrinsically difficult problems within the wider NP class.

2.4.5 NP-Hard Problems

NP-hard problems are a subclass of optimization problems whose decision equivalents (problems asking whether a solution exists with specific properties) are complete in NP. The vast majority of real-world optimization problems fall into the NP-hard category. For these problems, it is highly unlikely that proven effective algorithms (with the complexity of polynomial time) exist. To solve them optimally, it often takes an exponential time (except P = NP). In the absence of precise and effective algorithms, metaheuristics provide valuable alternative approaches to solving these computationally challenging optimization problems.

2.4.6 Example NP-hard problem

Sequencing and scheduling problems such as flow-shop scheduling, job-shop scheduling, or open-shop scheduling.

2.5 Optimization Methods

This section looks into the area of optimization issues. It explores various optimization strategies used to identify either the optimal solution or a solution that is close to the optimal, while minimizing the computational effort.

In the field of operational research, combinatorial optimization problems present a unique challenge. Although many solution methodologies exist, they can be widely classified into two primary classes: exact methods and approximate methods. Exact methods offer the advantage of guaranteeing an optimal solution. However, their complexity can be prohibitive, often resulting in excessively long execution times. In contrast, approximate methods give priority to efficiency,

often producing high-quality solutions within a reasonable timeframe. However, they lack the guarantee of optimum combined with accurate methods. This compromise between solution quality and the complexity is a central consideration when choosing an appropriate solution method for combinatorial optimization problems.

2.5.1 Exact methods

Exact methods work by exploring the entire research space, either explicitly or implicitly, in order to identify the optimal solution. One of the main advantages of these methods is their ability to ensure that the solution they find is indeed the best. Their cost of calculation increases considerably with the increase in the size of the problem and the optimization of the number of targets. As a result, the practical application of exact methods is often limited to problems of a smaller scale.

Exact optimization methods are the preferred approach when it comes to problems where the cost function exposes a polynomial growth relative to the size of the problem. These problems belong to class P of optimization problems. However, for problems classified as NP-hard, accurate optimization methods require exponential complexity of time. This can make them impractical for even medium-sized problems, making them untreatable.

2.5.2 The branch and bound algorithm

Branch and bound (B&B) is a general-purpose optimization algorithm used to identify optimal solutions to combinatorial optimization problems. It works by systematically exploring the research space, which encompasses all possible solutions. This exploration involves the iterative division of the research space into smaller subproblems, each of which is resolved recursively. A key feature of the B&B is the maintenance of a lower threshold on the value of the optimal solution encountered so far. By taking advantage of this lower limit, the algorithm can strategically cut portions of the search space that are guaranteed not to contain solutions that exceed the current best solution. This cut-off mechanism significantly reduces the overall research effort. Because of its effectiveness, B&B has become an accurate method widely used to deal with combinatorial optimization problems in various areas, including operational research, computer science and engineering. B&B methods solve a discrete optimization problem by breaking up its feasible set

into successively smaller subsets, calculating bounds on the objective function, and using them to discard certain subsets from further consideration. The procedure ends when there is no better solution than the existing solution [12].

2.5.3 Dynamic programming

The origins of dynamic programming, like many models and methods of operational research, go back to World War II. During this period, American mathematician Abraham Wald developed a new approach called "sequential analysis" to improve quality control procedures in the production of ammunition [13].

Dynamic programming (DP) is a problem-solving paradigm that is suited to optimization problems. It works by systematically breaking up a complex problem into a series of smaller, overlapping sub-problems. A definite feature of DP is that each subproblem is solved only once, and that solutions are strategically stored (often in a table or using memorization) to prevent redundant calculations. This sub-problem solution storage strategy allows iterative construction of the overall solution, taking advantage of optimum solutions pre-calculated for smaller components [14]. The applicability of DP extends across a variety of fields, including computer science, operations research, economics, and bioinformatics [15].

Unlike branching algorithms, dynamic programming plays a crucial role in polynomial and exponential time methods. The basic principle underlying dynamic programming is to start by solving minor or trivial sub-problems and gradually address larger and more complex problems. This approach is to construct solutions for the larger sub-problems by taking advantage of the solutions obtained for the smaller ones. This is in contrast to the ramification algorithms, where the emphasis is on decomposing the problem. The selection of subproblems is often critical to the effectiveness of dynamic programming [16].

2.5.4 Approximation methods

A major challenge in computer optimization lies in the intrinsic complexity of real-world problems. Most of these problems are classified as NP-hard, which implies that finding the optimal, or optimal, solution for large-scale cases may often require IT resources exceeding current capabilities

[17]. A key subdomain of optimization research focuses on analyzing the computational requirements of solution methods relative to the size of the problem instance. Although NP-hard problems may not guarantee the effective discovery of optimal solutions, they can still facilitate the identification of solutions close to optimum. As a consequence, the design of approximation algorithms shares some high-level similarities with the conception of exact algorithms. Both approaches involve identifying appropriate problem structures and developing algorithmic techniques to take advantage of these structures. However, approximation algorithms often require more complex problem structures and algorithm techniques, often built on generalizations and extensions of powerful tools discovered in the study of exact algorithms.
To meet the challenge of finding near-optimal solutions to NP-hard problems, a more theoretical approach known as approximation algorithms emerged. These algorithms aim to provide high-quality solutions wheelieing efficient computing complexity (polynomial time). While achieving feasibility, or simply finding a valid solution, is usually not a major obstacle, the key aspect of approximation algorithms lies in their ability to guarantee a certain level of solution quality. This quality is measured by the maximum distance between the solutions produced by the approximation algorithm and the optimal solutions in all possible problem cases. In essence, it is said that an approximation algorithm approximately solves an optimization problem if it consistently provides feasible solutions that are close to the optimal value.

Among the approximate methods for optimization problems, there are two main sub-categories: approximation algorithms and metaheuristic algorithms. Heuristic algorithms give priority to finding adequate quality solutions within a reasonable time frame. On the other hand, approximation algorithms are designed to identify solutions that have a guaranteed level of quality, especially for cases of large-scale problems.

2.5.5 Approximation algorithms

It represents a class of iterative optimization algorithms that gradually build a solution in a step-by-step manner. These methods start with an initially empty partial solution and iteratively seek to extend it by incorporating additional elements at each step. This process of extending the partial solution continues until a complete solution to the problem is obtained.
The applicability of constructive methods is often appropriate for scenarios where the focus is not

primarily on obtaining the absolute optimal solution, or where examples are of reasonable size. They are often used to generate an initial solution in a metaheuristic framework. Their main advantages lie in their efficiency and ease of deployment.

The Jacobian technique: The Jacobi method is an iterative approach used in numerical linear algebra to approximate the solution of a system of n linear equations in n variables. It is effective for dominant diagonal systems. The method iteratively updates the solution vector by calculating new values for each variable based on the current estimates of the other variables. This process continues until a convergence criterion is met. In particular, the Jacobi method has a historical significance, as it was initially introduced as the process of Jacobi transformation for matrix diagonalization. In addition, it is sometimes called the simultaneous displacement method.

The Gauss-Seidel: an iterative technique for solving linear equation systems, is well suited for addressing systems of the shape $Ax = b$, where A is a square matrix of the order n and x and b are vectors of Rn. The method involves a series of manipulations: first, A is broken down into $D - L - U$, where D is a diagonal matrix, - L is a lower triangular matrix (L for the lowest), and - U is a upper triangle matter. (U for Upper). Subsequently, the system is transformed into an iterative pattern, where x values are updated iteratively until convergence is achieved. We can then transform the system into:

Sure, here's the entire transformation written in one line:

$$Ax = b \Leftrightarrow (D - L)\,x - Ux = b \Leftrightarrow x = (D - L)^{-1}\,Ux + (D - L)^{-1}b$$

The successive over-relaxation (SOR): The method of successive over-relaxation (SOR) appears to have emerged in the 1930s, with its initial mention found in [18]. However, the formal theoretical foundation of SOR was established almost simultaneously by Frankel [19] and Young [20]. A central focus in the development of SOR theory is the identification of complex numerical values ($\omega \in C \setminus \{0\}$) for which the method converges. All these values define the region of convergence. Furthermore, if possible, the theory seeks to determine the optimal value (ω b) that gives an asymptotic optimal convergence, meaning min $\omega \in C\{0\}\,\lambda(L\ \omega\ b) = \lambda(\,L!)$, where λ denotes the spectral radius. Finding convergence regions is usually less difficult than determining ω b.

However, in both cases, some knowledge on the spectrum of the associated Jacobi iteration matrix (J; σ(J)) is assumed to be available. This information is mainly derived from matrix A properties (and the chosen partitioning scheme). Significantly, Kahan [21] established a SOR method property that is independent of matrix A properties, with the exception of those that define the same method. This property is presented below.

2.5.6 Metaheuristics methods

The term "metaheuristic" comes from the Greek, combining "heuriskein" (which means "to be found") and the prefix "meta" (meaning "beyond" or "at a higher level"). Metaheuristics are a class of problem-solving methodologies that take advantage of natural phenomena. They are specifically designed for application to optimization problems, where the main objective is to identify the overall optimum, which can often be obscured by the presence of many local optima. Metaheuristic algorithms can be widely classified into two main categories: neighborhood search methods and evolutionary algorithms.

2.5.7 Neighborhood Methods

Neighbourhood search methods are iterative optimization techniques that start with an initial solution. This solution can be obtained either by exact methods or by random generation. The basic principle of these methods is to explore the solution space by gradually moving away from the initial solution. This exploration follows a defined path, leading to the identification of improved solutions in the research space. Neighbourhood research methods include a diverse set of algorithms, two of the most commonly used approaches:

- Simulated annealing
- Tabu search

2.5.8 Simulated annealing

The simulated annealing algorithm (SA), initially proposed by Metropolis et al. (1953), is inspired by the physical process of annulment in metallurgy. During the cancellation, a material is heated to a high temperature, allowing its atoms to reorganize freely. As the temperature is gradually

lowered, atoms are increasingly likely to settle in a low-cost configuration, eventually reaching a state. Similarly, SA seeks to find the optimal solution for an optimization problem. The cost function, also called the objective function, plays a crucial role in SA [22]. It guides the research towards cost-effective configurations. Kirkpatrick et al. (1983) introduced the concept of temperature as a control parameter in SA. The algorithm iteratively generates new configurations (solutiEon) from an initial solution (S0) and a high temperature (T0). As temperatures gradually decrease, the likelihood of accepting worse (more expensive) solutions decreases, eventually leading to the search for the overall optimum, which represents the state of equilibrium at zero temperatures.

2.5.9 Tabu search

Tabu Search (TS) is a meta-hero optimization technique that guides a local search process to explore the solution space beyond the local optima. This exploration capability is mainly motivated by its use of adaptive memory, which allows for more flexible search behavior compared to deterministic methods. Memory-based strategies are therefore a definitive feature of the TS approaches, in line with the concept of "integration principles" where various memory structures are combined with effective strategies for their exploitation [23].

2.5.10 Evolutionary algorithms

Evolutionary algorithms (EAs) are a class of probabilistic optimization algorithms inspired by the principles of natural evolution. They deal with complex problems through an iterative process of perfecting a population of candidate solutions. At the core, the EAs maintain a population of individuals representing potential solutions. These individuals undergo evolution-inspired operations, including selection, crossover, and mutation, which mimic the processes of natural selection and genetic recombination observed in biological systems. Three of the most commonly used algorithms:

- Genetic algorithm
- Ant colony
- Biogeography-based optimization

2.5.11 Genetic algorithms

(GA) are a class of computational optimization techniques inspired by the principles of natural selection and genetics. They deal with complex problems by imitating the evolutionary process, iteratively improving a population of potential solutions. These algorithms work on a set of candidate solutions, encoded as binary number strings or other data structures, and use selection, cross and mutation operators to explore the solution space [24]. Due to their population-based approach and their ability to escape local optima, Gas have become one of the most widely used evolutionary algorithms to solve challenging optimization problems in various scientific and engineering fields.

2.5.12 Ant colony

The Amphibious Colony Optimization (ACO) algorithm, a multi-agent approach inspired by the feeding behavior of real ant colonies, has proved effective in solving various discrete optimization problems [25]. In ACO, artificial ants explore the research space, initially randomly, in search of optimal solutions. Once a source of food (solution) is found, ants lay down traces of pheromones along their way back to the nest. The amount of pheromone deposited is usually correlated with the quality (e.g., physical condition) of the solution [26]. Traces of pheromones evaporate over time, promoting the exploration of alternative routes. However, the remaining pheromone signals guide the later ants, favoring pathways with higher initial concentrations. This positive feedback mechanism gradually leads the colony to converge towards shorter or better-quality solutions.

2.5.13 Biogeography-based optimization

(BBO) is a population-based metaheuristic algorithm inspired by the principles of the island's biogeography. In BBO, each candidate solution is represented as a "habitat", and its quality is evaluated using a "Habitat Suitability Index" (HSI), similar to the physical conditioning function in traditional scalable algorithms. Each component of a solution is called a Suitable Variable Index (SIV) variable, reflecting the factors that influence the suitability of a habitat in a natural ecosystem [27]. These factors, similar to environmental variables in real-world islands, may include aspects such as precipitation, land surface, plant diversity, variety adaptation (uncertain concept,

clarification of needs) and climate. High HSI values indicate well-adapted habitats, while low HSI value indicates less favourable environments. BBO imitates the process of species migration by employing two main operators: migration and mutation [28]. These operators guide research towards optimal solutions within the problem space, similar to how species explore and adapt to appropriate islands in a natural environment.

3 Conclusion

Global optimization, the process of identifying the absolute optimal solution within a set of feasible alternatives, represents a major challenge in the field of operational research. (OR). This problem has been extensively studied for decades due to its frequent application in various areas. In OR, optimization refers to the selection of the "ideal" solution from a set of candidate solutions that maximizes (or minimizes) a given objective function.

This chapter provided a fundamental understanding of the optimization problem model and its main characteristics. We explored different classification patterns for optimization problems, highlighting the prevalence of restricted problems in real-world scenarios. Contrary to the theoretical emphasis on unconstrained problems, most practical applications involve constraints that limit the space of feasible solution. The presence of equality and inequality constraints introduces additional complexities in the implementation of optimization algorithms [5]. As recommended practice, limited optimization problems can often be transformed into unlimited problems using established constraints management techniques.

Finally, the chapter provides an overview of two leading approaches to optimization solutions: exact methods and approximation algorithms. Exact methods offer the advantage of guaranteeing the identification of the optimal solution, but their applicability is limited to problems with polynomial time complexity (P problems). For computably difficult and large-scale problems classified as non-deterministic polynomial problems (NP-hard), the exact methods become ineffective. In such scenarios, approximation algorithms, including metaheuristic, become more appropriate by providing almost optimal solutions within a reasonable timeframe.

CHAPTER 2 : Artificial Intelligence

1 Introduction:

Artificial Intelligence (AI) is a field of computer science where experts build intelligent machines that can take in data, process it, and then take actions based on their findings [29]. Due to its ability to learn and reason about complex problems, artificial intelligence (AI) has become a powerful tool widely used in computer science and operational research [30].

The basic aim of artificial intelligence (AI) is to mimic human brain intelligence. This enables machines to recognize and apply the appropriate piece of "knowledge" at a given stage of problem-solving. AI uses several technologies to let machines mimic human intelligence.

Artificial intelligence is influenced by a wide range of fields, including mathematics, biology, philosophy, psychology, neuroscience (which studies the human mind and its behavior), statistics (which handles large amounts of data), and computer science (which runs the algorithms to put the concepts into practice) [29]. All of them have already adopted some form of artificial intelligence. Accordingly, AI can be classified into three main types:

- Narrow artificial Intelligence Weak AI: NAI is a goal-oriented tool used to tackle specific problems in specific situations. While it can outperform people in data processing, it cannot tackle problems outside of its scope [31].
- General artificial Intelligence: a machine intelligence that capable of performing general human intelligent behaviors with a full spectrum of human cognitive capacities [31].
- Strong artificial intelligence: This type of AI aims to outperform human capabilities. While challenging, it is conceivable for it to outperform humans in certain tasks. It's possible that machines will eventually surpass humans and may represent a threat to humanity [29].

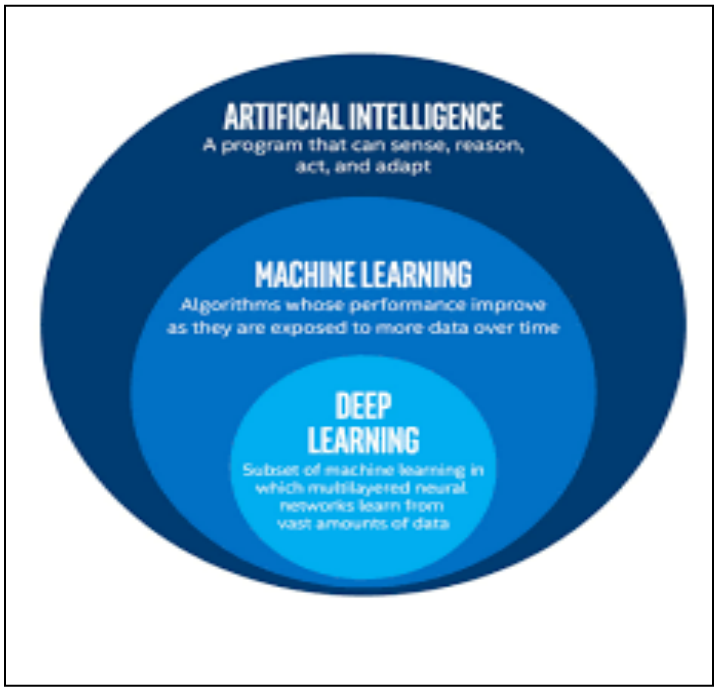


Figure II- 1: Hierarchical Architecture of Artificial Intelligence

Artificial Intelligence consists of various subdivisions, the most famous of which are machine learning and deep learning

Machine Learning (ML): Explain how ML enables machines to learn and develop without requiring explicit programming. Discuss the different learning paradigms:

- Supervised learning
- Unsupervised learning
- Reinforcement learning

2. Machine learning

2.1. Definition

Machine learning (ML) is a subset of computer science and artificial intelligence that explores how computers can replicate human learning activities. This approach assumes computers can learn from data, spot patterns, and form conclusions without being explicitly programmed [32]. The machine analyzes the available data set, also known as training data, and uses algorithms to predict possible outputs for a given input [29].

2.1.1 Types of machine learning

Machine learning algorithms are trained using labeled, unlabeled, or hybrid datasets. Three types of machine learning have been identified: supervised, unsupervised, and Reinforcement learning. Understanding how they affect input and output variables is essential [33].

2.1.2 Supervised learning

This type of ML requires supervision, in which machines are trained on labeled data to educate a system that can predict outcomes based on its training. The model evaluates the link between input and output data to identify underlying patterns [33]. In this context, input variables (X) and an output variable (Y) are given. The objective is to select an algorithm capable of learning the mapping function from the inputs to the output, denoted as $Y = f(x)$. This function is derived from a dataset consisting of a collection of labeled examples $\{(x_i, y_i)\}N_{i=1}^{x}$. Each x_i among the N examples is referred to as a feature vector. A feature vector is a vector in which each dimension $i = 1, \ldots, D$ contains a value that describes the sample in some way. After the training and processing are done, we test the model with sample data to see if it can accurately predict the output.

Supervised machine learning is divided into two broad categories:

- Classification: Classification is an SL approach that divides data into different classes. It's a recursive technique recognizes and categorizes data objects using predefined labels. In simple terms, it assigns each new observation to an appropriate set or class. The output variable can belong to multiple classes

[34][35]. The purpose is to learn a mapping from inputs x to outputs y, where $y \in \{1, \ldots, C\}$, where C is the number of classes. If C = 2, this is known as binary classification. If $C > 2$, this is called multiclass classification [36].

For example, determining whether an email is spam or not is a popular use case of classification.

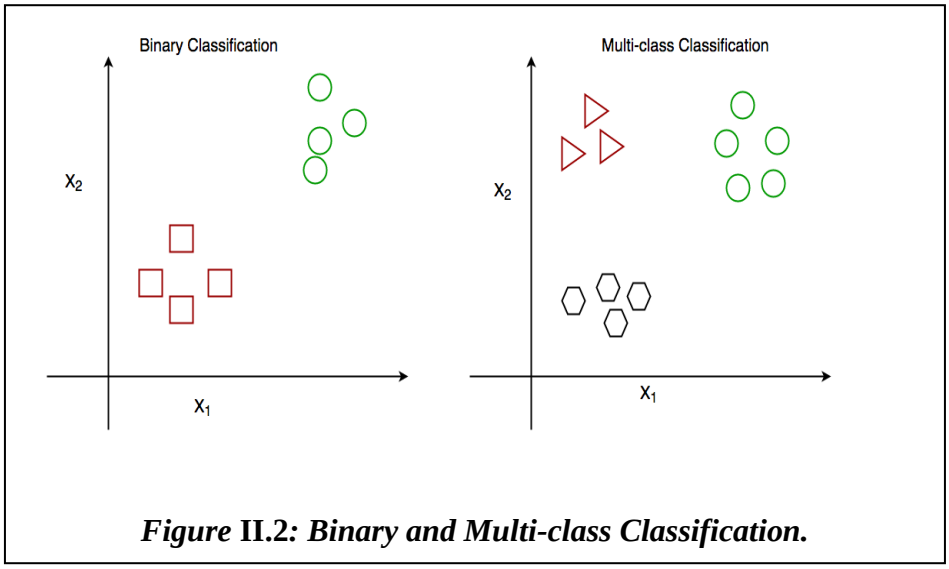


Figure* II.2*: Binary and Multi-class Classification.

- Regression: Regression is a statistical technique that examines the connection between a dependent variable and one or more independent variables, regression is similar to classification, only the response variable is continuous [34]. Common algorithms for supervised learning include regression analysis (i.e. linear regression, logistic regression, non-linear regression) [33].

  Figure II.3 presents a simple example with a single real-valued input x_i and response y_i in R.

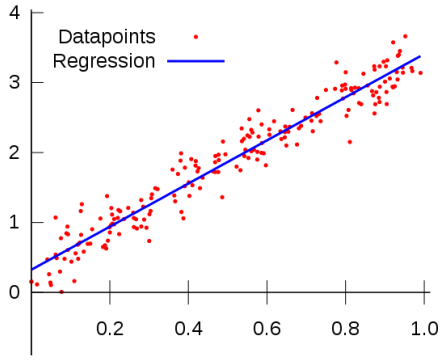


Figure II- 3: Linear regression on some 1d data.

2.1.3 Unsupervised learning

Unsupervised learning involves unlabeled output variables and unknown input-output pairings. In this method, a machine learning model searches for differences, patterns, similarities, and structures in data. Unsupervised learning methods are frequently classified into two categories: clustering and association analysis [33][34]. Unsupervised learning allows for the exploration of previously unnoticed data patterns. Unsupervised learning is perhaps more similar to human and animal learning. This method is more widely applicable than supervised learning as it doesn't require human expertise [36].

- **Clustering:** Clustering is a way of organizing items into groups so that the objects with the most similarities stay in one group and have few or no similarities with the objects in another [38]. It seems that clustering and classification are similar but in classification, There are predetermined labels provided to each input while these labels are not present in clustering.
- **Association:** An association rule is an unsupervised learning strategy used to identify links between variables in a big database. It identifies the group of items that appear together in the dataset [38].

2.1.4 Reinforcement Learning

Reinforcement Learning (RL) is a unique form of machine learning that enables machines and software agents to determine optimal behavior in order to achieve desired outcomes [37]. Opposed to previous approaches, in RL the data is not predetermined; instead, it is provided to the agent as it investigates the environment. The approach uses a reward mechanism to guide the agent's learning based on previous knowledge, rather than being fully random [39].

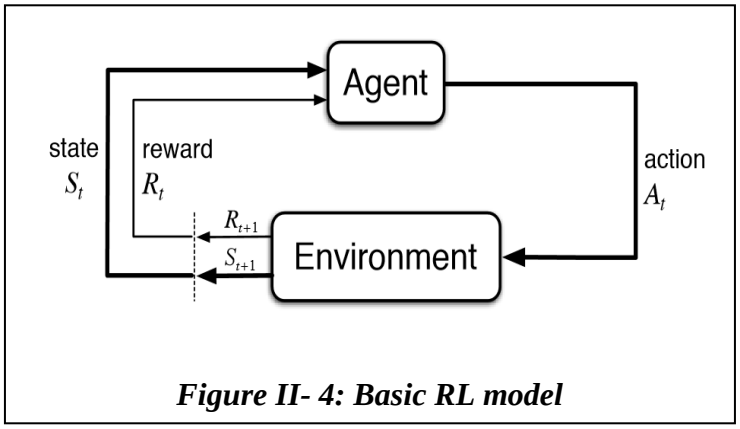


Figure II- 4: Basic RL model

2.2 Machine learning algorithms

2.2.1 Supervised learning algorithms

There are several SL algorithms, each with unique approaches. The algorithm used depends on the dataset and usage.

- **K-Nearest Neighbors (KNN)**

KNN categorizes data points based on a simple majority vote among their closest neighbors. Its precision is dependent on the quality of the data and it is resistant to noise. The biggest issue in using KNN is determining the proper number of neighbors. KNN is commonly used for statistical estimate and pattern identification based on proximity between objects [40][41].

- Decision tree:

Decision trees (DT), also known as classification trees, are a common and successful predictive modeling approach used in statistics and data mining. They are among the first and most recognized SL algorithms, making them ideal for classification problems. These are graphical representations of a problem or choice based on certain criteria, including internal and exterior nodes linked by branches [42].

2.2.2 Unsupervised learning algorithms

Unsupervised learning algorithms are better suited for more complex problems, Below we describe some examples.

- K-means algorithm:

The K-means algorithm is simple to comprehend. K-means' complaint is that it splits a given set of data into K groups and then assigns the matching center to each sample. There are four phases [44].

- Data processing.
- Select K centers.
- Defining the loss function.
- calculates the distances between each data point and each of the k centroids. Next, it uses the mean of the corresponding data points from each cluster to update each centroid's position.

- Hierarchical clustering:

hierarchical clustering is applied to create a cluster out of the unlabeled datasets. This algorithm creates a tree-like structure called a dendrogram, which is the hierarchy of clusters assembled into a tree [38]. It looks similar with k-means however we do not have to set the fixed K value in the hierarchical clustering [44].

2.2.3 Reinforcement learning algorithms

reinforcement learning provides a variety of approaches, based on the nature of assignment and the particulars of the environment.

3. Deep learning

3.1. Definition

Deep learning is a subset of machine learning in artificial intelligence (AI) based on artificial neural networks. DL models high-level abstractions in data through non-linear transformations [45].

Deep learning models, often known as deep neural networks, are inspired by the structure and operation of the human brain. They are made up of several layers of interconnected nodes (neurons), with each layer transforming the input data into more abstract representations. To decrease prediction errors, the network is trained using huge datasets and optimization tools like as backpropagation [46][47].

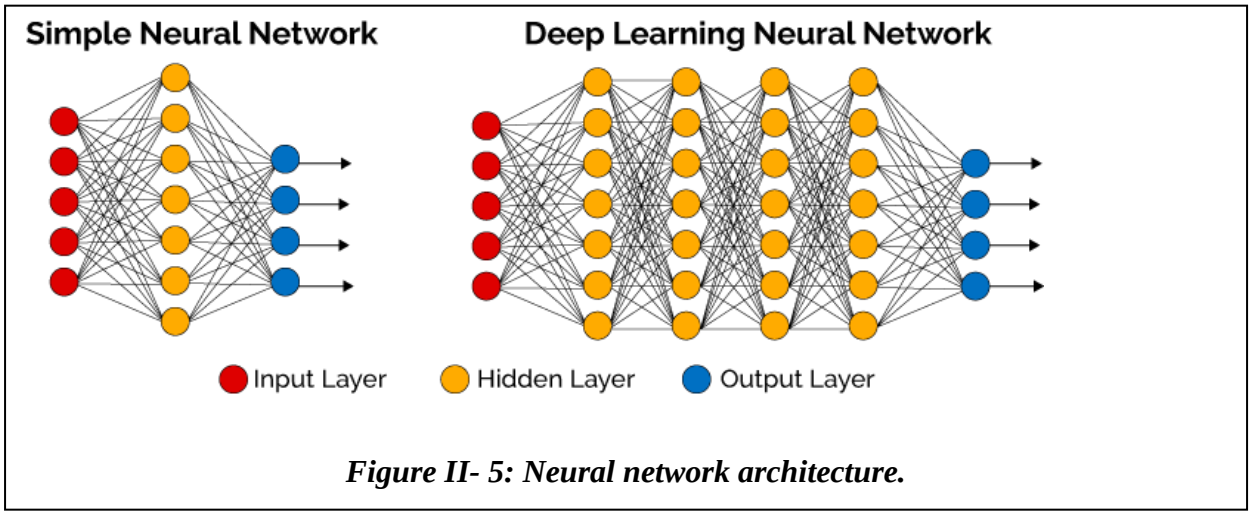


Figure II- 5: Neural network architecture.

3.2. Deep learning models

3.2.1 Convolutional Neural Networks (CNN)

Convolutional Neural Networks (CNNs) are a type of deep learning model that is specifically built to handle structured grid data like photographs. They use convolutional layers and filters to automatically and adaptively learn spatial feature hierarchies from input photos [46].

CNN is a useful method for recognizing visual patterns in pixel images with minimal preprocessing. In 2012, Alex Krizhevsky developed Alex Net, a CNN that reduced error from 26% to 16% in the ImageNet Large Scale Visual Recognition Challenge [20].

3.2.2 Recurrent Neural Network (RNN)

Recurrent Neural Networks (RNNs) are a type of neural network that recognizes patterns in sequences of data, such as time series or natural language. They have connections that form directed cycles, allowing data to survive [46].

In the context of sequential data, RNN can be very powerful for the prediction process compared to other networks because it uses feedback loops in processing data. The advantage of RNN is that it considers what is encountered before to process current information [46].

3.3. Application of deep learning

- Healthcare: in the field of medicine doctors use deep learning to classify photos of skin lesions as benign or malignant skin malignancies achieves the same accuracy as board-certified dermatologists [49].

- Auto Driving: Deep learning has been used to improve vehicle safety by automatically identifying the road, traffic signs, pedestrians, and providing necessary advice. This can help improve traffic regulation and reduce accidents [50].

- Natural Language Processing (NLP): Deep learning has transformed processes like machine translation, which includes translating text from one language to another, and sentiment analysis, which identifies the sentiment expressed in text. Text summarization creates brief summaries of lengthier texts, whereas speech recognition translates spoken language to text. Siri, Alexa, and Google Assistant are examples of chatbots and virtual assistants that use deep learning [51].

CHAPTER 3: Traveling tournament problem (TTP)

1 Introduction

The Traveling Tournament Problem (TTP) was introduced by Easton, Nemhauser, and Trick in 1993, inspired by Trick's work on scheduling Major League Baseball (MLB) in the United States. Since 1996, Trick and baseball executive Doug Bureman have collaborated on optimizing MLB schedules, demonstrating the practical importance and impact of this problem. The TTP's popularity surged as other sports leagues recognized the potential for significant operational cost reductions through mathematically generated schedules.

TTP represents a global challenge that sports leagues face in their quest to schedule games that are fair and effective for all clubs. This combinatorial optimization problem seeks to find the most cost-effective tournament itinerary for a group of teams, aiming to meet specific constraints while minimizing the total distance travelled by all participating teams. In a double round-robin competition with n teams, each team plays two games against each of the other $n-1$ teams, resulting in a complex scheduling challenge.

The total travel distance between venues is a key consideration in tournament scheduling, with significant real-world implications. Efficient scheduling can lead to substantial cost savings, reduced travel fatigue for players, and improved overall fairness in the competition. This chapter explores different TTP variants, assesses their complexity, and introduces models and formulations designed to minimize travel costs and ensure fair tournament scheduling.

We begin by presenting the TTP in its original form, followed by a discussion of existing solution methods. Subsequently, we explore various problem variants and advancements made in addressing these challenges. Finally, we delve into intriguing related problems that have emerged from the study of the TTP, highlighting ongoing research and future directions.

2 The Traveling Tournament Problem

In a round-robin tournament involving n teams, where n is an even number, each team competes against every other team exactly once. An m -round-robin tournament extends this concept by requiring each team to play every other team m times. Each match is designated to be played at one team's home venue, making them the home team, while their opponent is the away team.

A sequence of consecutive away games is referred to as a road trip, while consecutive home games are known as a home stand. At the start of the tournament, all teams are located at their home venues and must return to their home venues at the end of the tournament.

The distances between all team venues are provided in an $n \times n$ matrix D. When a team plays an away game, they travel from their home city to their opponent's city. During consecutive away games, the team travels directly from one opponent's city to the next. Upon concluding their road trip, the team returns to their home city.

Note that the length of a road trip (or home stand) is given by the number of opponents played and not the distance travelled by the team.

The Traveling Tournament Problem can be succinctly captured as follows:

Input: An even number of teams, n; an n by an integral distance matrix, D; L, U integer parameters.

Output: A double round-robin tournament on the n teams such that:

- every team plays every other team once at their home venue and once at the venue of the opponent
- the length of every home stand and road trip is between L and U, and
- the total distance travelled by the teams is minimized.

A schedule satisfying the first constraint is a feasible double round-robin tournament. The second constraint is called the at-most constraint. The final constraint ensures optimality.

There may be an additional constraint placed on the tournament:

- No repeaters: there are no teams i, j such that in the tournament schedule, i plays at j, followed immediately by j plays at i.

Note that for $L = 1$ and $U = n - 1$ a team may take a road trip visiting all other teams, which equates to finding a traveling salesman tour. On the other hand, for small U, teams must return home often thus the distance travelled will increase. The TTP was originally introduced with $U = 3$ but we will explore the problem for different values. We rename the parameter k and the version of the problem being discussed is distinguished by using the notation TTP(k). Over the years, a growing number of variants and instances of TTP were introduced. Here are several classes of instances which have become popular benchmarks:

The circle instance is one where n cities correspond to the nodes of a circle graph of size n. Nodes i and $i + 1$, and nodes $n - 1$ and 0 are joined by an edge of length 1. Then the shortest distance from i to j $(i > j)$ is the minimum of $i - j$ and $j - i + n$. A circle instance with n teams is denoted $CIRCn$.

The National League instance with n teams is denoted NLn, and is given by n teams of the Major League Baseball league and distances are given by the "air distance" from city centers. Optimal solutions to NL4 and NL6 are given in [52]. NL8 was solved optimally for the first time in [54] using a branch and price algorithm in parallel with constraint programming although the no-repeaters condition was neglected.

The constant distance instance (denoted CONn) was introduced by Ribeiro and Urrutia [55] and the problem is formulated just as we would expect: the distance between all pairs of cities is constant, typically equal to one. Fujiwara, Imahori, Matsui, and Miyashiro [56] proposed a lower bound for the constant distance TTP and devised two approximation algorithms that produce feasible solutions with values close to their lower bound.

A description of these instances as well as the most current progress on them can be found at the web-page maintained by Trick [57].

The constraints:

The TTP is the problem of finding a feasible schedule that minimizes the distance traveled by the teams, and satisfies the following constraints [66]:

Double round robin constraint (DRRT): that means that each team plays with every other team exactly twice, once in its home and once in the home of its opponent.

- AtMost constraint: each team must play no more than U and no less than L consecutive games at home or away. Specially, in our case, L is set to 1 and U to 3.
- NoRepeat constraint: A game $ti - tj$ can never be followed in the next round by the game $ti - tj$.

1. The TTP contains the number of teams (denoted n) and the distance matrix (denoted Dis).

2. The output are: a double round robin tournament on the n teams respecting the three constraints AtMost, NoRepeat and DRRT, and where the total distance traveled by the teams is minimized.

Table 1 gives an example of a schedule for n = 4 teams. The negation sign means that the team plays away.

Round Team	Round 1	Round 2	Round 3	Round 4	Round 5	Round 6
T1	3	2	4	-3	-2	-4
T2	-4	-1	-3	4	1	3
T3	-1	4	2	1	-4	-2
T4	2	-3	-1	-2	3	1

Table 1. Example of double round robin tournament with n = 4

This schedule specifies that the team t2 has the following schedule: it successively plays against t3 at home, t4 away, t1 away and t3 away, teams t4 at home, t1 at home.

The travel cost of team t1 is: $Dis24 + Dis41 + Dis13 + Dis32.$

We note that the travel costs of a schedule S is the sum of the travel cost of every team (denoted Travel-cost(S)).

2.1 Unconstrained TTP

The unconstrained TTP (uTTP) is formulated identically to the original TTP except we do not enforce the at-most constraint. Thus, we do not impose a bound on the maximum length of home stands and road trips. In this formulation, a team can take an arbitrarily long road trip, possibly visiting all other teams, before returning home. The unconstrained version of the problem was shown to be NP-hard [58], and when the number of consecutive home/away games is fixed to three, the problem is strongly NP-complete [59]. NP-completeness was later shown for all fixed values of k > 3 [60].

2.2 Mirrored TTP

The mirrored TTP (mTTP) is a widely studied variation of the problem. The first paper on mTTP is due to Ribeiro and Urrutia [61]. In this version, the first half of the season's schedule is identical to the second half, except the venues are reversed. If team A plays in city B on the first day of the half of the season, team B would play in city A on the first day of the second half of the season. The sports scheduling community is interested in this form of the problem due to its prevalence in Latin American tournament structures. Cheung [62] used a two-phase method based on generating timetables from 1-factorizations

and finding optimal home/away assignments to solve NL8 and CIRC8 to optimality. Moreover,

Cheung used a Benders approach to compute lower bounds for the mirrored TTP [63].

These bounds remain the current best-known lower bounds for the mirrored problem. Incidentally, several key results for the mirrored TTP were not intended to specifically solve the mirrored problem. The fact that the resulting schedule was mirrored was simply a side effect of the technology employed to create it. Several significant findings on $mTTP$ are described in subsequent sections.

2.3 Approximation algorithms

The Traveling Tournament Problem (TTP) was initially introduced with a constraint k on the number of consecutive home or away games a team can play. Influenced by Major League Baseball, the value of k was set to 3. However, research has also explored the problem for other values of k. The following presents the best current approximations for different values of k, assuming the distances between team venues satisfy the triangle inequality, thus making the distances metric.

Imahori, Matsui, and Miyashiro [67] proposed a constant factor approximation algorithm for the unconstrained TTP (i.e., k=n−1, where n is the number of teams). The solution returned by the algorithm is a mirrored schedule, making it a feasible solution to the unconstrained $mTTP$.

There is no feasible schedule when $k = 1$ [68].

Campbell and Chen [69] studied the problem for $k = 2$ in 1976, predating the formal definition of the TTP. Since then, numerous attempts have been made to achieve a good approximation ratio, but the result depends on the parity of $n/2$. Thielen and Westphal [70] devised an algorithm for each case and found a bound of $3/2 + O(1)/n$ for $n/2$ odd and $1 + O(1)/n$ for $n/2$ even. Building on this method, Imahori [71] proved a $1 + O(1)/n$ bound when $n/2$ is odd. Zhao and Xiao [72] also achieved a $1 + O(1)/n$ ratio for $n/2$ odd, which slightly improved the ratio ($1 + 12/n$ versus $1 + 24/n$) found by Imahori.

For $n/2$ even, Xiao and Kou [73] provided an approximation algorithm with a ratio of $1 + 1/4n$. Chatterjee and Roy [74] found an approximation ratio of $1 + \lceil log2(n/4) \rceil + 4/(2(n-2))$ yielding slightly better results than Xiao and Kou for $n \leq 32$.

For $k = 3$, the most popular version of the problem, Imahori, Matsui, and Miyashiro [75] proposed a randomized approximation algorithm that yields a feasible solution with an approximation ratio of less than 2+(9/4)/(n−1). They then used a rerandomization technique to find a deterministic approximation algorithm with the same ratio. These algorithms were the first to achieve a constant approximation ratio. This result relies on a new lower bound, which, although not as tight as previously known bounds, is easier to calculate. This new bound implies that any feasible solution

to a $TTP\ (3)$ instance is at most three times the optimal value. In fact, their lower bond yields a trivial approximation ratio of k for any feasible $TTP(k)$ solution.

For $k \geq 4$, Westphal and Noparlik [76] proposed an approximation algorithm producing a solution for any $n \geq 6$. Yamaguchi, Imahori, Matsui, and Miyashiro [77] considered $TTP(k)$ for all fixed $k \geq 3$. For $k \leq 5$, the approximation ratio of the proposed algorithm is bounded by $(2k-1)/k + O(k/n)$, and for $k > 5$, the ratio is bounded by $(5k-7)/(2k) + O(k/n)$. For k=3, the ratio is $5/3 + O(1)/n$. The schedule produced by their algorithm is a mirrored schedule.

The Bipartite Traveling Tournament Problem ($BTTP$) was introduced by Hoshino and Kawarabayashi [78], where 2n teams are divided into two groups of n teams. Each team plays one home and one away game against all other teams in the other group. These variant models the scheduling of games for the Nippon Professional Baseball (NPB) league, Japan's largest sports league. Each NPB team plays 24 inter-league games each season, thus the bipartite structure of games between two groups was relevant. The problem is NP-complete. Despite its computational hardness, an optimal schedule for inter-league games was found for the NPB, which consists of only 12 teams. Hoshino and Kawarabayashi [79] developed a polynomial-time algorithm to solve the BTTP with an approximation ratio of $1 + 2c/3 + (3-c)/3n$, where ccc is the approximation factor of the TSP.

2.4 TTP Complexity Analysis

The Traveling Tournament Problem (TTP) is a well-known challenging problem that is thought to be extremely NP-hard. Numerous scholars have examined the difficulty of TTP, and various

techniques have been suggested to address this issue. No research, however, has been able to tackle TTP with more than ten teams in the best way up until this point. The first proof of NP-completeness for a variant of the original TTP was given by Bhattacharyya [64].

Numerous algorithms, including accurate and heuristic methods, have been developed to solve TTP. Exact methods provide the best solution for TTP instances, but they are usually not practical

for large instances due to their high computing cost. Integer programming is a methodical technique to solving TTP that determines the optimal solution using a mathematical model. Conversely, heuristic procedures take a reasonable amount of time to produce a less-than-ideal result. Heuristics like as greedy algorithms, local search algorithms, and evolutionary algorithms have been used to handle TTP scenarios. These methods usually work well and are of high quality, but they do not guarantee optimality.

The size, complexity, and needed level of optimality all play a role in the method that is selected to solve TTP. Smaller TTP instances can be solved using accurate approaches, however larger instances frequently need for heuristic techniques. When choosing an algorithm to solve TTP, it is crucial to take the trade-off between optimality and computing time into account [65].

3 Conclusion

This chapter provided a comprehensive overview of the Traveling Tournament Problem (TTP), its origins, and its significance in the field of sports scheduling. The TTP, inspired by Major League Baseball scheduling, presents a complex combinatorial optimization challenge aimed at minimizing the total travel distance for teams while adhering to specific constraints.

We delved into the fundamental structure of the TTP, discussing the double round-robin format, road trips, and home stands. The problem formulation, including the input parameters and constraints, was thoroughly examined. Various instances and benchmarks of the TTP, such as the circle instance, National League instance, and constant distance instance, were introduced to illustrate the diversity and complexity of real-world scheduling problems.

In summary, the TTP is a critical problem in sports scheduling with significant practical implications. This chapter laid the foundation for understanding the TTP, its variants, and the existing solution methods. The insights gained from this exploration will serve as a basis for developing new and innovative approaches to address the challenges posed by the TTP, ultimately contributing to more efficient and fairer tournament scheduling in the future.

Chapitre IV : BBO

1 History of BBO

Biogeography-Based Optimization (BBO) is a nature-inspired optimization technique that considers the distribution of species across different geographic areas. BBO was first introduced by Dan Simon in 2008 as a novel evolutionary algorithm for solving optimization problems.

2 Biogeography based optimization (BBO)

Mathematical models in biogeography, such as those developed by Mark, describe the migration of species between habitats, the emergence of new organisms, and extinction processes. In the context of BBO, a habitat represents an isolated living space, and its quality is evaluated using a Habitat Suitability Index (HSI). The variables influencing this index, termed Suitability Index Variables (SIVs), can include factors such as rainfall, vegetation diversity, topographic features, land area, and temperature.

The main characteristics of the habitats are as follows:

- Habitats that are favorable to the residence (good geographical conditions) of biological species are called high habitat suitability index (HSI). It is intelligible that the habitat with a high HSI tends to have a large number of species. Similarly, habitats which have bad geographical conditions are called low habitat suitability index.
- High HSI habitats tend to share their features with low HSI habitats by migrating their species to the low HSI habitats. The shared features re- main in the heightens HSI habitats while appearing at the same time as new features in the low HSI habitats.

Figure IV.1 illustrates a linear migration model of species abundance within a single habitat, emphasizing the relationship between species count, immigration rate (λ), and emigration rate (μ). As the number of species in a habitat increases, indicative of a good habitat, the immigration rate

(λ) decreases while the emigration rate (μ) increases. Conversely, when the species count is low, indicative of a habitat with a low Habitat Suitability Index (HSI), the immigration rate (λ) increases and the emigration rate (μ) decreases.

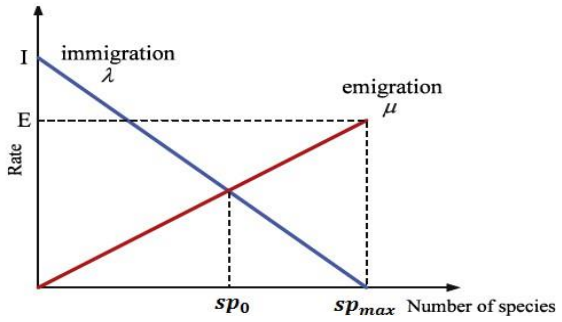


Figure IV.1: Evolution of immigration and emigration rates with the number of species

- A habitat is a candidate solution for the optimization problem.
- The quality of the solution (fitness) is analogous to the HSI.
- The variables defining the solution are SIVs.
- A good solution is analogous to a habitat with a high HSI, and thus with a high number of sp

2.1 BBO implementation

The immigration rate (λ) for each habitat is inversely proportional to the fitness (HSI) value, while the emigration rate is directly proportional to the HSI value (As shown in Fig. 1): The emigration (μ) and the immigration rate (λ) of each habitat is formulated as follows:

$$\lambda_{sp} = I\ (1 - \frac{sp}{sp_{max}})$$

$$\mu_{sp} = E\left(\frac{sp}{sp_{max}}\right)$$

λ_{sp}: is the immigration rate in a habitat with *sp* species.

μ_{sp}: is the emigration rate in a habitat with *sp* species. I: refers to the maximum immigration rate.

E: is the maximum emigration rate.

sp: the number of species in the habitat.

sp_0: is the equilibrium number of species.

sp_{max}: is the maximum number of species that can live in the habitat.

BBO= (τ, ψ) is a 2-tuple:

$\tau = \theta \{Habitatn, HSI\}$ is a function that creates an initial population (set of habitats).

$$\psi = \lambda^n \circ \mu^n \circ \Omega^n HSI^n \circ M^n \circ HSI^n$$

The population transition function initiates by calculating the immigration (λ^n) and emigration (μ^n) rates for each individual using the 2 formulas. One solution, designated as $Habitat_i$ is then selected for modification. The immigration rate of $Habitat_i$ determines whether a Suitable Index Variable (SIV) will be altered. Following this, the emigration rate (μ) is utilized to select a high-quality solution $Habitat_j$, whose SIV will migrate (as outlined in Algorithm 1). The migration process (Ω^n) is subsequently applied between the immigrating $Habitat_i$ and the emigrating $Habitat_j$, wherein superior SIVs from the good solutions ($Habitat_j$) replace those in the poorer solutions ($Habitat_i$) followed by a recalculation of the Habitat Suitability Index (HSI).

Finally, mutation (M^n) is performed, followed by another HSI recalculation for each habitat. In BBO, mutation is analogous to a sudden climatic change within a habitat that may alter its HSI. This is modeled in BBO through a mutation operator, which randomly modifies a habitat's SIV based on the mutation rate.

Algorithm 1: Migration process

```
for i=1: number habitats do
    Select a habitat Habitat_i with probability αλ_i
    if H_i is selected then
        if rand ∈ (0, 1) < λ_i then
            Select Habitat_j with probability αμ_i
            if H_j is selected then
                Habitat_i(SIV) ← Habitat_j (SIV)
            End if
        End if
    End if
end for
```

$$Pmut(sp) = \alpha \frac{1-P_{sp}}{p_{max}}$$

Where:

- $Pmut\,(sp)$: Mutation rate of a habitat with sp species.
- α: Parameter defined by the user.
- P_{sp}: Probability of having sp species in the habitat.
- P_{max}: Probability of having maximum species (max P_S).
- P_{sp}: is the probability of existence of sp species in the habitat.

$$P_{sp} = \begin{cases} \frac{\lambda_0\lambda_1 \dots \lambda_{sp-1}}{\mu_1\mu_2 \dots \mu_{sp}(1+\sum_{l=1}^{n}\frac{\lambda_0\lambda_1 \dots \lambda_{l-1}}{\mu_1\mu_2 \dots \mu_l})} & 1 < sp < number\ habitats \\ \frac{1}{1+\sum_{l=1}^{n}\frac{\lambda_0\lambda_1 \dots \lambda_{l-1}}{\mu_1\mu_2 \dots \mu_l}} \dots sp = 0 \end{cases}$$

Algorithm 2: Mutation

```
for j = 1 to number habitats do
    Use λi and μi to compute Pmuti
    Select Habitati(j) with probability αPmuti
    if Habitati(j) is selected then
        Replace Habitati(j) with a random SIV
    end if

end for
```

3 Conclusion

In this chapter, we have explained the biogeography-based optimization algorithm, First, we discussed the history of BBO and then provided a detailed description of this algorithm. This algorithm is designated by the migration of bad solutions towards excellent solutions. We then addressed the steps of this algorithm, which are construct initial population and migration and mutation

Chapitre V : PROPOSED APPROCH

1. Introduction

In the previous chapter, we introduced and discussed the Tournament Traveling Problem (TTP) and its constraints. In this chapter, we will focus on discussing the proposed approaches to apply machine and deep learning to enhance Stochastic Local Search (SLS) and BBO algorithms performance on solving the TTP. First, we will explore the generation of the initial solution and neighborhood structures that are used in both approaches. Then explain the BBO and its projection on our problem and how we implement a deep learning model for parameters tuning. Additionally, we will discover the application of reinforcement learning as neighborhood structure within the SLS algorithm.

2. BBO for TTP

The Biogeography-Based Optimization (BBO) algorithm is a powerful evolutionary algorithm inspired by the geographical distribution of biological species. It has been successfully applied to a wide range of optimization problems. To solve the scheduling challenge of The Traveling Tournament (TTP), we utilize the BBO algorithm. First, we will investigate the implementation of Neighborhood Structures and several swap methods. Then we will explore the BBO implementation using these methods.

2.1.Initialize schedule

Starting with creating an initial schedule that satisfies the DRRT (Double Round Robin Tournament) constraint. We create this structure using graph theory modeling, as follows:

We number the graph's vertices from 1 to n, where n represents the number of teams. We center the top n vertices and form a circle around them.

On the first day, we plan games between (Team 1 and Team n), (Team 2 and Team n−1), (Team 3 and Team n−2), and so on, until the game between (Team n/2 and Team n/2 + 1).

On the following day, we repeat the previous day's pairings by rotating the team couplings clockwise.

Using this method will give us an initial single round robin schedule. We mirror it to get a schedule that verifies the DRRT constraint. The following tables shows the final output for a tournament with n = 4.

N = 4		Rounds					
		1	2	3	4	5	6
Teams	1	4	3	2	-4	-3	-2
	2	3	-4	-1	-3	4	1
	3	-2	-1	4	2	1	-4
	4	-1	2	-3	1	-2	3

Table V.1: A Double Round Robin Tournament (DRRT) schedule.

To find a feasible configuration that satisfies the remaining constraints specifically AtMost, NoRepeat and DDRT. We utilize a cost function to penalize configurations that break the constraints and neighborhood structures to navigate the search space. Additionally, we propose to use a local search method which is VNS to construct these feasible schedules.

The cost function is combination of two terms. The first term is to penalize constraints violation AtMost and NoRepeat. The second term is about the total traveled distance.

The penalty At-most $f_{At_{most}}(S)$ is the total number of times the teams play more than three consecutive home games or three consecutive away games and the penalty $f_{No\ repeat}(S)$ is the total number of times that the game (ti, tj) is followed immediately by (tj , ti) in the schedule S.

Now, we define the following equations:

$$Constraints\_Cost(S) = f_{At_{Most}}(S) + f_{No_{Repeat}}(S) \quad \text{(V.1)}$$

$$Cost(S) = e^{Constraints\_Cost(S)} \times Distance\_cost(S) \quad \text{(V.2)}$$

2.2. Neighborhood structures

We utilize four neighborhood structures, as stated below:

N1 swap home: The Swap Home move is an operation that swaps the home and away roles of two teams involved in a specific match. This technique helps to explore alternative scheduling configurations while maintaining the tournament's overall structure. Given a schedule S and teams $t_i$and t_j , the Swap Home move is denoted as $SwapHome\ (S,\ t_i, t_j)$. For each round r in the schedule, determine if teams $t_i$and t_j are playing against each other, if so exchange their home\away roles. For example, if we apply $SwapHome\ (S,\ 1,\ 3)$ the results will be

N = 4		Rounds					
		1	2	3	4	5	6
Teams	1	4	-3	2	-4	3	-2
	2	3	-4	-1	-3	4	1
	3	-2	1	4	2	-1	-4
	4	-1	2	-3	1	-2	3

Table V.2: $SwapHome\ (S\ ,1,\ 3)$ results.

N2 swap round: The Swap Round move involves exchanging the positions of two rounds in the schedule. This helps explore different orderings of the rounds while keeping the matchups within each round intact. Given a schedule S and two rounds $r1$ and $r2$, the Swap Round move is denoted as $SwapRound\ (S,\ r_1,\ r_2)$. This technique Swap the positions of all the games scheduled in round $r1$ with those in round $r2$

N = 4		Rounds					
		1	2	3	4	5	6
Teams	1	-4	-3	2	4	3	-2
	2	-3	-4	-1	3	4	1
	3	2	1	4	-2	-1	-4
	4	1	2	-3	-1	-2	3

Table V.2: $SwapRound\ (S, 1,\ 4)$ results.

N3 swap team: The Swap Team move swap the entire schedules of two given teams except the rounds they play against each other's. Given a schedule S and two teams $t_i$and t_j , the Swap Team move is denoted as $SwapTeam\ (S,\ t_i, t_j)$

N = 4		Rounds					
		1	2	3	4	5	6
Teams	1	-2	-3	4	2	3	-4
	2	1	-4	-3	-1	4	3
	3	4	1	2	-4	-1	-2
	4	-3	2	-1	3	-2	1

Table V.3: $SwapTeam\ (S, 2,\ 4)$ results.

N4 swap path: The Swap Path move involves transferring the entire schedule of one team from one schedule to another and apply all the necessary adjustment. This helps explore different scheduling configurations across different schedules while keeping the overall structure of the tournaments intact. Given two schedules $S1$ and $S2$, and a team t_i, the Swap Path move is denoted as $SwapPath(S1,\ S2,\ t_i)$.

N = 4		Rounds					
		1	2	3	4	5	6
Teams	1	-2	4	-3	2	3	-4
	2	1	3	4	-1	-4	-3
	3	-4	-2	1	4	-1	2
	4	3	-1	-2	-3	2	1

Table V.4: Feasible Schedule S1.

N = 4		Rounds					
		1	2	3	4	5	6
Teams	1	-2	4	-3	2	3	-4
	2	1	3	4	-1	-4	-3
	3	-4	-2	1	4	-1	2
	4	3	-1	-2	-3	2	1

Table V.5: $SwapPath\ (S\ ,\ S1,\ 1)$ results.

VNS works on a variety of neighborhoods. In our study, we use three (k = 3) neighborhood structures: N1 (Swap Home), N2 (Swap Round), and N3 (Swap Team). At each iteration, we choose one of the three structures to generate neighbor solutions.

2.3.BBO implementation

2.3.1. Initialize population

We generate an initial population of an n candidate solution where each element (habitat) is a schedule that satisfies the DRRT. To enhance the initial solution, we apply the Variable Neighborhood Search (VNS) algorithm.

```
Algorithm 1: InitializePopulation
Input: populationSize
Output: population
Population ← [ ]
S ← GenerateDRRT(TTPinstance)
For i in populationSize do
    S ← VNS(S)
    Population.append(S)
Return Population
```

2.3.2. Evaluate population

We calculate the fitness, or the total cost of each schedule based on the total travel distance and constraint violations then we sort the population from the worst to the best that means the position of the solution indicate its quality.

2.3.3. Migrate and mutation

In the migration process, we take two solutions from the population based on the migration rate, use the $SwapPath$ structure to move an entire schedule of one team from the best solution and aim to integrate that schedule in the worst solution. Following that we execute the mutation either it happens or not based on the mutation rate.

```
Algorithm: Migration
    Input: Population, migrationRate
    Output: Population
1   Select Hi with probability αλ_i
2   If Hi selected then
3       For j in n do
4           Select Hj with probability αμ_j
5           If Hj selected then
6               t ← Extract the team that has the minimum path
7               SwapPath (Hj , Hi, t)
8           End if
9       End for
10  End if
11  Return Population
```

Then, we repeat the entire process of calculating costs, sorting the table, performing migration, and executing mutation for a predefined number of generations.

```
Algorithm: BBO
    Inputs: nbumberOfGenerations, populationSize, migrationRate, mutaionRate
    Output: BestSchedu
1   Call InitializePopulation(populationSize)
2   for i in nbumberOfGenerations do
3       Population ← Evaluate Population //calculate fitness and sort the list
4       Calculate s, λ, μ for each habitat // s = ln (index + 1)
5       Call migration (migrationRate)
6       call Mutation (mutaionRate)
7   end for
8   Return BestSchedul
```

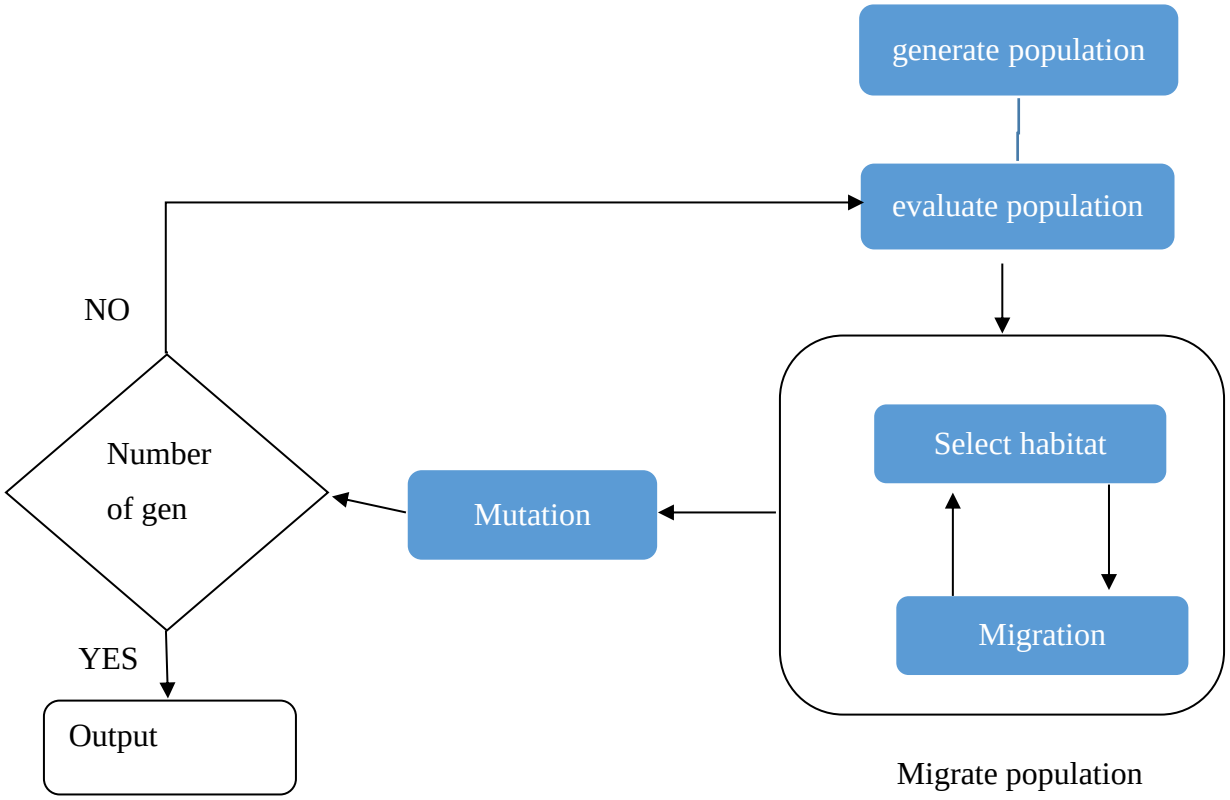


Figure V.1: BBO diagram

3. SLS with Reinforcement learning

During our studies on the TTP problem, we observed that the order of the rounds plays a significant role in minimizing the total traveled distance. This is because the order in which the games are played may have a significant impact on the entire travel schedule and, as a result, the teams total distance traveled. Trying all possible combinations of round orders is an exhaustive search can be very time-consuming, especially as the number of teams increases and considering that this process is repeated for every deferent set of rounds. We find a lot of similarities between this process and the traveling salesman problem (TSP) with some differences due to TTP's constraints. Our task can be split into two distinct components; manipulate the rounds structure to explore feasible regions, then uses a reinforcement learning model to solve the TSP corresponding to the current schedule structure.

The following steps demonstrate the modeling of a reinforcement learning model as a neighborhood structure in the sls heuristic.

3.1.Extract TSP

- Inputs:

  A double round robin tournament timetable

  Integer distance matrix

Teams	1	2	3	4
1	0	745	665	929
2	745	0	80	337
3	665	80	0	380
4	929	337	380	0

Table V.6: NL4 integer distance matrix

N = 4		Rounds					
		1	2	3	4	5	6
Teams	1	4	3	2	-4	-3	-2
	2	3	-4	-1	-3	4	1
	3	-2	-1	4	2	1	-4
	4	-1	2	-3	1	-2	3

Table V.1: A Double Round Robin Tournament (DRRT) schedule.

- Output

  TSP matrix

  Possible actions for each round

Round (city)	0	1	2	3	4	5	6

0	0	1009	1002	1125	1009	1002	1125
1	1009	0	2011	1490	2018	1490	2011
2	1002	2011	0	1974	1974	2004	2011
3	1125	1490	1974	0	1974	1490	2250
4	1009	2018	1974	1974	0	797	797
5	1002	1490	2004	1490	797	0	797
6	1125	2011	2011	2250	797	797	0

Table V.7: TSP matrix for the initial schedule

Round (city)	0	1	2	3	4	5	6
0	0	1	1	1	1	1	1
1	1	0	1	1	0	1	1
2	1	1	0	1	1	0	1
3	1	1	1	0	1	1	0
4	1	0	1	1	0	1	1
5	1	1	0	1	1	0	1
6	1	1	1	0	1	1	0

Table V.8: Possible actions matrix.

We consider the rounds as the cities and calculate the distance traveled by all the teams between the rounds. We extract a new matrix that represents the distance between the rounds which is the TSP representation of the current schedule. The first city (index = 0) represents the starting point to make sure that all teams begin in their home city and must return there after the tournament last constraints. The second matrix represents the possible action for each round (city) where 1 means that we can move from round i to round j and 0 indicate that we can't. Unlike traditional TSP it seems that the cities are not fully connected to each other that is due to the NoRepeat constraint.

3.2.Reinforcement learning (RL) for TSP

Machine and deep learning approaches shows a good potential for solving the TSP such ass GNN [80] and ANN [81]. We chose RL since it does not require any training data and due to its ability the dynamic change of the constraints over the process.

3.2.1. Environment :

- Observation: The observation visible to the agent is a (numCities + 1) length integer array, where
    - 1 indicates that the round has been visited
    - 0 indicates that the round is yet to be visited
- Action space: the agent can only take actions in a discrete space, which can also be described as [0, numRounds]. Action x will take the agent to round x
- Reward: a visit to a new round will reward the agent with the cost of the trip between nodes. We calculate the reward as follows: $Reward = 1\ /distance$, so that shorter connections will have more weight and to avoid the AtMost constraints we define a function that returns the available actions based on the current state, we use two lists to calculate the consecutive home and away games and check in the not selected rounds if it will violate the constraints so it masked that round.

The $reset(self)$ function Restart the environment for experience replay. Returns the first state in our case its always the round 0.

The $step(self, action)$ function Takes an action in a given state based on the current path to avoid constraints violation and it returns the next state and the reward of such an action

At the end the model we define a function to rebuild the TTP from the TSP path by eliminating both endpoints and follows the TSP path to order the rounds.

CHAPTER VI: NUMERUCAL RESULT

1 Introduction

In this final chapter, we present the numerical results obtained from our proposed approach for solving the Traveling Tournament Problem (TTP). Our approach leverages a deep learning model to predict optimal Biogeography-Based Optimization (BBO) parameters based on training data, aiming to enhance the performance of the BBO algorithm.

4. Numerical Results

In this section, we present the numerical results obtained from our experiments on solving the Traveling Tournament Problem (TTP) using the proposed approach. The results are summarized in a table that provides insights into the performance of our method. The table includes columns such as Instance, Best Known, Initial Solutions, Best Solution. The Instance column indicates the specific dataset used, while the Best-Known column represents the known optimal solution.

The Initial Solutions column displays the objective values before optimization, and the Best Solution column shows the optimal or near-optimal solutions obtained by our approach.

The results obtained from our experiments using the proposed approach for the TTP are summarized in a table. This table provides important information about the performance of our method. It includes

columns such as Instance, Best Known, Initial Solutions, Best Solution. These columns provide insights into the quality and efficiency of our approach. By comparing the Best Solution values with the Best-Known solutions, we can assess the quality of our method's results.

between our best solution and the best-known solution. Overall, these numerical results offer a

comprehensive analysis of our proposed approach's performance and its ability to produce competitive solutions for the challenging TTP.

We compared the results of this new method with the cumulative results of recent studies, which in turn achieved very similar results.

The benchmarks consist of a set of teams and their corresponding distances or times between venues.

Here are a few examples of TTP benchmarks:

NL4:

	Team 1	Team 2	Team 3	Team 4
Team 1	0	745	665	929
Team 2	745	0	80	337
Team 3	665	80	0	380
Team 4	929	337	380	0

Table VI.1: NL4 distance matrix

Instance	Best known	Initial solutions	BBO without DL	BBO with DL	Gap %
Con4	17	30	**17**	**17**	**0**
Con6	43	82	**43**	**43**	**0**
Con8	80	148	**80**	**80**	**0**
Galaxy4	416	477	**477**	**416**	**0**
Galaxy6	1365	1578	1408	1388	-1.68
Galaxy8	2373	3325	2651	2492	-5.01
NL4	8276	8413	8579	**8276**	**0**
NL6	23916	26893	24526	24073	-0.65
NL8	39947	57759	45628	43195	-8.13

Table VI.2: Numerical results found by the overall approach

Parameter Results:

To validate our approach, we conducted experiments using various instances of the TTP, including NL4, NL6, NL8, CON4, CON6, and CON8. For each instance, we applied our deep learning model

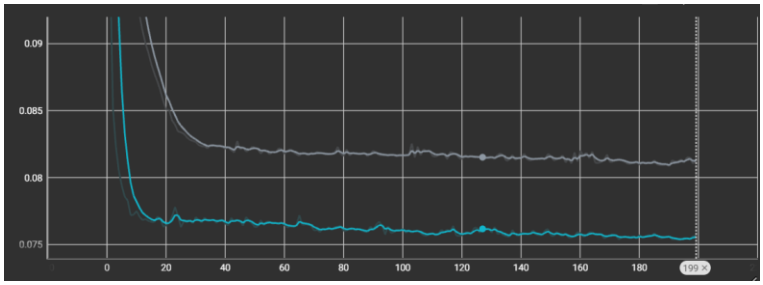
to predict the optimal BBO parameters and then ran the BBO algorithm with these parameters. The results are summarized as follows:

	Population size	generations	Migration rate	Mutation rate	Max iteration
NL4	19	19	0.48	0.48	97
NL6	140	139	0.31	0.44	305
NL8	172	118	0.09	0.12	355
CON4	29	30	0.5	0.36	34
CON6	106	109	0.5	0.47	43
CON8	87	87	0.24	0.46	230
Galaxy4	27	29	0.49	0.5	35
Galaxy6	91	102	0.18	0.36	150
Galaxy8	113	109	0.11	0.48	230

5. Deep learning model Performance Representation

Time:

This figure shows the training and validation loss curves over 200 epochs for our deep learning model used in optimizing the Biogeography-Based Optimization (BBO) parameters. The x-axis represents the number of epochs, while the y-axis indicates the loss value. The blue line depicts the training loss, and the grey line represents the validation loss. Initially, both the training and validation losses decrease rapidly, indicating that the model is learning effectively from the data. As the epochs progress, the training loss continues to decrease slightly and stabilizes, showing that the model is fitting well to the training data. Meanwhile, the validation loss also decreases and stabilizes, demonstrating that the model generalizes well to unseen data without significant overfitting. This figure is crucial for assessing the model's performance and ensuring that it achieves a good balance between fitting the training data and generalizing to new data.



FigureVI.1: Training and Validation Loss over Epochs

Distribution:

This figure shows the distribution of weights in one of the layers of the deep learning model over 200 epochs of training. The x-axis represents the weight values, while the y-axis shows the epoch number. The orange shading illustrates the density of weight values at each epoch. Initially, the weights have a narrow distribution around a central value, but as training progresses, the distribution becomes wider and more spread out. This behavior indicates that the model is adjusting its weights to better capture the underlying patterns in the data, leading to improved learning. The spreading of the weight distribution is a sign of the model's adaptation and optimization process as it fine-tunes its parameters to minimize the loss function. Monitoring the weight distribution helps in understanding the training dynamics and ensuring that the model parameters are updated effectively.

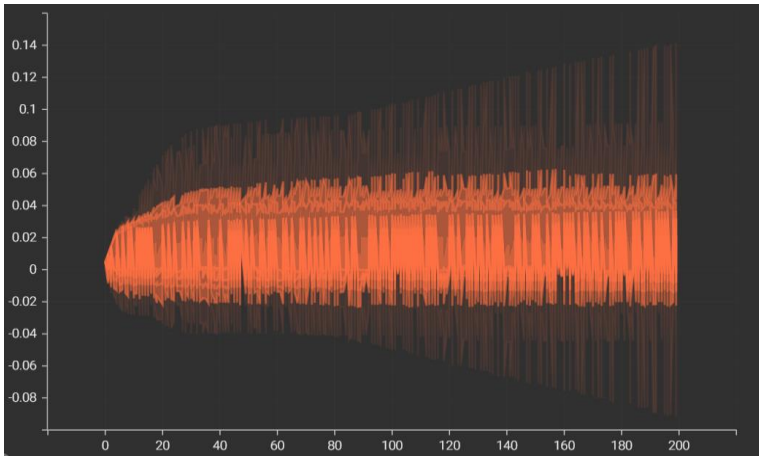
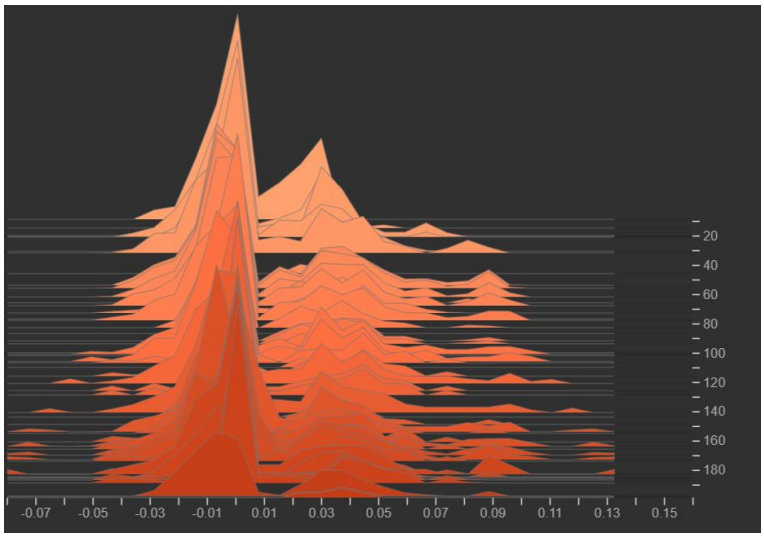


Figure VI.2: Gradient Distribution of the Model

Histogramme:

This figure depicts the distribution of biases in one of the layers of the deep learning model across 200 epochs of training. The x-axis represents the bias values, while the y-axis shows the epoch number. The orange shading indicates the density of bias values at each epoch. Initially, the biases are concentrated around a specific value, but as training progresses, the distribution broadens and becomes more varied. This broadening suggests that the model is actively adjusting its biases to better fit the training data, which is a crucial aspect of the learning process. By examining the evolution of bias distributions, we can gain insights into how the model parameters are being optimized over time. This information is valuable for understanding the training dynamics and ensuring the model's effective learning and generalization capabilities.



Fiaure VI.3: Bias Distribution Over Time

6. Limitations

While our proposed approach has demonstrated promising results in solving the Traveling Tournament Problem (TTP) using deep learning-enhanced Biogeography-Based Optimization (BBO), there are several limitations to consider. The deep learning model and the BBO algorithm require significant computational resources, especially for large instances of the TTP, which can limit the applicability of our approach to smaller datasets or environments with high computational power. Although our method performs well on benchmark instances like NL4, NL6, and NL8, its scalability to even larger instances remain uncertain, as performance may degrade or become impractical due to increased computational demands. The deep learning model is trained on specific instances of the TTP, and its ability to generalize to entirely new instances or different types of scheduling problems may be limited, necessitating further research to evaluate the model's robustness across diverse problem domains. The performance of the BBO algorithm is highly sensitive to the chosen hyperparameters, and while our model predicts these parameters, small changes can lead to significant variations in results, indicating a need for precise tuning. The effectiveness of the deep learning model also relies heavily on the quality and diversity of the training data; insufficient or biased training data can lead to suboptimal parameter predictions and hence, poorer optimization performance. Lastly, the convergence of the BBO algorithm can be

slow for certain instances, leading to longer computation times. Addressing these limitations will be crucial for enhancing the practical applicability and robustness of our proposed method in solving large-scale and dynamic instances of the Traveling Tournament Problem.

7. Conclusion

In this study, we presented a novel approach for solving the Traveling Tournament Problem (TTP) by leveraging a deep learning model to predict optimal parameters for the Biogeography-Based Optimization (BBO) algorithm. Our method aimed to enhance the performance and efficiency of the BBO algorithm by accurately predicting the best parameters based on training data, thus improving the overall solution quality for various instances of the TTP.

The numerical results demonstrated that our proposed approach is capable of producing competitive solutions, achieving near-optimal or optimal results for benchmark instances such as NL4, NL6, NL8, CON4, CON6, and CON8. The comparison of our results with existing methods showed that our approach can match or surpass the performance of state-of-the-art techniques, highlighting the potential of integrating deep learning with optimization algorithms.

Despite the promising outcomes, we also identified several limitations, including the high computational demands, the potential need for extensive hyperparameter tuning, and the challenges of generalizing the deep learning model to new or different problem instances. These limitations point to areas for future research, such as developing more efficient computational techniques, exploring adaptive hyperparameter tuning methods, and extending the model's applicability to a broader range of scheduling problems.

In conclusion, our study contributes a significant advancement in the application of machine learning to combinatorial optimization problems. The integration of deep learning with the BBO algorithm represents a powerful combination that can improve the efficiency and effectiveness of solving the TTP. Future work should focus on addressing the identified limitations and further refining the approach to enhance its robustness and scalability, ultimately contributing to more efficient and fairer tournament scheduling in practical applications.

Conclusion

The Traveling Tournament Problem (TTP) represents a challenging optimization problem with significant practical applications in sports scheduling. This thesis has provided a comprehensive overview of the TTP, discussed its complexity, and reviewed various methods and formulations for solving it. Throughout this work, we have emphasized the importance of the TTP in sports scheduling and optimization and have highlighted the benefits and limitations of our proposed approach integrating Biogeography-Based Optimization (BBO) and Stochastic Local Search (SLS) algorithms enhanced by machine learning and deep learning techniques.

Our proposed methodology demonstrated promising results in optimizing tournament schedules, reducing total travel distance, and adhering to the specific constraints of the TTP. The integration of deep learning for parameter tuning and reinforcement learning within the SLS algorithm has shown to be effective in addressing the complexities of the TTP.

Looking forward, our future research will focus on enhancing the efficiency and scalability of our solutions, particularly in tackling larger TTP instances. We plan to explore alternative strategies to improve the effectiveness of our approach, such as refining the machine learning models and investigating other evolutionary algorithms. Additionally, we aim to expand the applicability of our methods to other domains, further bridging the gap between traditional optimization techniques and modern machine learning approaches.

By addressing these challenges, we hope to advance the field of TTP research and its practical applications, contributing to more efficient and effective solutions in sports scheduling and beyond.

References

1. Pardalos, P. M., & Resende, M. G. C. (2014). *Introduction to Optimization Methods. Springer*.

2. Chong, E. K. P., & Żak, S. H. (2013). *An Introduction to Optimization (4th ed.). Wiley.*

3. Neller, T. W. (1994). *State Space Search. Springer.*

4. Boussaïd, I., Lepagnot, J., & Siarry, P. (2013). *An Introduction to Metaheuristics for Optimization. Springer.*

5. Wang, J. (Ed.). (2014). *Encyclopedia of Business Analytics and Optimization. IGI Global*.

6. Tan, Y., Shi, Y., & Niu, B. (Eds.). (2014). *Fireworks Algorithms and Swarm Intelligence. Springer.*

7. MathWorks. (n.d.). MathWorks. Retrieved June 15, 2024, URL: https://www.mathworks.com/

8. Dantong, X. (n.d.). Lecture Notes. University of Texas at Dallas. Retrieved June 15, 2024

9. Nie, Q. (n.d.). Numerical Optimization. University of California, Irvine. Retrieved June 15, 2024

10. Alseda, L. (n.d.). Constrained Optimization. Universitat Autònoma de Barcelona. Retrieved June 15, 2024

11. Talbi, E.-G. (2009). *Metaheuristics: From Design to Implementation. Wiley.*

12. Balas, E., & Toth, P. (1983). "Branch and Bound Methods for the Traveling Salesman Problem". Management Science Research Report, (MSRR 488).

13. Zitouni, F. (2021). *Recherche opérationnelle avancée.*

14. Bellman, R. E. (2021). Dynamic Programming. In Dynamic Programming. Princeton University Press.URL: https://doi.org/10.1515/9781400835386

15. Fomin, F. V., & Kratsch, D. (2010). Exact Exponential Algorithms. Springer-Verlag.

16. Gomes, C. P., & Williams, R. (2005). Approximation Algorithms. In E. K. Burke & G. Kendall (Eds.), Search Methodologies: Introductory Tutorials in Optimization and Decision Support Techniques p. 557–585. Springer US.

17. Southwell, R. V. (1946). Relaxation Methods in Theoretical Physics. Clarendon Press.

18. Frankel, S. P. (1950). Convergence rates of iterative treatments of partial differential equations. "Mathematical Tables and Other Aids to Computation, 4"(30), 65-75.

19. Young, D. M. (1950). Iterative methods for solving partial differential equations of elliptic type (Doctoral dissertation). Harvard University.

20. Kahan, W. (1958). Gauss–Seidel methods of solving large systems of linear equations (Doctoral dissertation). University of Toronto.

21. Kirkpatrick, S., Gelatt, C. D., & Vecchi, M. P. (1983). Optimization by simulated annealing. *Science, 220*(4598), 671-680.

22. Glover, F. (1989). Tabu Search, Part I. *ORSA Journal on Computing, 1*(3), 190-206.

23. Sampson, J. R. (1976). Adaptation in natural and artificial systems. In J. H. Holland.

24. Dorigo, M., & Di Caro, G. (1999). Ant colony optimization: A new meta-heuristic. In Proceedings of the 1999 Congress on Evolutionary Computation-CEC99 (pp. 1470-1477). IEEE.

25. Pillac, V., Gendreau, M., Guéret, C., & Medaglia, A. L. (2013). A review of dynamic vehicle routing problems. *European Journal of Operational Research, 225*(1), 1-11.

26. Simon, D. (2008). Biogeography-based optimization. *IEEE Transactions on Evolutionary Computation, 12*(6), 702-713. https://doi.org/10.1109/TEVC.2008.919004

27. Ma, H., & Simon, D. (2011). Blended biogeography-based optimization for constrained optimization. *Engineering Applications of Artificial Intelligence, 24*(3), 517-525.

28. Kirkpatrick, S., Gelatt, C. D., & Vecchi, M. P. (1983). Optimization by simulated annealing. *Science, 220*(4598), 671-680. https://doi.org/10.1126/science.220.4598.671

29. Moumita Ghosh and A. Thirugnanam *Introduction to Artificial Intelligence*

30. Rupali , K., & Deepali , S. (2018). APPLICATIONS OF ARTIFICIAL INTELLIGENCE IN HUMAN LIFE. *International Journal of Research -GRANTHAALAYAH*.

31.The levels of artificial intelligence application in human resource systems January 202 *The European Journal of Applied Economics* 19(2)

32. “A Brief Review of Machine Learning and its Application” WANG Hua MA Cuiqin

33. O. Theobald, Machine Learning For Absolute Beginners, London, UK: Scatterplot press, 2017.

34. “An overview of machine learning classification techniques” Amer F.A.H. ALNUAIMI1* and Tasnim H.K. ALBALDAWI

35. S. Bansal, "ANALYTIXLABS," MACHINE LEARNING What is Classification Algorithm in Machine Learning? With Examples, 25 JANUARY 2023. [Online]. URL: https://www.analytixlabs.co.in/blog/classification-in-machine-learning

36.

Murphy, K. P. (n.d.). *Machine Learning A Probabilistic Perspective*

37. V. Rastogi, S. Satija, P. K. Sharma and S. Singh, "MACHINE LEARNING ALGORITHMS:OVERVIEW," International Journal of Advanced Research in Engineering and Technology (IJARET), vol. 11, no. 9, pp. 512-517, 2020.

38. https://www.javatpoint.com/unsupervised-machine-learning

39. Algorithms for Reinforcement Learning by Morgan & Claypool Publishers Csaba Szepesv´ari June 9, 2009∗

40. M. Yuval, B. Yaman and Ö. Tosun, "Classification Comparison of Machine Learning Algorithms Using Two Independent CAD Datasets," Mathematics , vol. 10, no. 3, p. 311, 2022.

41. R. V. K. Reddy and U. R. Babu, "A Review on Classification Techniques in Machine Learning," International Journal of advance research in science and engineering, vol. 7, no. 3, pp. 40-47, 2018.

42. A. Ram and Meenakshi, "Short Review on Machine Learning andits Application," SPECIALUSIS UGDYMAS / SPECIAL EDUCATION, vol. 1, no. 43, pp. 9894-9902, 2022.

43. Mohan Zhang “Unsupervised Learning Algorithms in Big Data: An Overview” Beijing 21st century international school, Beijing, China

44. A. Graves, A. Mohamed, and G. E. Hinton. Speech recognition with deep recurrent neural networks. CoRR, abs/1303.5778, 2013.

45. Goodfellow, I., Bengio, Y., & Courville, A. (2016). *Deep Learning*. MIT Press.

46. Schmidhuber, J. (2015). Deep learning in neural networks: An overview. *Neural Networks*, 61, 85-117.

47. A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, Advances in Neural Information Processing Systems 25,

48. Esteva, A., et al. (2017). Dermatologist-level classification of skin cancer with deep neural networks. *Nature*, 542(7639), 115-118.

49. B. Huval, T. Wang, S. Tandon, J. Kiske, W. Song, J. Pazhayampallil, M. Andriluka, P. Rajpurkar, T. Migimatsu, R. Cheng-Yue, F. A. Mujica, A. Coates, and A. Y. Ng. An empirical evaluation of deep learning on highway driving.

49. B. Huval, T. Wang, S. Tandon, J. Kiske, W. Song, J. Pazhayampallil, M. Andriluka, P. Rajpurkar, T. Migimatsu, R. Cheng-Yue, F. A. Mujica, A. Coates, and A. Y. Ng. An empirical evaluation of deep learning on highway driving.

50.Vaswani, A., et al. (2017). Attention is all you need. *Advances in Neural Information Processing Systems*

52. - Kelly Easton, George Nemhauser, and Michael Trick. The traveling tournament problem description and benchmarks. In Toby Walsh, editor, Principles and Practice of Constraint Programming — CP 2001, pages 580–584, Berlin, Heidelberg, 2001.Springer Berlin Heidelberg.

53. -Constraint Programming for the Travelling Tournament Problem By Gan Tiaw Leong

54. Kelly Easton, George Nemhauser, and Michael Trick. Solving the Travelling Tournament Problem: A Combined Integer Programming and Constraint Programming Approach. In Edmund Burke and Patrick De Causmaecker, editors, Practice and Theory of Automated Timetabling IV, pages 100–109, Berlin, Heidelberg, 2003. Springer Berlin Heidelberg.

55. S. Urrutia and Celso C. Ribeiro. *Maximizing breaks and bounding solutions to the mirrored traveling tournament problem*. *Discrete Applied Mathematics*, 154(13):1932 1938, 2006. Traces of the Latin American Conference on Combinatorics, Graphs and Applications.

56. N. Fujiwara, S. Imahori, T. Matsui, and R. Miyashiro. *Constructive algorithms for the constant distance traveling tournament problem.* PATAT'06, page 135–147, Berlin, Heidelberg, 2006. Springer-Verlag.

57. M. Trick. “Challenge traveling tournament instances”. URL https://mat.tepper.cmu.edu/TOURN/. Accessed: 2022-05-12.

58. R. Bhattacharyya. *Complexity of the unconstrained traveling tournament problem.* *Operations Research Letters*, 44(5):649–654, 2016.

59. C. Thielen and S. Westphal. *Complexity of the traveling tournament problem.* *Theoretical Computer Science*, 412:345–351, 02 2011.

60. D. Chatterjee. “*Complexity of traveling tournament problem with trip length more than three”*, 2021. arXiv:2110.02300.

61. C. Ribeiro and S. Urrutia. *Heuristics for the mirrored traveling tournament problem*. *European Journal of Operational Research*, 179(3):775–787, 2007.

62. Kevin K.H. Cheung. “*Solving mirrored traveling tournament problem benchmark instances with eight teams*”. Discrete Optimization, 5(1):138–143, 2008.

63. Kevin K.H. Cheung*.” A benders approach for computing lower bounds for the mirrored traveling tournament problem”*. Discrete Optimization, 6(2):189–196, 2009.

64. A. Jaafari *et al.*, “*Meta optimization of an adaptive neuro-fuzzy inference system with grey wolf optimizer and biogeography-based optimization algorithms for spatial prediction of landslide susceptibility*,” *CATENA*, vol. 175, pp. 430–445, Apr. 2019

65. M. Ghobaei-Arani, “A workload clustering based resource provisioning mechanism using Biogeography based optimization technique in the cloud based systems,” *Soft Comput.*,vol. 25, no. 5, pp. 3813–3830, Mar. 2021, doi: 10.1007/s00500-020-05409-2.

66. M. Khelifa, D. Boughaci Computing and Informatics, Vol. 37, 2018, 1386{1410, doi: 10.4149/cai 2018 6 1386.

67. S.Imahori, T. Matsui, and R. Miyashiro. “A 2.75-approximation algorithm for the unconstrained traveling tournament problem”. Annals of Operations Research, 218, 10 2011.

68. D. de Werra. Some models of graphs for scheduling sports competitions. *Discrete Applied Mathematics*, 21(1):47–65, 1988.

69. R.Thomas Campbell and DS Chen. “A minimum distance basketball scheduling Problem”. Management science in sports, 1976

70. Clemens Thielen and Stephan Westphal. Approximation algorithms for TTP(2). Mathematical Methods of Operations Research, 76:1–20, 2012.

71. Shinji Imahori. A 1 + O(1/n) approximation algorithm for TTP(2), 2021. arXiv:2108.08444.

72. J.Zhao and Mingyu Xiao. The traveling tournament problem with maximum tour length two: A practical algorithm with an improved approximation bound. In Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence, International Joint Conferences on Artificial Intelligence Organization, 08 2021.

73. Mingyu Xiao and Shaowei Kou. “An Improved Approximation Algorithm for the Traveling Tournament Problem with Maximum Trip Length Two”. In Piotr Faliszewski, A.Muscholl, and R.Niedermeier, editors, Proceedings in Informatics (LIPIcs, Dagstuhl, Germany, 2016. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.

74. Diptendu Chatterjee and Bimal Kumar Roy. An improved scheduling algorithm for traveling tournament problem with maximum trip length two. 2021. arXiv:2109.09065.

75. Shinji Imahori, Ryuhei Miyashiro, and Tomomi Matsui. An approximation algorithm for the traveling tournament problem. Annals of Operations Research, 194:317–324, 2012.

76. Stephan Westphal and Karl Noparlik. A 5.875-approximation for the traveling tournament problem. *Annals of Operations Research*, 218:1–14, 01 2010.

77. D. Yamaguchi, S. Imahori, R. Miyashiro, and T. Matsui. An improved approximation algorithm for the traveling tournament problem. *Algorithmica*, 61:1077–1091, 01 2011.

78. R. Hoshino and K. Kawarabayashi. “The inter-league extension of the traveling tournament problem and its application to sports scheduling”. Proceedings of the AAAI Conference on Artificial Intelligence, 25(1):977–984, 08 2011.

79. R. Hoshino and K. Kawarabayashi. “An approximation algorithm for the bipartite traveling tournament problem”. Mathematics of Operations Research,

38(4):720–728, 2013.

80.Chaitanya K. Joshi , T. Laurent, and X. Bresson “Graph Neural Networks for the Travelling Salesman Problem” NTU, Singapore, LMU, LA, USA 22nd October, 2019

81.S. Roy, S. Sarma, S. Chakravorty, S. Maity “A COMPARATIVE STUDY OF VARIOUS METHODS OF ANN FOR SOLVING TSP PROBLEM”