People's Democratic Republic of Algeria
الجمهورية الجزائرية الديمقراطية الشعبية
Ministry of Higher Education and Scientific Research
وزارة التعليم العالي والبحث العلمي

Kasdi Merbah University of Ouargla
جامعة قاصدي مرباح ورقلة

Academic Master Thesis
To obtain a master's degree in Computer Science
Major : Fundamental Computing

A combined evolutionary metaheuristic techniques and integer programming approach to the traveling tournament problem

Realized by:
Hacini Mohammed Abdelaziz
Baylik Afifa

Supervised by:
DR. Khelifa Meriem
(UKMO)

Presented in 17 june 2023, in front of the jury composed of:
DR. Zitouni Farouk: UKMO - president
DR. AMER KHADIDJA: UKMO - examiner

Promotion: 2022/2023

Contents

List of Figures

List of Table

Dedication 1

I offer my thanks and loyalty to God Almighty for his great bounties and countless virtues and for granting me the grace to complete the completion of this memorandum.
To my mother ZOHRA who raised me and passed away, may god have mercy upon her.
to my father MOHAMMED thank you for being a pillar for me, and a guide in the path of life.
To my sisters FATIMA, ZOLIKHA, KALTOUM, MERIEM the one who was a support for me to continue my studies and for the encouragement and honest wishes of success to me.
To my brothers Abdellatif , Abdelhalim for give me moral support.
To my dear friend Abdelkader.B. for encouraging me to complete my studie.
To my family all by his name and dearest friend Houria , Bouchra ,Fatima , Manel.
To my colleagues Fatima ,Khadidja, Mohamed Abdelaziz at work .
Special gratitude to Dr. khalifa meriem for their valuable input.

Thank you all

BAYLIK AFIFA

Dedication 2

I dedicate this work to my family and friends who have supported me throughout. I am especially grateful to my dear friend Marwa, who has been a constant source of support and has restored my confidence in myself, enabling me to complete this project and overcome various challenges. I would also like to express my special thanks to my colleagues Mahdi, Abdelali, badro, Afifa, and Zahra for their continuous collaboration and assistance throughout the duration of this endeavor. Lastly, I extend a special expression of thanks to Professors Meriem Khelifa , Farouk Zeitouni for their valuable input.

M.AbdelAziz

Acknowledgment

First and foremost, we thank God for giving us wisdom and good health to undertake this humble endeavor. We would like to express our sincere appreciation to our supervisor, Mrs. Khalifa Mariam, for her invaluable guidance and supervision throughout the preparation of this work. We are also deeply grateful to all the members of the jury for their kind approval of evaluating our work and for their valuable contribution to the jury.

Abstract

This study focuses on the Traveling Tournament Problem (TTP), an NP-hard combinatorial optimization problem in sports scheduling. TTP holds significant importance within the sports community as suboptimal optimization can lead to financial losses in league management budgets. TTP aims to create a double round-robin tournament schedule while minimizing the total distance traveled by the teams and adhering to specific constraints unique to TTP.

To solve the Traveling Tournament Problem (TTP), we have proposed a new method that combines the benefits of biogeographic-based optimization (BBO) and branch and bound techniques. This hybrid approach allows for a balanced exploration and exploitation of the search space. The exact (branch and bound)method technique performs the exploitation steps, while the BBO technique performs the exploration steps. This combination, applied for the first time in TTP, aims to balance thorough exploration and efficient exploitation.

The proposed method is evaluated on publicly available standard benchmarks and compared to other state-of-the-art approaches. The computational experiments demonstrate that our method achieves promising and interesting results, which can be compared favorably to previous techniques.

Keywords : traveling tournament problem, biogeographic based optimisation, branch and bound, method exact, NP hard combinatorial optimisation, benchmarks

Résumé

Cette étude se concentre sur le problème du voyage dans les tournois (TTP), qui est un problème complexe d'optimisation combinatoire dans la planification des sports. L'objectif du TTP est de créer un calendrier de tournoi à double round-robin tout en minimisant la distance totale parcourue par les équipes et en respectant en même temps les contraintes spécifiques du TTP. Le TTP revêt une importance considérable au sein des communautés sportives, car une optimisation sous-optimale peut entraîner des pertes financières dans les budgets de gestion des ligues.

Pour résoudre le problème du voyage dans les tournois (TTP), nous avons proposé une nouvelle méthode qui combine les avantages de l'optimisation basée sur la biogéographie (BBO) et methode exact BB(branch and bound). Cette approche hybride permet une exploration et une exploitation équilibrées de l'espace de recherche, où la methode exact (ME) effectue des pas d'exploration très proches, tandis que la technique BBO effectue des pas d'exploration plus larges. Cette combinaison, appliquée pour la première fois dans le contexte du TTP, vise à trouver un équilibre entre une exploration approfondie et une exploitation efficace.

La méthode proposée a été évaluée sur un ensemble de benchmarks standard disponibles publiquement et comparée à d'autres approches de pointe. Les expériences de calcul montrent que notre méthode obtient des résultats prometteurs et intéressants, qui peuvent être comparés favorablement à d'autres techniques modernes.

Mots clés : problème de voyage dans les tournois, optimisation basée sur la biogéographie, tri et sélection, méthode exacte, optimisation combinatoire NP difficile, benchmarks.

ملخص

تركز هذه الدراسة على مشكلة السفر في البطولات (TTP)، وهي مشكلة تحسين تركيبية صعبة في جدولة الرياضات. هدف TTP هو إنشاء جدول لبطولة ذهاب وإياب مزدوجة يقلل من المسافة الإجمالية التي يسافرها الفرق ويتوافق في نفس الوقت مع القيود الخاصة لـ TTP. يحتل TTP أهمية كبيرة ضمن المجتمعات الرياضية، حيث يمكن أن تتسبب التحسينات غير المثلى في خسائر مالية في ميزانية إدارة الدوريات.

لحل مشكلة السفر في البطولات (TTP)، قمنا بتقديم طريقة جديدة تجمع بين فوائد تقنية الأمثلية الجغرافية الحيوية (BBO) وتقنية الفرز والتحديد (BB). يتيح هذا النهج المزدوج التوازن بين استكشاف واستغلال مساحة البحث، حيث تقوم تقنية الفرز والتحديد (BB) بخطوات استكشاف متقاربة جدًا، في حين تقوم تقنية الأمثلية الجغرافية الحيوية (BBO) بخطوات استكشاف متباعدة. تهدف هذه الجمعية، والتي تم تطبيقها للمرة الأولى في سياق TTP، إلى إيجاد توازن بين استكشاف شامل واستغلال فعّال.

تم تقييم الطريقة المقترحة على مجموعة من المقاييس القياسية المتاحة للجمهور ومقارنتها مع تقنيات أخرى حديثة لـ TTP. تظهر التجارب الحسابية أن الطريقة المقترحة تحقق نتائج واعدة ومثيرة للاهتمام، ويمكن مقارنتها مع أساليب أخرى حديثة

الكلمات المفتاحية: مشكلة السفر في البطولات، تحسين الجغرافيا الحيوية المبنية على الأحياء، طريقة الدقة المطلقة، تحسين تركيبي

General Introduction

Context :

Desite the difficulties and challenges that humans life is full of, humans can learn from their past experiences and adopt continous improvement to enhance the quality of life and achieve goals with greater efficiency. Therefore, humans evolve and progress in their lives and achieve successes and accomplishments in various areas of their lives.

Optimization problems can be found in many fields, including engineering, economics, finance, management, sport, and operations research. The goal is to find the most efficient or effective way to allocate resources, make decisions, or design systems. Optimization is an important tool in decision-making and problem-solving, as it allows for the evaluation of different options and the identification of the best course of action. It is also essential for improving processes and systems, reducing costs, and increasing efficiency. Modern optimization techniques involve the use of algorithms and computer programs, which can handle large and complex problems. These tools allow for faster and more accurate optimization, and have contributed to the growth of optimization in many fields. There are different method to solve optimization problems, including **exact method** and **approximate algorithms** the exact method yields optimal solutions but take a long time to do so as it is unable to solve the most complex problems (NP_hard problems), approximate algorithms (heuristic and meta_heuristic), while they can provide high_quality solutions within a reasonable time, there is no assurance of finding a globally optimal solution**.** Each type of problem requires a different approach and set of techniques.

Problematic

This study focuses on the traveling tournament problem (TTP), a well-known NP_hard problem. TTP is a fascinating problem in the fields of sports scheduling and combinatorial optimization

Objective

A hybrid between the exact method and the biogeographic-based optimization (metaheuristic) to improve the result

Organization of the thesis

To properly present our work, we have chosen the following structure: We start our thesis with a general introduction introducing then:

The first chapter We give are view of the main definition and fundament a lissues of the optimization problem.

The second chapter We describe the traveling tournament problem (TTP), the main focus of this work. We also give an overview of past approaches to the TTP.

In the third chapter We present the proposed approach for TTP problem in detail.

In the last chapter w highlights the experiments and compares our results with the best ones in the literature.

Chapter 1 :The combinatorial optimization

Chapter 1 : The combinatorial optimization

1-1 Introduction

Combinatorial optimization, which emerged as a separate subject around 50 years ago, is currently considered one of the most dynamic and cutting-edge fields of discrete mathematics, driving much of its progress today[1]

Combinatorial optimization is the branch of optimization theory that deals with discrete and often combinatorial problems. The field has its origins in the mid-20th century, when researchers began applying mathematical tools to solve problems related to transportation, scheduling, and logistics. In the late 1940s and early 1950s, George Dantzig developed the simplex algorithm, which is still the most widely used algorithm for solving linear programming problems. Linear programming provides a framework for modeling and solving many combinatorial optimization problems.[2] and In the 1950s and 1960s, researchers also began to study the traveling salesman problem, which involves finding the shortest possible route that visits a given set of cities and returns to the starting point. This problem has been one of the most intensively studied combinatorial optimization problems.[3] In recent years, there has been significant progress in developing algorithms for large-scale combinatorial optimization problems. This progress has been driven by advances in optimization theory, algorithms, and computer hardware.[4]

In these chapter, we will cover the basic concepts and fundamental points of combinatorial optimization, which include the concept of optimization and identifying some problems that require them. Then, we will discuss various solution methods, and finally, we will learn about complexity classes.

1-2 Optimization problems

Search space

Search space in combinatorial optimization refers to the set of all possible solutions to a given problem that satisfy the constraints on the problem's variables.[5]

Exploration

Exploration in combinatorial optimization refers to the process of systematically searching through a large space of possible solutions to identify the optimal solution. This process involves examining various combinations of discrete variables to find the best solution while satisfying certain constraints or objectives. The goal of exploration is to find the global optimum or the best solution that minimizes or maximizes the objective function.[5]

Exploration in combinatorial optimization is typically achieved through the use of algorithms, such as exact algorithms, heuristic methods, and metaheuristic approaches[5]. Exact algorithms explore all possible combinations of the problem's variables to find the optimal solution[6], while heuristic methods seek to find a good solution quickly by exploring a subset of the solution space[7], [8] Metaheuristic approaches use high-level strategies to search for good solutions and can be used to solve very large-scale problems.[9]

Exploitation

Exploitation in combinatorial optimization refers to the process of refining and improving a solution that is already known to be feasible or optimal.[5] This process involves making small adjustments to the solution and evaluating the resulting improvement in the objective function[8]. The goal of exploitation is to find a local optimum or to improve the quality of a known solution.[10]

Global optimum

Global optimum in combinatorial optimization refers to the best possible solution[5] that satisfies all the constraints [11]and optimizes the objective function over the entire search space of the problem.[12]

Local optimum

Local optimum in combinatorial optimization refers to a solution that is optimal within a specific region of the search space[5], but may not be the best possible solution over the entire search space.[10]

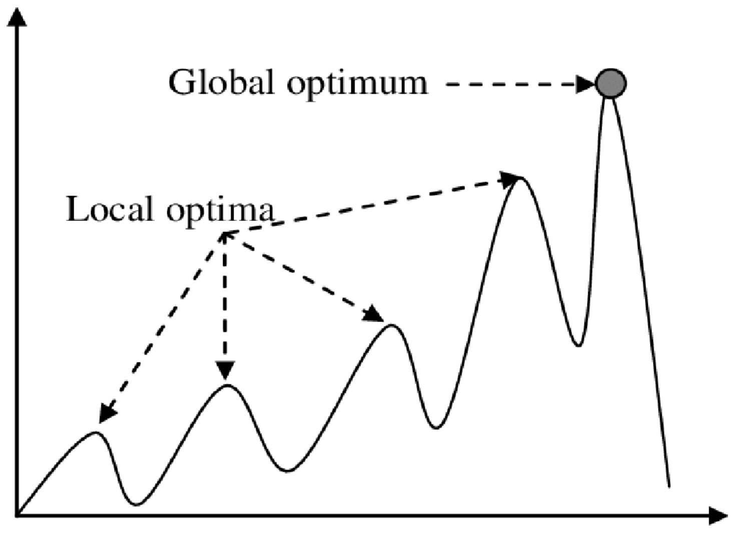


Figure 1-1 :local optimum VS global optimum

Optimization problem is a problem of mathematics and computer science that deals with finding the best solution among a finite set of possible solutions for a problem characterized by a discrete set of variables [13]. These problems typically involve searching through a large number of possible combinations to identify the optimal solution, subject to some constraints or objectives. Combinatorial optimization is an essential tool in a wide range of applications, including logistics, scheduling, resource allocation, and network optimization.[14] There are various methods used for solving combinatorial optimization problems, including exact algorithms, heuristic methods, and metaheuristic approaches.[15] Exact algorithms aim to find the optimal solution by systematically exploring all possible combinations of the problem's variables. In contrast, heuristic methods seek to find a good solution quickly by exploring a subset of the solution space, often guided by heuristics or rules of thumb [16]. Metaheuristic approaches use high-level strategies to search for good solutions and can be used to solve very large-scale problems.[17]

The general form of an optimization problem can be expressed as follows :

Minimize (Maximize) f(x)

Subject to :

$g(x) \leq b$

$h(x) = c$

where ‘f(x)’ is the objective function, ‘g(x) ≤ b’ and ‘h(x) = c’ are the constraints, and ‘x ‘is the decision variable. The objective function is the function that we want to optimize, while the constraints define the feasible region in which the decision variable x must lie[18]

There is two main classes for optimization problem:

1-3 Constrained optimization

Constrained optimization involves finding the minimum or maximum value of an objective function subject to a set of constraints. The constraints restrict the feasible region of the optimization problem, meaning that the solution must satisfy certain conditions. Examples of constraints include inequalities (e.g., x >= 0) and equalities (e.g., Ax = b), where x is the vector of variables, A is a matrix, and b is a vector. Constrained optimization problems can be solved using various methods, such as simplex method (algebric and by matrix) , graphic method .

Exemple of them,the previous general form in definition

1-4 Unconstrained optimization

Unconstrained optimization, on the other hand, involves finding the minimum or maximum value of an objective function without any constraints on the variables. This means that the solution can take any value in the feasible region. Unconstrained optimization problems can be solved using techniques such as gradient descent, Newton's method, and quasi-Newton methods.

constrain optimisation to unconstrain optimization :

Constrained optimization problems can be solved changing them to unconstrain optimization ,

using various methods, such as Lagrange multipliers, KKT conditions, and penalty methods.

Lagrange multipliers :

The Lagrange multipliers method provides a rigorous mathematical framework for solving constrained optimization problems by converting them into unconstrained ones. On the other hand, the penalty method is relatively easy to implement in practice.

For instance, if we have a function that we want to minimize subject to certain constraints, the Lagrange multipliers method can be used to convert the constrained problem into an unconstrained one. This allows us to use standard optimization techniques to find the minimum value of the function

$$\min_{x\in R^d} f(x) \qquad x \in (x_1, x_2, \dots, x_d)$$

subject to multiple non linear equality constraints :

$$g_j(x) = 0 \quad , j = (1, \dots, m)$$

we can use MLagrange multipliers λj(j =1,...,M) to reformulate the problem as the minimization of the following function:

$$L(x, \lambda_j) = f(x) + \sum_{j=1}^{m} \lambda_j \, g_j(x)$$

Penalty function :

The penalty method is a popular approach for incorporating equality and inequality constraints into a non-linear optimization problem. This method is commonly used when dealing with optimization problems that have complex constraints.

$$\min_{x\epsilon R^n} f(x), \qquad x = (x_{1,\dots,} x_d)\epsilon R^d,$$

Subject to :

$$\Phi_i(x) = 0, \quad (i = 1, \dots, M),$$

$$\Psi_j(x) \leq 0, \quad (j = 1, \dots, N)$$

The idea is to define a penalty function so that the constrained problem is transformed into an unconstrained problem. One commonly used penalty formulation is

$$g(x) = f(x) + p(x)$$

Where p(x) is the penalty term defined by

$$p(x) = \sum_{j=1}^{N} v_j \, \max\left(0, \Psi_j(x)\right)^2 + \sum_{i=1}^{M} \mu_i \, |\Phi_i(x)| \, .$$

Here μi >0,μj >0 are penalty constants or penalty factors. The advantage of this method is to transform the constrained optimization problem into an unconstrained one.That is, all the constraints are incorporated into the new objective function. However,this introduces more free parameters whose values need to be defined so as to solve the problem appropriately.

Obviously, there are other forms of penalty functions that may provide smoother penalty functions. For example, we can define

$$\Pi(x, \mu_i, \nu_j) = f(x) + \sum_{i=1}^{M} \mu_i \phi_i^2(x) + \sum_{j=1}^{N} \nu_j \psi_j^2\,(x)$$

Where μi >>1 and νj ≥ 0, which should be large enough, depending on the solution quality needed.

1-5 Computational Complexity

In solving problems, there are often multiple approaches available. It becomes important to compare the performance of different algorithms and select the most suitable one for a specific problem. When analyzing an algorithm, we primarily focus on its computational complexity, which is further divided into two categories: time complexity and space complexity.

Time complexity refers to the amount of time or number of operations required by an algorithm to execute, typically measured in terms of the input size. It helps us understand how the algorithm's execution time grows as the input size increases. Algorithms with lower time complexity are generally more efficient and desirable.

Space complexity, on the other hand, refers to the amount of memory or storage space required by an algorithm to solve a problem. It measures the maximum amount of memory needed at any point during the execution of the algorithm. Similar to time complexity, algorithms with lower space complexity are preferred as they optimize the usage of memory resources.

By analyzing the time and space complexity of different algorithms, we can make informed decisions about which one is the most efficient and suitable for a specific problem. It allows us to assess the trade-offs between execution time and memory usage and choose the algorithm that best meets the requirements of the problem at hand.

1-5-1 problem classes

An algorithm is considered to have a polynomial running time, denoted as "P," if its execution time on an input of size "n" can be described by the worst-case time complexity formula O(n^k), where "k" is a positive constant. This category includes algorithms with linear, quadratic, cubic, and higher polynomial time complexities. Conversely, algorithms with exponential running time are not classified as polynomial [9]

The classification of algorithms into different complexity classes includes well-known categories:

- P (Polynomial Time): This class consists of problems that can be solved in polynomial time by a deterministic Turing machine. The running time of algorithms in this class can be bounded by a polynomial function of the input size.

- NP (Non-deterministic Polynomial Time): This class includes problems that can be verified in polynomial time by a deterministic Turing machine. While the solutions to these problems may not be found in polynomial time, if a solution is proposed, its correctness can be verified in polynomial time.

- NP-complete: NP-complete problems are a subset of NP problems that are both solvable in polynomial time (in a large polynomial time) and verifiable in NP. These problems are considered the most difficult problems in the NP class, and if a polynomial-time algorithm is discovered for any NP-complete problem, it would imply polynomial-time solutions for all NP problems.

- NP-hard: NP-hard problems are a broader class that includes problems that do not have polynomial-time solutions. These problems are at least as difficult as NP-complete problems, and all problems in the NP class can be reduced to NP-hard problems.

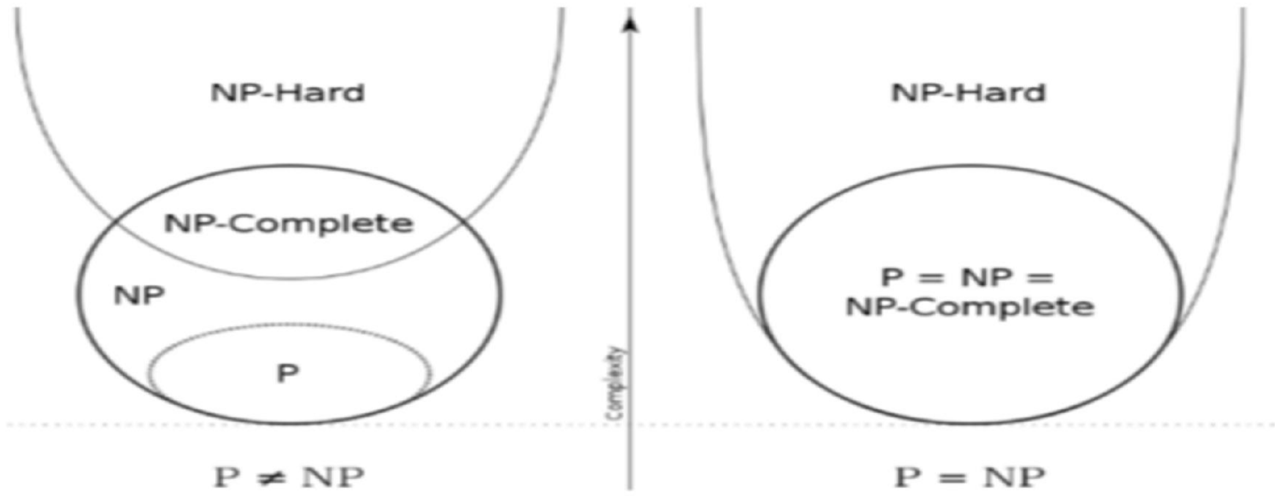


Figure 1-5-1 : There are two possibilities depending whether P = NP Or P≠ NP

In summary, the classification of problems based on their complexity helps us understand the efficiency and difficulty of solving them. Polynomial-time algorithms are considered efficient, while problems in the NP and NP-complete classes are more challenging, and NP-hard problems are the most difficult ones.

1-5-2 Time complexity

Time complexity refers to the measurement of the number of operations performed by an algorithm in order to accomplish a task. It assumes that each operation takes the same amount of time. The efficiency of an algorithm is determined by the one that requires the fewest number of operations, regardless of the specific machine it runs on.

In the context of time complexity, it is important to define what constitutes an operation. The choice of defining operations depends on the problem at hand. We typically select small operations that the algorithm frequently performs and consider them as basic operations for measuring complexity.

When computing the time complexity T(n) of an algorithm, it is rare to obtain an exact result. Instead, we typically obtain an estimate. This is why computer science utilizes asymptotic notations, which provide approximations of the number of elementary operations. There are three main asymptotic notations:

- Big Ω (Omega) notation: It represents the best-case scenario for time complexity. It provides a lower bound on the running time of an algorithm.
- Big Θ (Theta) notation: It represents the average-case scenario for time complexity. It provides a tight bound on the running time of an algorithm.
- Big O notation: It represents the worst-case scenario for time complexity. It provides an upper bound on the running time of an algorithm.

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 in 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.

The Big O notation provides a high-level understanding of how an algorithm performs as the input size increases. It allows us to make informed decisions about choosing the most efficient algorithm based on its scalability and efficiency requirements.

By utilizing these asymptotic notations, we can analyze and compare the efficiency of different algorithms and make informed decisions regarding their suitability for specific problem-solving tasks.

1-6 Resolution methodes

There are many solution methods in operational research for combinatorial optimization. They can be classified into two main categories: **exact methods**, which guarantee the attainment of an optimal solution but may require prohibitively long computation times; and **approximate methods**, which may not guarantee an optimal solution but are often able to provide good quality solutions within a reasonable computation time.

1-6-1 Exact methods

Exact methods aim to find the optimal solution by exhaustively exploring the entire search space, either explicitly or implicitly. These methods offer the advantage of guaranteeing the optimal solution. However, their computational time becomes exceedingly long as the problem size and the number of objectives to optimize increase. Consequently, exact methods are often limited in their practical use to small-sized problems.

There are several generic exact methods, including Branch & Bound, Branch & Cut, and Branch & Price. Additionally, there are more specialized methods such as Integer Linear Programming and the A* algorithm. Some methods are specific to certain problems, like Johnson's algorithm for scheduling.

Within the class of exact methods, classical algorithms such as Dynamic Programming, the Branch and X family of algorithms developed in the operations research community, Constraint Programming, and the A* family of search algorithms developed in the artificial intelligence community can be found [9]

A) The branch and bound algorithm :

a general-purpose algorithm used for finding optimal solutions in combinatorial optimization problems. It works by exploring the search space of possible solutions, dividing it into smaller subproblems and solving them recursively. The algorithm maintains a lower bound on the optimal solution value found so far and prunes parts of the search space that cannot possibly contain better solutions than the current best. Branch and bound is a widely used exact method for solving combinatorial optimization problems, and it has been applied to various fields, including operations research, computer science, and engineering.
A* is a graph traversal algorithm that finds the shortest path between two nodes in a graph. It uses a heuristic function to guide the search and expand the nodes that are most promising, leading to a faster search than uninformed search algorithms like breadth-first search or depth-first search. A* has been widely used in pathfinding applications, such as navigation systems and video games.

B) Dynamic programming : a problem-solving technique that involves breaking down a complex problem into smaller overlapping subproblems, solving each subproblem only once, and storing the solutions for later use. It is typically used for optimization problems where the optimal solution can be constructed from optimal solutions to subproblems.

In dynamic programming, the problem is divided into a series of stages or steps, and the solution to each stage is stored in a table or memoized. This allows for efficient computation by avoiding redundant calculations. The solutions to subproblems are used to build up the solution to the overall problem iteratively.
Dynamic programming has applications in various domains, including computer science, operations research, economics, and bioinformatics.[19], [20]

C) Linear programming : a mathematical optimization technique used to find the best possible outcome in a mathematical model characterized by linear relationships. It involves maximizing or minimizing a linear objective function, subject to a set of linear constraints. The objective function and constraints are represented by linear equations or inequalities.

In linear programming, the goal is to find the values of decision variables that optimize the objective function while satisfying all the constraints. The variables are typically subject to non-negativity constraints, meaning they cannot take negative values.

Linear programming has a wide range of applications, including resource allocation, production planning, transportation optimization, and portfolio optimization.[21]
Forme of linear programming :

$$\begin{aligned} &Minimize\ (maximize) && c_1x_1 + c_2x_2 + \cdots + c_nx_n = z \\ &\text{Subject to} && a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n = b_1 \\ &&& a_{21}x_1 + a_{22}x_2 + \cdots + a_{2n}x_n = b_2 \\ &&& \dots\dots\dots\dots\dots\dots\dots\dots \\ &&& a_{m1}x_1 + a_{m2}x_2 + \cdots + a_{mn}x_n = b_m \\ &&& x_1, x_2, \dots., x_n \geq 0 \end{aligned}$$

1-6-2 Approximate methods

Within the category of approximate methods, we can identify two subcategories: **approximation algorithms** and **metaheuristic algorithms**. Heuristic algorithms typically aim to find reasonably good solutions within a reasonable amount of time, while approximation algorithms focus on finding good solutions for large-scale problem instances.

A . Approximation algorithms

These are iterative methods that gradually build a solution step by step. Starting from an initially empty partial solution, they seek to extend the partial solution from the previous step at each iteration. This process continues until a complete solution is obtained.

Usage: Constructive methods are typically applicable when the quality of the solution is not a primary factor or when the instance size is reasonable. They are commonly used to generate an initial solution in a metaheuristic. These methods are fast and easy to implement.Heuristic algorithms can be further classified into two families: specific heuristics and metaheuristics. Specific heuristics are specifically tailored and designed to solve a particular problem or problem instance. On the other hand, metaheuristics are general-purpose algorithms that can be applied to solve a wide range of optimization problems.

A.1) *Glouton algorithm*

The process involves creating a viable solution by breaking it down into a series of decisions made at each step, guided by a local criterion, without reevaluating previous decisions. Typically, the resulting solution is an approximation.

Advantage: These algorithms are straightforward to implement.

Disadvantage: The quality of the approximate solutions can vary, as they rely on a local criterion.

Exemple : Optimal Placement of 2D Pieces (Bin Packing).

We have a set of rectangular plates that are all identical. Our objective is to arrange rectangular pieces on these plates in a way that ensures they do not overlap. The pieces we need to place have different dimensions.

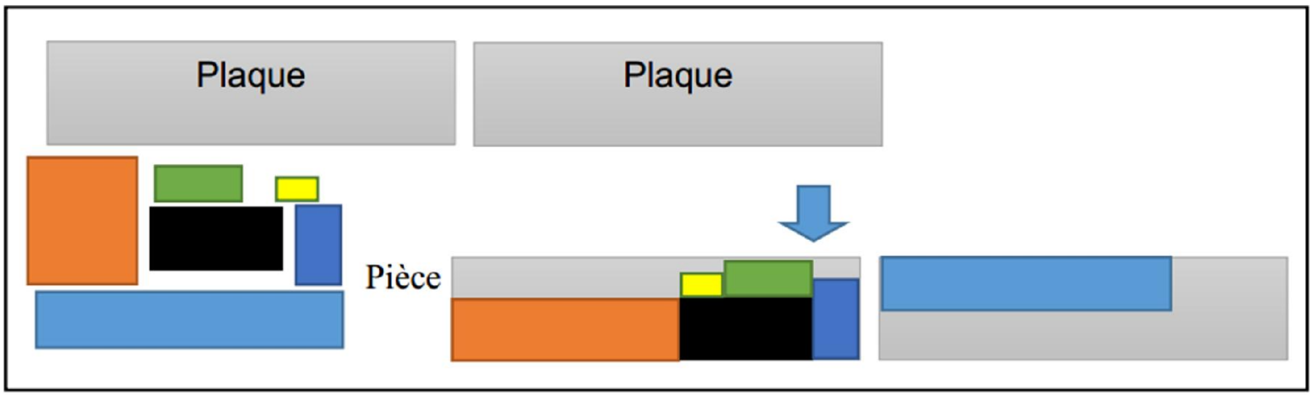


Figure A-1 : Optimal placement of pieces

Our objective is to minimize the number of plates used in the placement.To achieve this, we employ a greedy algorithm that involves sorting the pieces based on their size and then placing the largest pieces first.

B) Metaheuristic algorithms

The term "metaheuristic" is a combination of two Greek words:

"Heuristique," derived from the verb "heuriskein," meaning 'to find.'

"Méta," a suffix indicating 'beyond' or 'at a higher level.'

Metaheuristics are problem-solving methods that draw inspiration from nature. They are modern heuristics specifically designed for addressing optimization problems. Their primary objective is to achieve a global optimum, which is frequently concealed within numerous local optima.

Metaheuristics can be categorized into two subclasses:

- ✓ Neighborhood methods.
- ✓ Evolutionary methods.

B.1) Neighborhood Methods

Neighborhood methods involve starting with an initial solution, which can be obtained through exact methods or randomly. They then progressively move away from this solution to explore a trajectory or a progressive path within the solution space. This category comprises two commonly used methods

B-1-1) Simulated annealing

The simulated annealing method is an extension of the Monte Carlo method, designed to find the optimal solution for a given problem. It was developed in 1983 by three IBM researchers, S. Kirkpatrick, C.D. Gelatt, and M.P. Vecchi. Independently, V. Cerny also contributed to its development in 1985, drawing from the Metropolis algorithm that describes the evolution of thermodynamic systems [22].

The fundamental concept of simulated annealing, initially proposed by Metropolis in 1953, involves simulating the behavior of matter during the annealing process commonly used in metallurgy. The objective is to achieve a state of thermodynamic equilibrium, which corresponds to the optimal solution in simulated annealing, where the energy is minimized. The system's energy is evaluated using a cost function, also known as an objective function. The method aims to find the optimal solution by optimizing this objective function, utilizing a fictitious parameter known as temperature, introduced by Kirkpatrick, Gelatt, and Vecchi. In essence, the method iteratively generates configurations, starting from an initial solution S0 and an initial temperature T0, gradually decreasing the temperature throughout the process until reaching a final temperature or a state of equilibrium, which represents the global optimum.

B-1-2) Tabu search

The Tabu search, also called the local search method, was introduced in 1986 by F. Glover .Its main feature lies in implementing mechanisms that take inspiration from human memory. The concept is to remember past paths in a memory and refer to it to guide the search.

Basic Idea:
In each iteration, the tabu algorithm selects the best non-tabu neighbor, even if it leads to a lower cost function. For this reason, the tabu search is considered an aggressive method.

Tabu List (Memory):
The primary idea behind the tabu list is to remember the visited configurations or regions and use mechanisms to prevent the search from quickly revisiting them.

B.2) Evolutionary methods

These methods distinguish themselves from the ones previously studied by operating on a population of solutions. As a result, they are commonly referred to as population-based methods. Some of these methods draw inspiration from principles found in genetics and the behavior of insects. The intricate nature of these biological phenomena has provided a model for the development of increasingly advanced algorithms over the past two decades . In our discussion, we will exclusively focus on genetic algorithms.

Exempl :

- ✓ Genetic algorithms
- ✓ Ant Colony

B-2-1) Genetic algorithms

draw inspiration from the theory of evolution and the biological mechanisms that enable organisms to adapt to their surroundings. They were developed in the mid-1960s by researchers such as Holland, Rechenberg, and Fogel [23].

Natural selection, described by Darwin as the primary driving force behind evolution, favors individuals within a population that possess traits best suited to their environment. Following selection, individuals undergo crossover and mutation at the genetic level, where a set of genes forms an individual. When two parent individuals mate, they pass on a portion of their genetic makeup to their offspring. The offspring inherits characteristics that enhance its adaptation to the environment. Over successive generations, the selection process favors individuals with better adaptation, and the increasing number of well-adapted individuals propels the evolution of the entire population .

Implementing genetic algorithms involves several specific steps. The first step is encoding an individual's traits, represented by a chromosome. The second step is evaluating the fitness of each individual, typically through a fitness function. Finally, reproduction operators, such as crossover and mutation, are defined to generate new offspring and drive the evolutionary process.

B-2-2) Ant colony optimization

techniques have recently emerged as a novel metaheuristic for solving challenging combinatorial optimization problems. These techniques are inspired by the ability of ant colonies to find the shortest paths to food sources. Although individual ants have limited capabilities, their collective behavior as a colony exhibits complex problem-solving abilities.

Real ants communicate indirectly through pheromone trails, without relying on visual cues, to find the shortest path between food sources and their nests. As ants move, they deposit pheromone on the trail, and other ants follow these trails based on the density of the pheromone. The more ants that traverse a trail, the stronger the pheromone concentration becomes, attracting more ants to follow that path. Over time, this process enables ants to discover the shortest path.

Artificial ants in the context of optimization algorithms imitate the foraging behavior of real ants but can tackle much more intricate problems than their biological counterparts. The search algorithm that incorporates these principles is known as Ant Colony Optimization (ACO). Figure 1.7 illustrates how ants utilize this approach to find the shortest path.

In ACO, artificial ants traverse a problem space, gradually building solutions by making probabilistic decisions based on pheromone information. They update the pheromone trails, reinforcing the paths that lead to better solutions. Through iterations, the pheromone trails guide the search towards the most promising regions of the solution space, ultimately converging on the shortest path or optimal solution.

ACO has been successfully applied to various optimization problems, such as the Traveling Salesman Problem, Vehicle Routing Problem, and Job Shop Scheduling. By simulating the cooperative behavior of ant colonies, ACO offers an effective and efficient approach for finding high-quality solutions in complex optimization scenarios.

1-7 Conclusion

In conclusion, the field of combinatorial optimization is an important area of study that deals with solving complex optimization problems involving discrete decision variables. Throughout this chapter, we have introduced the fundamental concepts and principles necessary for understanding combinatorial optimization.

We discussed various types of optimization problems, such as constrain optimization , and unconstrain optimization .

Furthermore, we explored different resolution methods for combinatorial optimization problems, including exact methods that guarantee an optimal solution but may be computationally expensive, and approximate methods that provide fast solutions but with a certain level of approximation.

Time complexity was also discussed as a measure of an algorithm's efficiency, and we learned about asymptotic notations, such as Big O notation, which helps in analyzing the performance of algorithms in terms of time and space complexity.

In the upcoming chapter, we presented one of resolution methods, Among these methods is the Biogeography Based Optimisation(BBO), which we will talk about in the next chapter.

Chapter 2 : Biogeography Based Optimization (BBO)

Chapter 2 : Biogeography Based Optimization

2.1 History and Background of BBO

Biogeography-Based Optimization (BBO) is a nature-inspired optimization algorithm that is based on the principles of biogeography, which is the study of the distribution of species in different geographic regions. BBO was first introduced by Dan Simon in 2008 as a novel evolutionary algorithm for solving optimization problems.

2.2 BBO description

BBO, which stands for Biogeography-Based Optimization, is an evolutionary algorithm that draws inspiration from the field of biogeography science. Biogeography science focuses on studying the temporal and spatial distribution of species and biological organisms, including plant and animal life. Its primary objective is to understand how species are naturally distributed across different environments, how they sustain themselves, and how their characteristics evolve over time.

Mark developed mathematical models of biogeography that explain various aspects such as species migration between habitats, the emergence of new biological organisms, and the process of species extinction. A habitat refers to a distinct living space that is isolated from other spaces. The quality of each habitat is quantified using the habitat suitability index (HSI), which is determined by several variables known as suitability index variables (SIV). These variables include factors like rainfall, vegetation diversity, topographic features, land area, and temperature, which collectively describe the habitability of a particular habitat.

The characteristics of habitats can be described as follows:

1. Habitats with favorable geographical conditions for biological species are referred to as high habitat suitability index (HSI) habitats. It is evident that habitats with a high HSI tend to support a larger number of species. On the other hand, habitats with unfavorable geographical conditions are categorized as low habitat suitability index habitats.
2. High HSI habitats have a tendency to share their features with low HSI habitats through species migration. This means that certain characteristics present in high HSI habitats are retained while simultaneously appearing as new features in low HSI habitats.

2.3 Implementation BBO

The immigration rate (λ) for each habitat is formulated in a way that it is inversely proportional to the fitness value (HSI), as demonstrated in Figure 1. On the other hand, the emigration rate (μ) is directly proportional to the HSI value. The formulation for the emigration and immigration rates of each habitat is given by [31] as follows:

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

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

Where:

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

μ_{sp}: is the emigration rate in a habitat with sp species

I : refers to the maximum immigration rate

E : 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:

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

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

The population transition function involves several steps, starting with the computation of the immigration rate (λ) and emigration rate (μ) for each individual, as determined by formulas 2 and 3. One solution, denoted as "Habitati," is selected for modification, and the immigration rate of this solution is used to determine whether a suitability index variable (SIV) will be changed.

Once "Habitati" is selected, the emigration rate (μ) is used to determine which high-quality solution (denoted as "Habitatj") will migrate its SIV. The migration process, denoted as Ωn, is then applied between the immigrating habitat ("Habitati") and the emigrating habitat ("Habitatj"). In this process, the good solutions with favorable SIVs replace the worst solutions ("Habitati"), followed by a recalculation of the habitat suitability index (HSI).

After the migration process, a mutation operation (M^n) is performed, which represents a sudden change in the habitat's climate that may impact the HSI. The mutation operator randomly modifies the SIV of the habitat based on a given mutation rate. This mutation step is followed by another recalculation of the HSI for each habitat.

Algorithm 1 Migration process

```
1 : for i=1 :number habitats do
2 :  select a habitat Habitat i with probability $\alpha\lambda_i$
3 :  if Hi is selected then
4 :    if $rand \epsilon(0,1) < \lambda_i$ then
5 :      select habitat j with probability $\alpha\mu_i$
6 :      if Hj is selected then
7 :        Habitat i (SIV)← Habitat j (SIV)
8 :      end if
9 :    end if
10 :  end if
11 :end for
```

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

Where:

$p_{mut}(sp)$: 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 existenceof sp species in the habitat

$$p_{sp} = \begin{cases} \dfrac{\lambda_0\lambda_1 \dots . \lambda_{sp-1}}{\mu_1\mu_2 \dots \dots . \mu_{sp}(1 + \sum_{i=1}^{n} \frac{\lambda_0\lambda_1 \dots \dots \lambda_{I-1}}{\mu_1\mu_2 \dots . . \mu_I})} \dots . 1 \leq sp \leq number\_habitats \\ \dfrac{1}{1 + \sum_{I=1}^{n} \frac{\lambda_0\lambda_1 \dots . . \lambda_{I-1}}{\mu_1\mu_2 \dots \mu_I}} \dots sp = 0 \end{cases}$$

Algorithm 2 Mutation Algorithm :

```
1 : forj=1 to umber_habitats do
2 :   use $\lambda_i$ and $\mu_i$ to compute $Pmut_i$
3 :   select $Habitat_i$(j) with probability $\alpha\ Pmut_i$
4 :   if $habitat_i$(j) is selected then
5 :     Replace $Habitat_i$(j) with a random SIV
6 :   end if
7 : end for
```

2.4 Previous works of BBO

BBO has been successfully applied to a wide range of optimization problems in various domains. Here are a few examples of problems that have been solved using it

Economic Load Dispatch in Power Systems[24] , Meta optimization of an adaptive neuro-fuzzy inference system with grey wolf optimizer and biogeography-based optimization algorithms for spatial prediction of landslide susceptibility 2019 [25], A workload clustering based resource provisioning mechanism using Biogeography based optimization technique in the cloud based systems 2021[26] , Binary biogeography-based optimization based SVM-RFE for feature selection 2021 , CBO-IE: A Data Mining Approach for Healthcare IoT Dataset Using Chaotic Biogeography-Based Optimization and Information Entropy 2021[27], An efficient data replica placement mechanism using biogeography-based optimization technique in the fog computing environment 2023 [28] .

2.5 Conclusion

In this chapter, we have explained the biogeography based optimisation method, First, we discussed the history of BBO and then provided a detailed description of this algorithm. This algorithm is characterized by the migration of poor solutions towards good solutions. We then addressed the stages of this algorithm, which are create initial population and migration and mutation We also mentioned some problems in which BBO has been used to find solutions, demonstrating the effectiveness and strength of the algorithm in searching for optimal solutions, We want to apply this algorithm for the first time to the problem of... which we will explain in the next chapter .

Chapiter 3 :The Traveling Tournament Problem

Chapter 3 : The Traveling Tournament Problem

3.1 Introduction

Scheduling games in a fair and effective manner for all participating teams is an issue faced by sports leagues all around the world. An ideal tournament itinerary for a collection of teams is sought for by the Traveling Tournament Problem (TTP), a combinatorial optimization problem, in order to reduce the overall travel distance between venues[4]. Due to its potential to affect both the scheduling's practicality and the fairness of tournament results, this issue has major real-world applications. We will examine the numerous TTP variations, assess the problem's complexity, and discuss the various models and formulations that have been suggested to address it in this chapter. Readers will have a firm knowledge of the TTP and its significance in sports scheduling and optimization by the end of this chapter.

3.2 Background and History :

The Traveling Tournament Problem (TTP) is a combinatorial optimization problem that seeks to find the optimal tournament schedule for a group of teams while taking factors like the number of rounds, the number of games in each round, and the venues of the games into consideration [29].

The objective is to minimize the overall travel distance between venues while ensuring that each team plays an equal number of games and that each game is played at a different venue [29].

The problem is NP-hard, and several techniques and heuristics have been developed to assist in solving it [24]. The TTP is useful in industries like transportation planning, sports scheduling, and others where cutting down on travel time is important [4].

The Traveling Tournament Problem (TTP) has a long, illustrious history. The issue was initially raised in the context of sports scheduling, notably in the subject of combinatorial optimization. The TTP has been the focus of investigation and inquiry since the middle of the 20th century, despite the fact that the precise origins are not widely known.

In 1959, George B. Dantzig and John H. Ramser published a paper in which they proposed a mathematical model for the truck dispatching problem, which is related to the TTP. This is one of the early references to the TTP. The groundwork for future research into the issue and its variations was provided by this earlier work. [30]

Numerous academics have created metaheuristics and heuristics, such as simulated annealing, tabu search, evolutionary algorithms, and variable neighborhood search, in addition to exact methods, to solve the TTP. Variable neighborhood search (VNS), which Mladenovi and Hansen proposed in 2019 [24], is one of the TTP's most effective metaheuristics.

	1	2	3	4	5	6	7	8	9	10
1	6	-2	2	3	-5	-4	-3	5	4	-6
2	5	1	-1	6	4	3	-6	-4	-3	-5
3	-4	5	-5	1	6	-2	-1	-6	2	41
4	3	6	-6	5	-2	1	-5	2	-1	-3
5	-2	-3	3	-4	1	-6	4	-1	6	2
6	-1	-4	4	-2	-3	5	2	3	-5	1

Table 1.1: Example double round robin tournament schedule with even number of teams.

To maintain fairness, the tournament schedule frequently includes a home/away constraint that requires each team to play an equal number of games at home and away venues[1]. This aids in reducing any advantage that would be realized from playing more games in a comfortable setting.

In Table 1.1, the symbol "+" next to a team number indicates a home game, while "-" represents an away game. By examining the table, it is evident that when a team t plays at home against another team t', the latter team t' is always playing away against team t in the same time slot. This observation highlights the interconnectedness between the tours.

The number of rounds in a double round-robin tournament(DRR) is (2n-2), where n is the number of teams. To ensure fairness in scheduling and minimize travel distances, various constraints are implemented in double round-robin tournaments. The two main constraints are the Atmost and Norepeat constraints.

- The Atmost constraint limits the maximum number of consecutive home or away games for a team. Specifically, it ensures that no more than three consecutive home or away games are allowed for any team. This constraint ensures that a team does not have to play a string of games in either their home stadium or on the road, which could be disadvantageous.
- The Norepeat constraint ensures that a game of team i at team j's home cannot be followed by a game of team j at team i's home. This constraint ensures that no team plays another team twice in a row, which would be unfair and unbalanced. [31]

Team\Round	1	2	3	4	5	6
1	-3	-2	-4	+3	+2	+4
2	-4	+1	+3	+4	-1	-3
3	+1	-4	-2	-1	+4	+2
4	+2	+3	+1	-2	-3	-1

Table 1.2: illustrates the violation of the no-repeat and at-most constraints in (DRR).

3.3 TTP Complexity Analysis

A well-known challenging problem is The Traveling Tournament Problem (TTP), which is regarded as being severely 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 cases with more than ten teams in the best way up until this point. Bhattacharyya provided the first demonstration of NP-completeness for a modification of the original TTP,[25]

To solve TTP, a variety of algorithms, including exact and heuristic techniques, have been developed. For TTP instances, exact approaches offer the best solution, but they are computationally expensive and frequently unfeasible for large instances. A precise approach to solving TTP that uses a mathematical model to identify the best answer is called integer programming. Heuristic approaches, on the other hand, deliver a less-than-ideal solution in a fair length of time. TTP situations have been resolved using heuristics like greedy algorithms, local search algorithms, and genetic algorithms. The effectiveness and quality of these approaches are frequently good, but they do not ensure 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.[26]

3.4 Previous work

Numerous mathematical models have been, presented to solve The Traveling Tournament Problem (TTP), which has been researched for many years. The goal of the challenge is to reduce the aggregate trip distance or time of all teams while taking into account a number of limitations. This problem is often written as a mixed-integer linear programming (MILP) model. These restrictions include, among others, the round-robin format, no repetition, and at most restrictions. Alternative methods for mathematical programming have also been investigated recently, including dynamic programming (DP) and constraint programming (CP), among others. It has been demonstrated that these methods work well for resolving small to medium-sized TTP situations. Additionally, metaheuristic algorithms have been used to solve large TTP instances, offering close to optimal solutions, including genetic algorithms (GA), simulated annealing (SA), and tabu search (TS), among others. The choice of the most appropriate approach depends on the problem size, complexity, and available computer resources, even if each modeling approach has its own benefits and limits. [26]

3.5 Discussion of the previous work

Mixed-integer linear programming (MILP) is a popular formulation for TTP due to its ability to handle large problem sizes and its proven optimality guarantees. However, it can become computationally expensive as the problem size grows, and it may not be able to handle all types of TTP constraints[4]

Mathematical programming formulations, such as integer programming (IP), linear programming (LP), and constraint programming (CP), can also be used for TTP. IP and LP formulations are often less computationally expensive than MILP but may not provide as strong optimality guarantees. CP formulations, on the other hand, can handle complex constraints but may not scale well to larger TTP instances.[32]

Metaheuristic approaches, such as simulated annealing, genetic algorithms, and tabu search, have been widely used to solve TTP due to their ability to find good solutions quickly and their ability to handle complex constraints. However, they may not always provide optimal solutions and may require parameter tuning to achieve good performance.[33]

The size, limitations, and optimality of the solution all affect which TTP formulation is selected. A greater balance of quality and efficiency can be achieved by using hybrid techniques or metaheuristics.

In addition to earlier research on the Traveling Tournament Problem (TTP), there have been attempts to tackle the issue utilizing hybrid strategies, such as fusing local search methods with metaheuristic algorithms like genetic algorithms. These hybrid approaches seek to improve the search process and identify superior answers by utilizing the advantages of both paradigms. The inherent complexity of the TTP makes it vital to keep in mind that these hybrid methods may not always lead to the best solution. They might run into problems like premature convergence and trouble avoiding local optimums.

3.6 Review of various applications and case studies of TTP

The Travelling Tournament Problem (TTP) has numerous potential applications across a wide range of industries. One of the most popular uses of TTP is in sports scheduling, where it may be used to create league and tournament schedules that guarantee competitive balance and fairness. TTP can also be used to optimize the routes and schedules for vehicles and drivers in logistics and transportation planning. TTP has been applied in more specialized sectors as well as these more conventional ones. For instance, TTP has been applied to improve academic timetables and conference schedules, ensuring that all sessions and events are planned in an effective and efficient way. It has also been applied in the design of clinical trials, where it can be used to create balanced schedules of treatments and interventions for patients.

TTP has been applied in numerous applications, but its usefulness in real-world situations is still being researched. While some studies have demonstrated that TTP can provide balanced and fair timetables, others have observed that it might be challenging to discover the best solutions due to the problem's restrictions. Future work in this field should concentrate on identifying the best uses of TTP and creating more effective algorithms to tackle the issue.

The creation of hybrid algorithms that use various optimization methods to tackle TTP is one promising area of future study. For instance, it is possible to find strong optimality guarantees and high-quality solutions quickly by combining mathematical programming and metaheuristics. Algorithms that can learn from previous solutions and adapt to new issue situations could also be developed using machine learning and artificial intelligence.

3.7 Conclusion

The summary of key points discussed in this chapter includes the significance of TTP in sports scheduling and optimization. TTP is a significant issue that has gained much attention in academic and industrial study. It is a difficult combinatorial optimization issue involving the scheduling of a round-robin event and optimizing the overall travel distance for all teams. TTP comes in a variety of variations and formulations, each with its own set of pros and downsides.

Furthermore, the closing views underline the significance of TTP in academic and corporate research. TTP has been extensively researched, and numerous efficient algorithms and heuristics for solving the problem have been devised. Furthermore, TTP has real-world uses other than sports scheduling, such as routing air traffic or organizing conferences. As a result, more TTP research is required to investigate its possible uses, refine existing algorithms and heuristics, and propose novel approaches to solving the problem. The implementation of the suggested strategy to attain this solution will be discussed in the next chapter.

CHAPTER 4: Proposed approach (ME+BBO)

Chapter 4: Proposed approach (ME+BBO)

4.1 Introduction

In the previous chapter, we were introduced to the Sports Scheduling Problem and its variants, particularly the Traveling Tournament Problem (TTP) and its constraints. In this chapter, we will focus on applying the proposed heuristics approach to solve the TTP. We will first utilize the BBO method, followed by the exact method. Additionally, we will explore the implementation of Neighborhood Structures and different swap techniques within the BBO approach to further enhance our solution.

4.2 Proposed approach

In order to address the scheduling problem of The Traveling Tournament (TTP), we employed our proposed approach, which involved several steps. Firstly, we defined the necessary inputs and outputs for the problem. Subsequently, we implemented the BBO algorithm and the exact method to solve the TTP. The following steps outline the execution of these algorithms and their application to the problem at hand.

• Input :

1- The number of teams n ; in this example we chose the number of teams n=4
2- Integer distance matrix between each team and the other team :

Teams	Team 1	Team 2	Team 3	Team 4
Team 1	0	10	15	34
Team 2	10	0	22	32
Team 3	15	22	0	47
Team 4	34	32	47	0

Table 3.2.1: The distance between the teams cities

• Output:

- A double round robin tournament on the n teams
- The total distance traveled by the teams is minimized . .

• the number of rounds = 2 (number of teams -1) , In this example the number

of rounds= 6 rounds

• 1 schedule for(n *r): The negation sign means that the team plays away.

	Round 1	Round 2	Round 3	Round 4	Round 5	Round 6
Team 1	4	2	3	-4	-2	-3
Team 2	3	-1	4	-3	1	-4
Team 3	-2	4	-1	2	-4	1
Team4	-1	-3	-2	1	3	2

Table 3.2.2: Schedule for n = 4 teams

4.3 Ineger Programming

4.3.1 Objective Function

In this case , we will focus on the mathematical formulation of the Traveling Tournament Problem (TTP). The objective function of TTP is to minimize the total distance traveled by all teams while satisfying the constraints of the problem. The decision variables in the TTP formulation include xijk and tij, which represent whether team i plays against team j in round k and whether team t travels from i to j, respectively.

Min : $\sum_{i,j=1}^{n} d[i][j] \, * x[i][j]1 \; + \sum_{t,i,j=1}^{n} d[i][j] \, * y[t][i][j] \; + \sum_{i,j=1}^{n} d[i][j] \, * x[i][j][2n-2]$

$\boldsymbol{d_{ij}}$/ distance matrix and represent the distance between team i and team j

$\boldsymbol{x_{ijk}}$/ team i play against j in round(day) k

$\boldsymbol{y_{tij}}$/ team t traveling from i to j

4.3.2 Constraint

- No team can play against itself:

  1/ x[i][i][k] =0 with : (i = 1, . . . , n, k = 1, . . . , 2n − 2)

- Each team can play at most one game per day (Atmost L):

  2/ $\sum_{j=1}^{n}(x[i][j][k] + x[j][i][k]) = 1$ with:(i = 1, . . . , n, k = 1, . . . , 2n − 2)

- no more than one game involving teams i and j in consecutive rounds k and k+1(no repete).

  $$3/\; x[i][j][k] + x[j][i][k] + x[i][j][k+1] + x[j][i][k+1] \leq 1 \;\; with: (i,j = 1,\dots,n, k = 1,\dots,2n-3)$$

- each team i, on each day k, it plays exactly one game against another team j

  4/ $\sum_{k=1}^{2n-2} x[i][j][k] = 1$ with : (i, j = 1, . . . , n, i != j)

Another decision variable, denoted as zijk, is introduced in the model to represent whether team i plays at home or away against team j on day k. This variable takes the value of 1 if team i plays at home against team j on day k, and 0 otherwise.

In addition, the model includes two sets of constraints(5)(6) to ensure that the number of games played by each team is balanced over time. The first set of constraints limits the sum of the number of games played by team i in the current and previous three days by a predefined value L. Specifically, the sum of z[i][i][k+L] over L=0,1,2,3 is required to be less than or equal to 3, and greater than or equal to 1. These constraints are enforced for each team i and each day k.

5/ $\sum_{L=0}^{3} z[i][i][k+L] \leq 3$ with : (i = 1, . . . , n, k = 1, . . . , 2n − 2 − 3)

6/ $\sum_{L=0}^{3} z[i][i][k+L] \geq 1$ with : (i = 1, . . . , n, k = 1, . . . , 2n − 2 − 3)

The second set of constraints(7)(8) enforces that the number of times each team plays a game (home or away) on each day is the same. Specifically, for each team i and each day k, the number of games played by team i on day k, denoted as z[i][i][k], is equal to the sum of x[i][j][k] over all possible opponents j.

representing the number of times team i plays a game (home or away) on day k.

7/ z[i][i][k] = $\sum_{i=1}^{n} x[i][j][k]$ with: (i = 1, . . . , n k = 1, . . . , 2n − 2)

8/ z[i][j][k] = x[j][i][k] with: (i, j = 1, . . . , n, i 6= j, k = 1, . . . , 2n − 2)

If team t travels from team i to team j on day k, then either team i plays

an away game on day k against another team zijk, or team t plays an away game

on day k against another team ztjk, or both.

9/ y[t][i][j] ≥ z[t][i][k] + z[t][j][k+1] − 1 with: (t, i, j = 1, . . . , n, k = 1, . . . , 2n − 3)

field of values:

→ $x$ijk, $z$ijk, $y$tij ∈ {0, 1} (t, i, j = 1, . . . , n, k = 1, . . . , 2n − 2)

The values of the decision variables xijk, zijk, and tij are restricted to binary values of 0 or 1, which means that a team either plays a game or not, and a travel occurs between two teams or not .

4.4 Hybrid algorithm

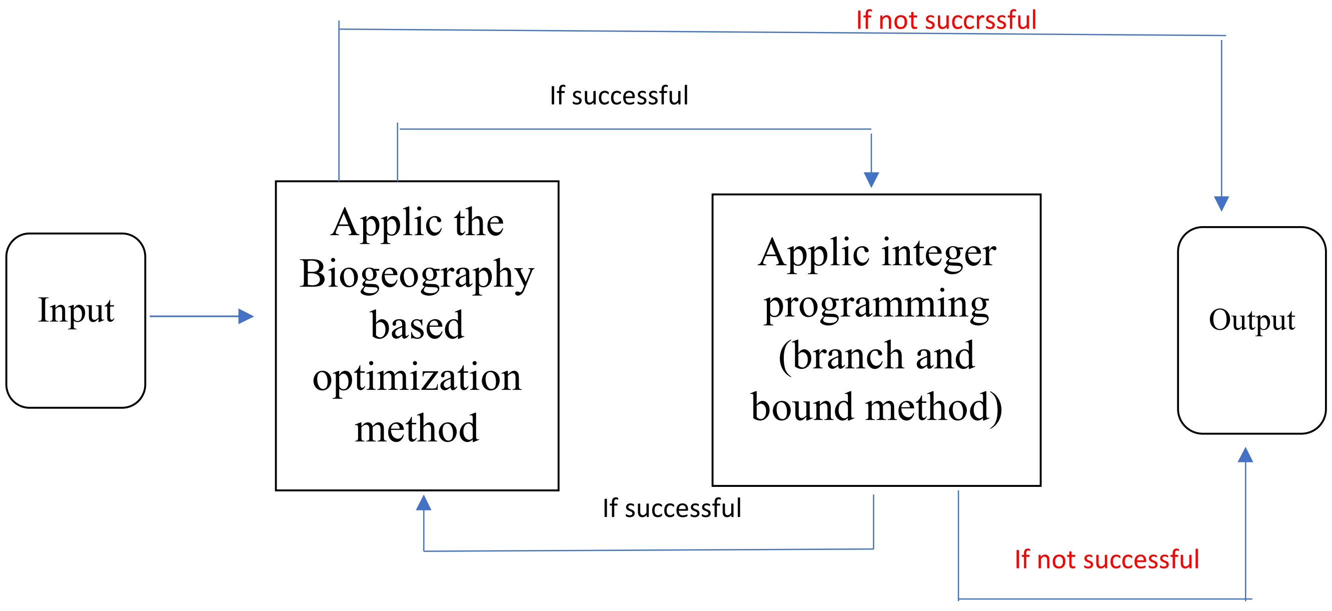

If not succrssful
If successful
Input
Applic the Biogeography based optimization method
Applic integer programming (branch and bound method)
Output
If successful
If not successful

Figure 3.5 :general diagramm

4.4.1 Applic the BBO to solve TTP

In this section, we will provide detailed explanations on how to apply this algorithm to the traveling problem**.**

A)- Generate initiale schedul : Firstly ,The search method begins by establishing an initial configuration that satisfies the DRRT (Double Round Robin Tournament) constraint. To create this configuration, we utilize graph theory modeling in the following manner:

Numbering Vertices: We assign numbers to the vertices of the graph, ranging from 1 to n, where n represents the number of teams. The top n teams are placed at the center of the graph, while the remaining vertices are arranged in a circular fashion around the top n.

First Day Pairings: On the first day, we schedule matches between the following team pairings: (Team 1 and Team n), (Team 2 and Team n-1), (Team 3 and Team n-2), and so on, until the match between (Team n/2 and Team n/2 + 1).

Subsequent Day Pairings: On the subsequent day, we replicate the pairings from the previous day by performing a simple clockwise rotation of the team couplings. This means that each team is matched with the team located one position clockwise from their previous opponent.

By following this approach, we establish an initial configuration for the tournament schedule that adheres to the DRRT constraint.

Secondly, Once the initial Double Round Robin Tournament (DRRT) schedule is generated, we proceed to apply a local search method to identify feasible configurations that satisfy three specific constraints: AtMost, NoRepeat, and DRRT. In this case, we employ Simulated Annealing (SA) as the local search method to construct these feasible schedules.

To guide the SA search, we introduce a cost function that penalizes configurations that violate the aforementioned constraints. The cost function evaluates the quality of a schedule based on the extent to which it adheres to the constraints. Configurations that deviate from the constraints receive higher cost values, indicating their lower desirability.

Furthermore, we leverage neighborhood structures to explore the search space during the SA process. These neighborhood structures define the set of possible modifications or moves that can be applied to a given configuration. By considering different neighborhood structures, we can efficiently navigate the solution space and identify potential improvements.

By combining the cost function, neighborhood structures, and the SA search process, we aim to discover feasible configurations that satisfy the AtMost, NoRepeat, and DRRT constraints for the tournament schedule. The SA algorithm iteratively refines the schedule by making moves that lead to lower costs, eventually converging towards a high-quality feasible solution

We use five neighborhood structures which are detailed in the following :

N1 Swap Home : The Swap Home move (S, ti, tj) is designed to exchange the home/away roles of teams involved in a game. For example, if we consider two teams, ti and tj, and team ti is initially scheduled to play as the home team against team tj in Round k and as the away team against team tj in Round l, the Swap Home move (S, ti, tj) results in a modified schedule, where team ti now plays as the away team against team tj in Round k and as the home team against team tj in Round l. The rest of the schedule remains the same as in S.

Round 1	(4-1)	(3-2)
Round 2	(2-1)	(3-4)
Round 3	(3-1)	(4-2)
Round 4	(1-4)	(2-3)
Round 5	(1-2)	(4-3)
Round 6	(1-3)	(2-4)

The application of the move Swap home away: N1 (S,t4,t1)

Round 1	(4-1)	(3-2)
Round 2	(2-1)	(3-4)
Round 3	(3-1)	(4-2)
Round 4	(1-4)	(2-3)
Round 5	(1-2)	(4-3)
Round 6	(1-3)	(2-4)

N2 Swap Round : The Swap Round move (S, Roundk, Roundl) involves the exchange of all games between a specific pair of rounds. For instance, when applying the Swap Round move to the schedule S, the games originally assigned to Roundk and Roundl are interchanged. This means that all the matchups and corresponding details in Roundk will be moved to Roundl, while the games originally scheduled for Roundl will be moved to Roundk.

Round 1	(4-1)	(3-2)
Round 2	(2-1)	(3-4)
Round 3	(3-1)	(4-2)
Round 4	(1-4)	(2-3)
Round 5	(1-2)	(4-3)
Round 6	(1-3)	(2-4)

After applying the move Swap Round: N2 (S,Round2,Round3):

Round 1	(4-1)	(3-2)
Round 2	(3-1)	(4-2)
Round 3	(2-1)	(3-4)
Round 4	(1-4)	(2-3)
Round 5	(1-2)	(4-3)
Round 6	(1-3)	(2-4)

N3 Swap Team : The Swap Team move (S, ti, tj) involves swapping all opponents of a specific pair of teams throughout all rounds. In other words, when applying the Swap Team move to the schedule S, the opponents originally assigned to team ti will be replaced with the opponents of team tj, and vice versa, across all rounds. This ensures that team ti will face the opponents that were initially scheduled for team tj, and team tj will face the opponents originally designated for team ti.

Round 1	(4-1)	(3-2)
Round 2	(3-1)	(4-2)
Round 3	(2-1)	(3-4)
Round 4	(1-4)	(2-3)
Round 5	(1-2)	(4-3)
Round 6	(1-3)	(2-4)

To facilitate this, we convert the schedul as follows :

	R1	R2	R3	R4	R5	R6
T1	4	3	2	-4	-2	-3
T2	3	4	-1	-3	1	-4
T3	-2	-1	-4	2	4	1
T4	-1	-2	3	1	-3	2

After applying the move Swap Team: N2 (S,Team2,Team3):

	R1	R2	R3	R4	R5	R6
T1	4	3	2	-4	-2	-3
T2	-2	-1	-4	2	4	1
T3	3	4	-1	-3	1	-4
T4	-1	-2	3	1	-3	2

N4 Swap Teams Partial : The proposed action involves swapping the opponents of teams t1 and t2 on a specific round, which may result in additional swaps being necessary to resolve any conflicts that arise as a consequence.

	R1	R2	R3	R4	R5	R6	R7	R8	R9	R10
T1	6	-2	4	3	-5	-4	-3	5	2	-6
T2	5	1	-3	-6	4	3	6	-4	-1	-5
T3	-4	5	2	-1	6	-2	1	-6	-5	4
T4	3	6	-1	-5	-2	1	5	2	-6	-3
T5	-2	-3	6	4	1	-6	-4	-1	3	2
T6	-1	-4	-5	2	-3	5	-2	3	4	1

The move Partial Swap Teams(S,T2,T4,R9) produces the schedule

	R1	R2	R3	R4	R5	R6	R7	R8	R9	R10
T1	6	-2	2	3	-5	-4	-3	5	4	-6
T2	5	1	-1	-5	4	3	6	-4	-6	-3
T3	-4	5	4	-1	6	-2	1	-6	-5	2
T4	3	6	-3	-6	-2	1	5	2	-1	-5
T5	-2	-3	6	2	1	-6	-4	-1	3	4
T6	-1	-4	-5	4	-3	5	-2	3	2	1

N5 Swap Rounds Partial : The suggested action is to swap the opponents of team t on rounds r1 and r2. As a result, additional swaps will be required between these rounds to ensure the schedule remains feasible. These subsequent swaps are necessary to maintain the practicality and coherence of the schedule.

	R1	R2	R3	R4	R5	R6	R7	R8	R9	R10
T1	6	-2	2	3	-5	-4	-3	5	4	-6
T2	5	1	-1	-5	4	3	6	-4	-6	-3
T3	-4	5	4	-1	6	-2	1	-6	-5	2
T4	3	6	-3	-6	-2	1	5	2	-1	-5
T5	-2	-3	6	2	1	-6	-4	-1	3	4
T6	-1	-4	-5	4	-3	5	-2	3	2	1

Obviously swapping the game in rounds r2 and r9 would not lead to a round-robin tournament. It is also necessary to swap the games of team 1, 4, and 6 in order to obtain:

	R1	R2	R3	R4	R5	R6	R7	R8	R9	R10
T1	6	4	2	3	-5	-4	-3	5	-2	-6
T2	5	-6	-1	-5	4	3	6	-4	1	-3
T3	-4	5	4	-1	6	-2	1	-6	-5	2
T4	3	-1	-3	-6	-2	1	5	2	6	-5
T5	-2	-3	6	2	1	-6	-4	-1	3	4
T6	-1	2	-5	4	-3	5	-2	3	-4	1

B)- generate intial population :

The following is a list with a length=n where each element represents a candidate solution (schedul). We iterate the SA local search on the initial schedul, and whenever we obtain a table that satisfies the three conditions, we add it to the list of candidates solutions.

C)- Migration process and mutation :

We calculate the cost for each candidate solution and then sort the list of solutions from best to worst. Afterwards, we introduce a process called migration, where we randomly select a game from the best solution and aim to incorporate that game into the worst solution. This is done by applying a swap using the partial swap team (PST) neighborhood until we obtain a new table that contains one characteristic from the best solution. This process is known as migration.

Following that, we execute a random mutation (either it happens or not based on a randomly assigned factor) to introduce further variations in the solution.

Then, the process of calculating costs, sorting the table, performing migration, and executing mutation is repeated until we find that the best solution When the proposed set of solutions becomes entirely symmetrical. At this point, we terminate the execution of this algorithm.

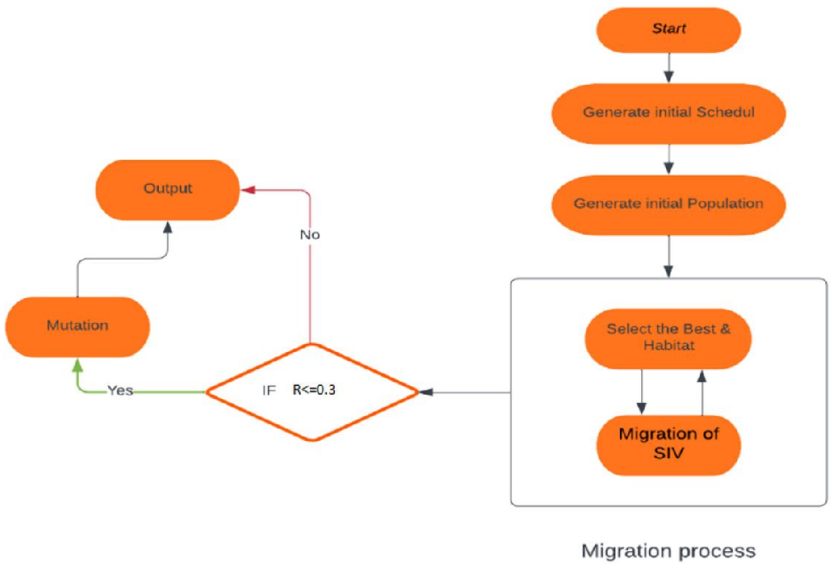


Figure 3-5-2 :Biogeography based optimisation algorithm

4.4.2 Apply method exact (branch and bound)

In the previous parts, we talk about integer programming of TTP. However, the constraints mentioned earlier are not sufficient to apply Branch & bound algorithm directly to the TTP, as it is a problem for which finding the optimal solution is not feasible. Therefore, we added an additional constraint to reduce the search space.

The added constraint is as follows :

- Select a random team i.
- Set the lower bound and upper bound of the decision variable X[i][j][k] to Xb[i][j][k].
- Iterate over all teams j (j = 1, ..., n) and rounds k (k = 1, ..., 2n-2).

Xb represents an input schedule .

By applying this constraint, we aim to fix all the matches of team i, ensuring that it plays against specific opponents in specific rounds. This helps to narrow down the search space and improve the efficiency of the integer programming approach.

It's important to note that this additional constraint is introduced based on our understanding of the problem and our attempts to find a feasible solution. It may vary depending on the specific formulation and approach used. The goal is to strike a balance between computational efficiency and solution quality.

By incorporating this constraint, we aim to enhance the integer programming approach for solving the TTP. However, further research and experimentation are needed to explore alternative formulations and approaches that can improve the results, especially for larger TTP instances.

The Branch-and-Bound algorithm, on the other hand, is a strong framework that is frequently used to resolve combinatorial optimization issues with discrete variables. The search space is divided into smaller branches, and the objective function is constrained within each branch. The algorithm effectively reduces the search space and removes suboptimal solutions by iteratively exploring various branches and evaluating lower and upper bounds. In relation to the TTP, the Branch-and-Bound algorithm enables us to navigate the vast solution space and find high-quality solutions. While the integer programming approach is enhanced by the additional constraint, further research and experimentation are needed to explore alternative formulations and approaches that can improve the results, particularly for larger TTP instances [3].

Implementation of Branch-and-Bound:

The Branch-and-Bound algorithm is a versatile approach employed for solving complex combinatorial optimization problems. Specifically, in the case of the TTP, the algorithm can be effectively utilized in the following manner:

a) Conversion to Binary:

First, the initial schedule table is converted into a binary format. This conversion is necessary to represent the schedule as a set of decision variables that can be subjected to constraints and used to calculate the objective function. Each entry in the table is transformed into a binary variable that represents whether a match between teams i and j occurs in round k.

b) Constraint Application and Objective Function Calculation:

With the binary representation, the algorithm applies the necessary constraints to the decision variables. These constraints include the rules of the TTP, such as ensuring that teams play against different opponents in each round and limiting the number of games per team per day. Additionally, the objective function, which measures the quality of the schedule, is calculated based on the binary variables.

c) Optimization of the Schedule:

Using the initial binary schedule as input, the algorithm proceeds to optimize it by iteratively exploring different branches of the search space. It partitions the search space into subsets based on branching rules and calculates lower and upper bounds for each subset. This allows the algorithm to discard suboptimal solutions and focus on promising branches that have the potential to yield better schedules.

d) Conversion to Ordinary Schedule:

After optimizing the binary schedule, the resulting solution is converted back to its ordinary representation, where matches are represented by team IDs rather than binary variables. This converted schedule is then returned as the output of the algorithm.

The implementation of the Branch-and-Bound algorithm for the TTP involves a combination of binary conversion, constraint application, objective function calculation, optimization through branch exploration, and conversion back to the original schedule representation. Through this process, the algorithm aims to find high-quality schedules that adhere to the TTP rules while considering computational efficiency.

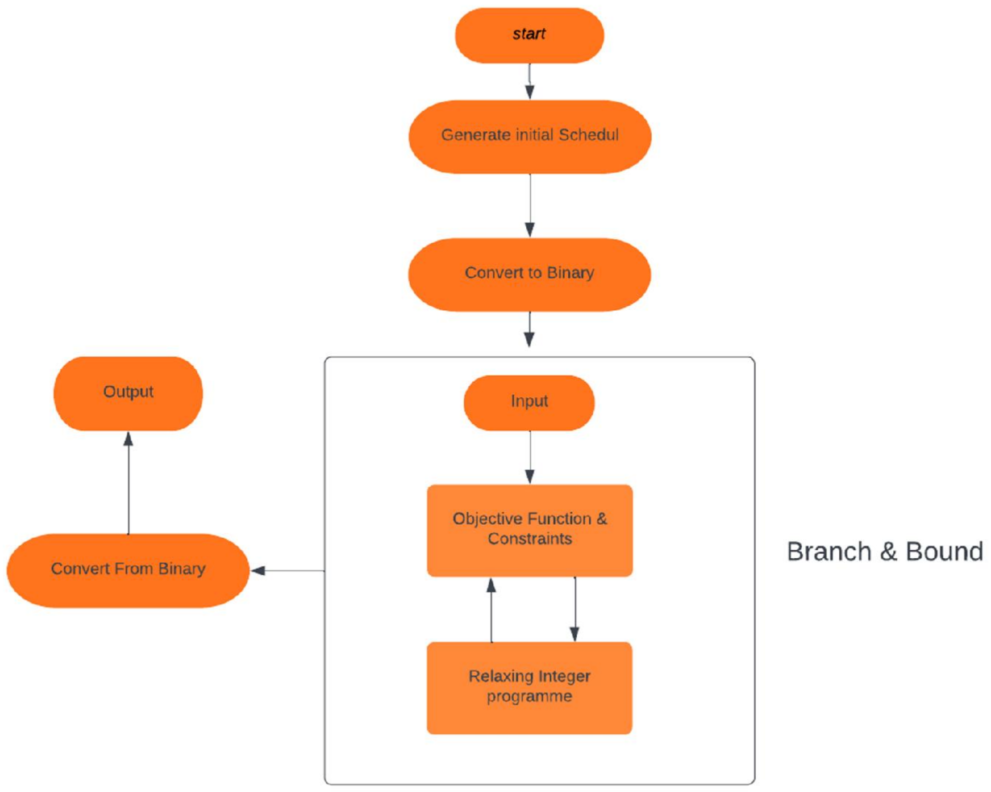


Figure 3-5-3 : represents Branch-and-Bound implementation.

4.4.3 Combination

The combined approach of Branch-and-Bound (B&B) and Biogeography-Based Optimization (BBO) involves multiple steps to tackle the optimization problem at hand. Here is an overview of the process:

Generating the initial schedule: The first step is to generate an initial schedule as a starting point for the optimization process. This schedule serves as a baseline.

Building the population in BBO: In the BBO phase, a population of candidate solutions is created. Each solution represents a potential schedule. The population undergoes optimization through iterations, where the solutions are improved based on BBO principles such as migration and mutation. The aim is to enhance the diversity and quality of the solutions.

Stopping criteria in BBO: The BBO process continues until a maximum number of iterations is reached or when all the solutions in the population become similar. This indicates that further improvement is unlikely, and the BBO phase is halted.

Capturing the best solution: At this point, the best solution obtained from BBO, which represents a potentially good schedule, is captured. This solution is then passed to the Branch-and-Bound algorithm for further refinement.

Refinement using Branch-and-Bound: The Branch-and-Bound algorithm is employed to refine the schedule obtained from BBO. It systematically explores different branches of the search space, narrowing down the possibilities and improving the objective function value. The process continues until no further improvement is observed.

Iteration between BBO and Branch-and-Bound: If the refinement using Branch-and-Bound results in improvement, the process goes back to the BBO phase and repeat the optimization cycle. This iteration continues until a stopping criterion is met, such as reaching a maximum number of iterations or no improvement.

By combining the strengths of BBO in generating diverse solutions and Branch-and-Bound in refining those solutions, this approach offers a powerful framework to tackle the optimization problem and potentially converge to high-quality schedules.

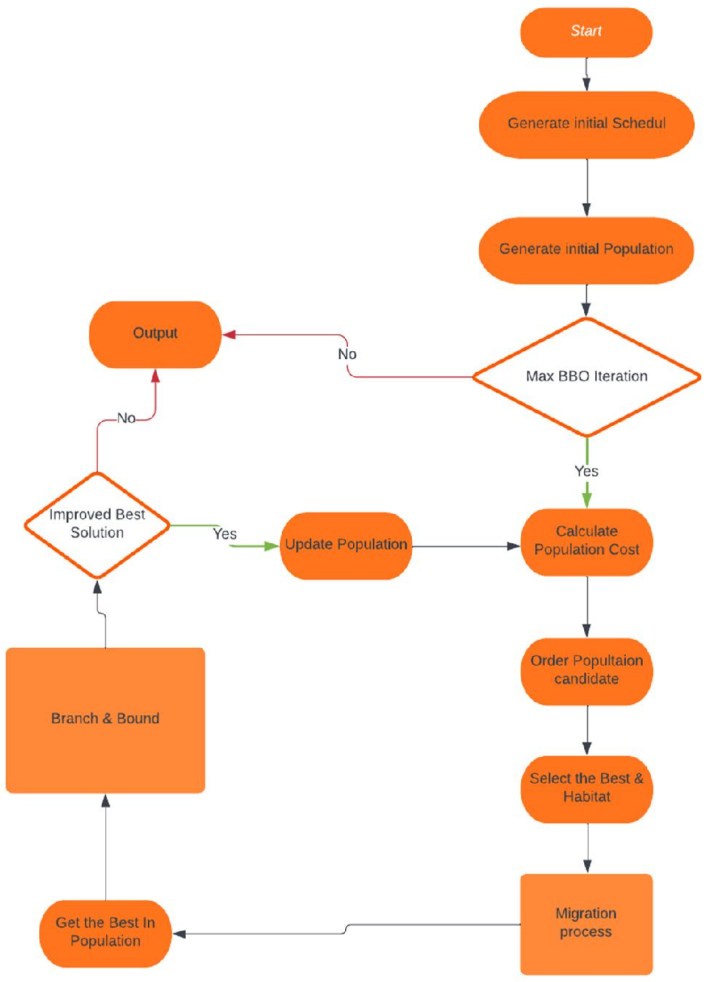


Figure 3-5-4 : represents the Combined BBO-Branch-and-Bound Approach.

4.5 Conclusion

In this chapter, we have explained the Hybrid method that we proposed . We started by introducing and presenting the integer program of TTP. Then, we elaborated on how we applied BBO for the first time and delved into explaining the swap methods. Finally, we presented the steps of the hybridization between the two approaches. In the next chapter, we will examine the results and perform an analysis.

CHAPTER 5 : Numerical Results

Chapter 5 : Numerical Results

5.1 Introduction

In this final chapter, we will delve into the presentation and analysis of the numerical results obtained through the implementation of the proposed approach. The approach we employed is based on a combination of two methods: the approximate method known as Biogeography-Based Optimization (BBO) and the more precise Integer Programming (IP) approach.

5.2 Software Libraries and Frameworks

In this section, we will discuss the software libraries and frameworks utilized in our implementation of the proposed approach for solving the Traveling Tournament Problem (TTP).

For the implementation of the Integer Programming (IP) method, we leveraged the power and functionality of the Python programming language. Python offers a wide range of libraries and tools that facilitate mathematical optimization tasks, making it a popular choice for tackling combinatorial optimization problems like the TTP , One of the key libraries we employed is PuLP, which stands for "Python Library for Mathematical Programming." PuLP provides a high-level interface for constructing mathematical models and formulating optimization problems. It supports a variety of solvers, including commercial solvers like CPLEX, as well as open-source solvers like GLPK and CBC.[2]

By utilizing PuLP, we were able to express the TTP problem as an Integer Programming model in a concise and intuitive manner. We defined decision variables, objective functions, and constraints using the PuLP syntax, allowing us to formulate the problem accurately.

Additionally, PyCharm IDE (Integrated Development Environment) served as our primary development environment for writing, debugging, and executing the Python code. PyCharm offers a range of features that enhance productivity, such as code completion, debugging tools, and version control integration.

The combination of Python as the programming language, PuLP as the optimization library, and PyCharm as the development environment provided us with a powerful and efficient platform for implementing and experimenting with the proposed approach for solving the TTP.

5.3 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, Average, CPU Times, and Gap %. 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 Average column reflects the consistency of our method, and the CPU Times column indicates the computational efficiency. The Gap % column quantifies the deviation from the best-known solution. By analyzing these results, we can evaluate the effectiveness of our approach in finding high-quality solutions for the TTP instances.

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, Average, CPU Times, and Gap %. 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. The Average column helps evaluate the consistency of our approach, and the CPU Times column provides information about the computational efficiency. The Gap % column quantifies the difference 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.

Chapter 5 : Numerical Results

We compared the results of this new method with the cumulative results of recent studies[34] [35] [36], 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:

	Team1	Team2	Team3	Team4
Team1	0	745	665	929
Team2	745	0	80	337
Team3	665	80	0	380
Team4	929	337	380	0

Instance	Best Known	Initial Solutions	Best Solution	Average	CPU Times (s)	Gap %
CON4	17	30	**17**	17	0.1	0
CON6	43	82	**43**	43	20	0
CON8	80	148	**80**	80	33	0
CON10	124	186	**124**	124	70	0
Galaxy4	416	477	**416**	416	60	0
Galaxy6	1365	1578	**1365**	1383	180	0
Galaxy8	2373	3325	2473	2531	1055	-2.038
Galaxy10	3676	6481	4933	5105	2001	-34.194
CIRC4	20	26	**20**	20	30	0
CIRC6	64	80	**64**	64	170	0
CIRC8	130	188	142	150	422	-9.230
NL4	8276	8413	**8276**	8276	20	0
NL6	23916	26893	24073	24892	350	-0.656
NL8	39947	52487	43141	43182	644	-7.995
NL10	59583	84116	66357	66857	921	-11.369
Super4	63405	65305	**63405**	63405	1203	0

Table 4-3: Numerical results found by the overall approach

5.4 Algorithm Performance Representation

-Index is number of scheduls
-The cost represents the distance traveled by the teams .

- For exploration and exploitation

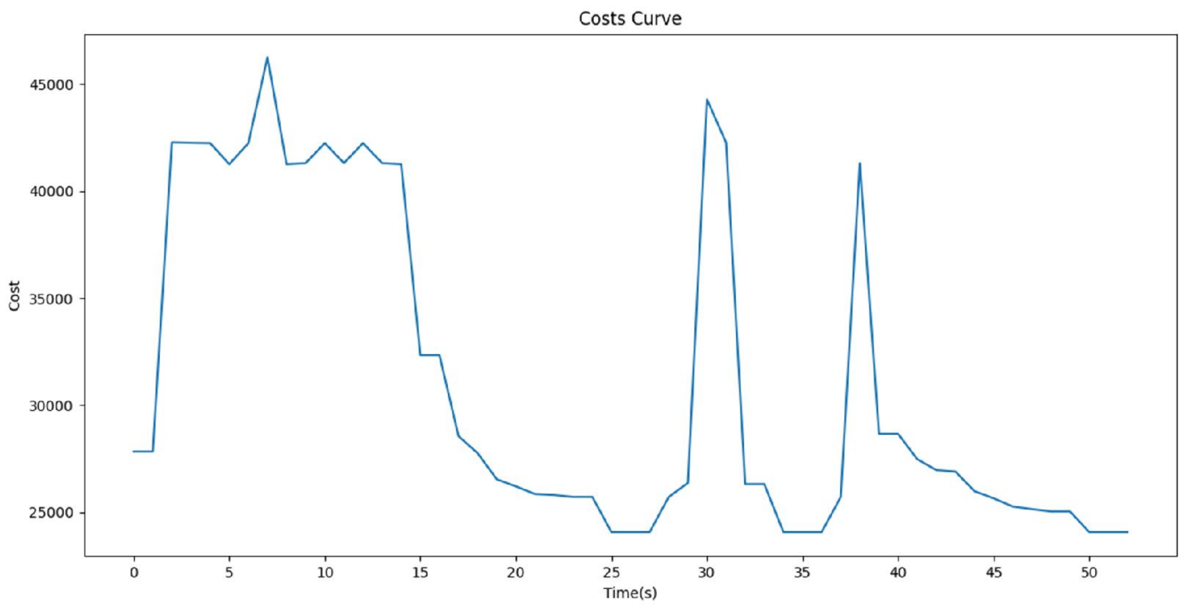


Figure 4.4.1 :NL6 performance

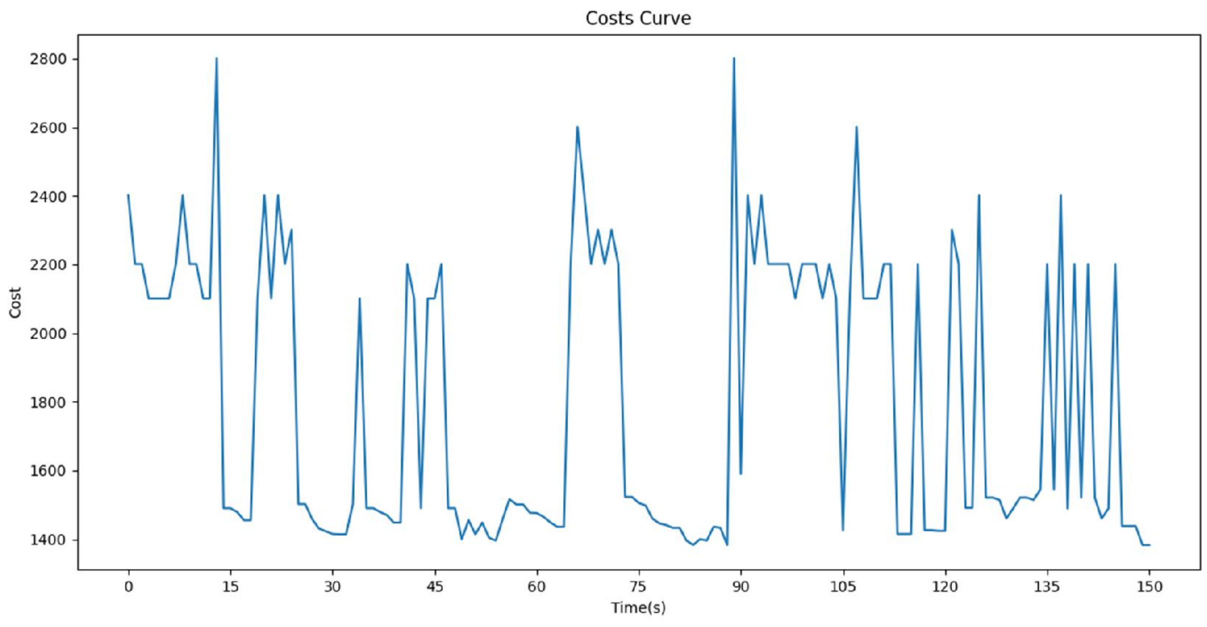


Figure 4.4.2 : GALAXY 6 performance

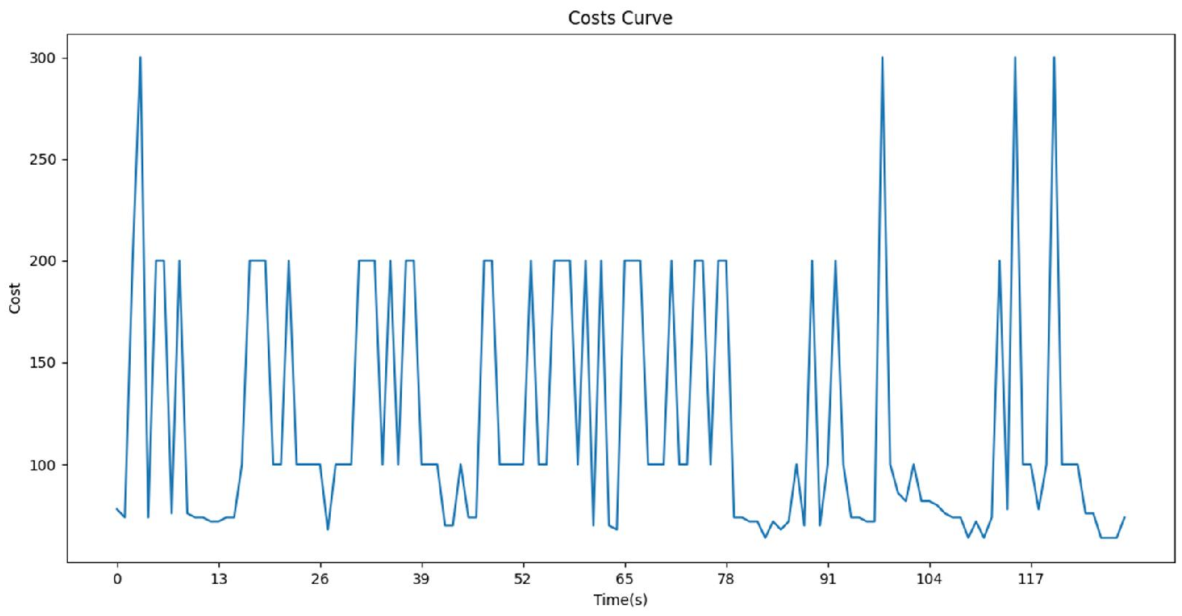
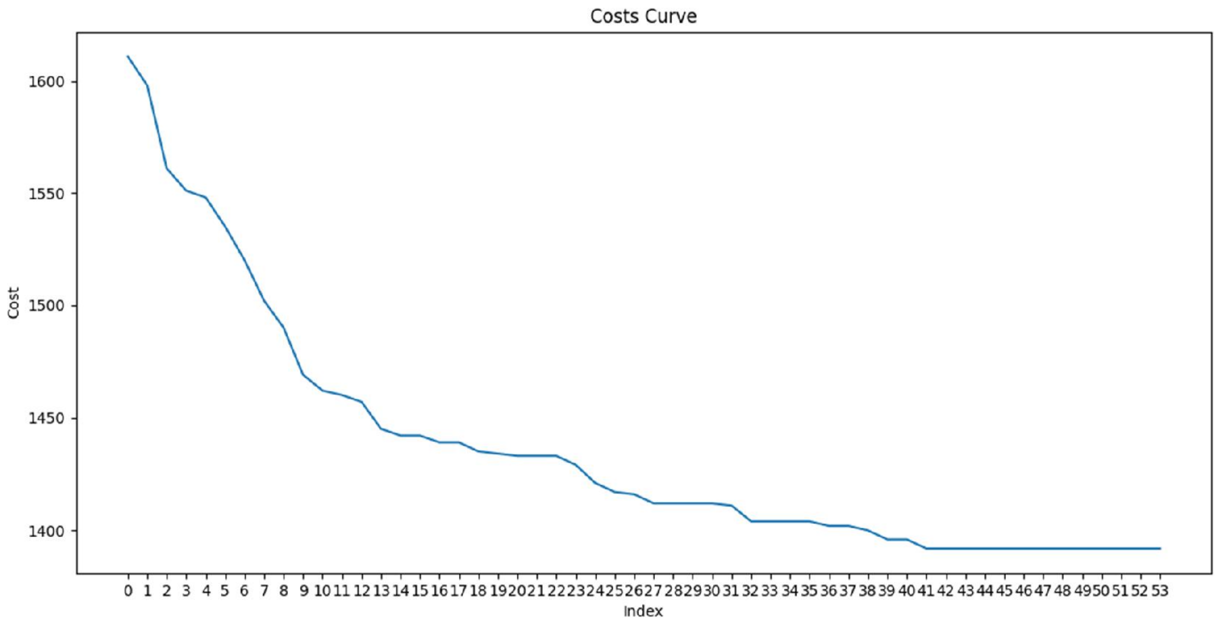
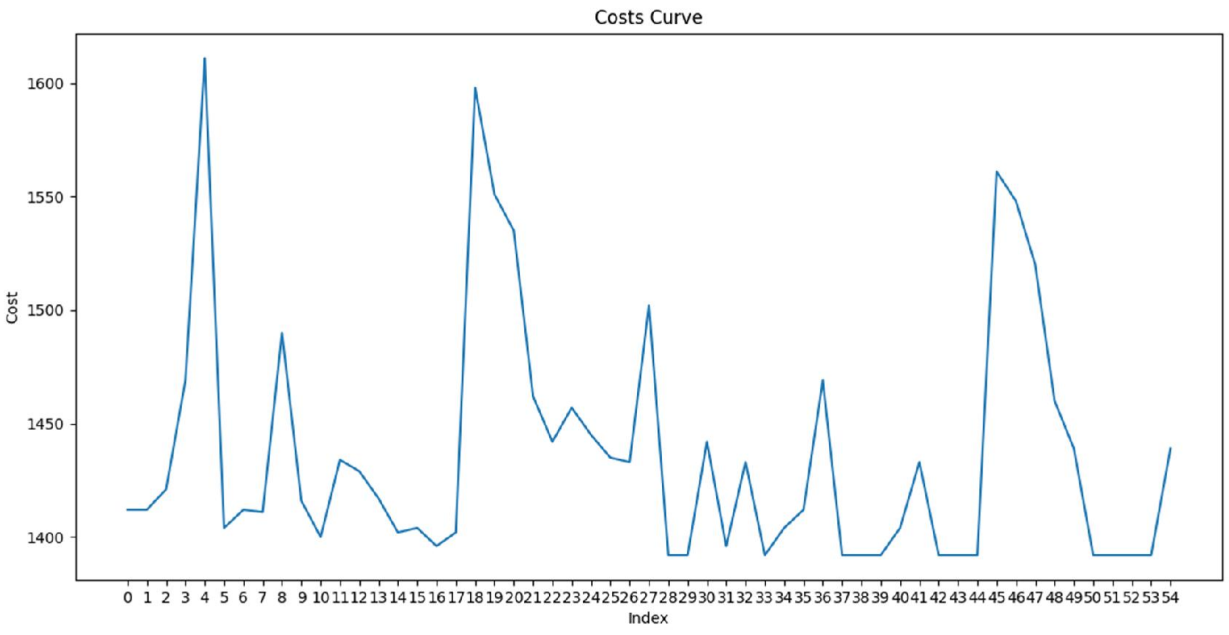


Figure 4.4.3 :CIRC 6 performance

- Regarding optimizing the distance :



- the impact of mutation on the diversity of exploration in the search space



5.4 Limitations

In the context of the Traveling Tournament Problem (TTP), we have observed certain limitations and identified future directions for improvement. One notable limitation is that the integer programming formulation used in our approach tends to exhibit high computational times when dealing with larger instances, particularly those involving more than 10 teams. Despite our efforts, we have not observed significant improvements in the runtime performance of the integer programming model for these larger benchmarks.

5.5 Conclusion

In summary, although our current approach using integer programming has encountered challenges with larger TTP benchmarks, we are optimistic about the future. Our future directions involve enhancing the performance of the integer programming model, with a specific focus on improving runtime for larger instances. Through relaxation techniques and other innovative approaches, we aim to overcome the current limitations and offer more efficient solutions for the Traveling Tournament Problem.

General conclusion

The Traveling Tournament Problem (TTP) is a difficult optimization problem that has a wide range of practical applications. We have given an overview of the TTP, examined its difficulty, looked at several methods and formulations for solving it, and spoken about the importance of the problem in terms of sports scheduling and optimization throughout this work. We have highlighted the advantages and disadvantages of the numerical findings produced using the Combined BBO & Branch-and-Bound Approach. Our future directions will focus on enhancing the solutions' effectiveness and scalability, particularly in addressing bigger TTP situations. We intend to improve TTP research and its useful applications in several sectors by tackling these difficulties.

As we look to the future, we aim to address this limitation by exploring alternative strategies to enhance the efficiency of the integer programming formulation. One computational efficiency. Such efforts will contribute to expanding the applicability of the integer programming approach to tackle larger and more complex TTP instances.

References

[1] B. Korte and J. Vygen, *Combinatorial Optimization: Theory and Algorithms*, vol. 21. in Algorithms and Combinatorics, vol. 21. Berlin, Heidelberg: Springer, 2012. doi: 10.1007/978-3-642-24488-9.

[2] “Programming of Interdependent Activities: I General Discussion on JSTOR.” https://www.jstor.org/stable/1905522 (accessed Apr. 11, 2023).

[3] “A Dynamic Programming Approach to Sequencing Problems | SIAM Journal on Applied Mathematics.” https://epubs.siam.org/doi/abs/10.1137/0110015?journalCode=smjmap.1 (accessed Apr. 11, 2023).

[4] R. E. Bixby, S. Ceria, C. M. McZeal, and M. W. P. Savelsbergh, “An Updated Mixed Integer Programming Library: MIPLIB 3.0,” Feb. 1998, Accessed: Apr. 11, 2023. [Online]. Available: https://scholarship.rice.edu/handle/1911/101898

[5] P. M. Pardalos and M. G. C. Resende, *Handbook of Applied Optimization*. Oxford University Press, 2002.

[6] L. A. Wolsey and G. L. Nemhauser, *Integer and Combinatorial Optimization*. John Wiley & Sons, 1999.

[7] F. Glover, “Tabu Search and Adaptive Memory Programming — Advances, Applications and Challenges,” in *Interfaces in Computer Science and Operations Research*, R. S. Barr, R. V. Helgason, and J. L. Kennington, Eds., in Operations Research/Computer Science Interfaces Series, vol. 7. Boston, MA: Springer US, 1997, pp. 1–75. doi: 10.1007/978-1-4615-4102-8_1.

[8] “How to Solve It: Modern Heuristics - Zbigniew Michalewicz, David B. Fogel - كتب Google.” https://books.google.dz/books?hl=ar&lr=&id=MpKqCAAAQBAJ&oi=fnd&pg=PA1&dq=+How+to+Solve+It:+Modern+Heuristics.&ots=s_lYkZ_Paf&sig=TTTSaTvzDshXc07V6kEkHLJsUcY&redir_esc=y#v=onepage&q=How%20to%20Solve%20It%3A%20Modern%20Heuristics.&f=false (accessed Apr. 12, 2023).

[9] E.-G. Talbi, *Metaheuristics: From Design to Implementation*. John Wiley & Sons, 2009.

[10] “Tabu Search—Part I | ORSA Journal on Computing.” https://pubsonline.informs.org/doi/abs/10.1287/ijoc.1.3.190 (accessed Apr. 12, 2023).

[11] E. L. Lawler, “THE TRAVELING SALESMAN PROBLEM : A GUIDED TOUR OF COMBINATORIAL OPTIMIZATION,” *WILEY-Intersci. Ser. DISCRETE Math.*, 1985, Accessed: Apr. 11, 2023. [Online]. Available: https://trid.trb.org/view/229226

[12] F. Glover, “Future paths for integer programming and links to artificial intelligence,” *Comput. Oper. Res.*, vol. 13, no. 5, pp. 533–549, Jan. 1986, doi: 10.1016/0305-0548(86)90048-1.

[13] C. H. Papadimitriou and K. Steiglitz, *Combinatorial Optimization: Algorithms and Complexity*. Courier Corporation, 1998.

[14] G. L. Nemhauser, L. A. Wolsey, and M. L. Fisher, “An analysis of approximations for maximizing submodular set functions—I,” *Math. Program.*, vol. 14, no. 1, pp. 265–294, Dec. 1978, doi: 10.1007/BF01588971.

[15] F. Glover and M. Laguna, *Tabu search I*, vol. 1. 1999. doi: 10.1287/ijoc.1.3.190.

[16] V. V. Vazirani, *Approximation Algorithms*. Berlin, Heidelberg: Springer, 2003. doi: 10.1007/978-3-662-04565-7.

[17] C. Blum and A. Roli, “Metaheuristics in combinatorial optimization: Overview and conceptual comparison,” *ACM Comput. Surv.*, vol. 35, no. 3, pp. 268–308, Sep. 2003, doi: 10.1145/937503.937505.

[18] J. Nocedal and S. J. Wright, *Numerical optimization*, 2nd ed. in Springer series in operations research. New York: Springer, 2006.

[19] R. E. Bellman, “Dynamic Programming,” in *Dynamic Programming*, Princeton University Press, 2021. doi: 10.1515/9781400835386.

[20] D. Bertsekas, *Dynamic Programming and Optimal Control: Volume I*. Athena Scientific, 2012.

[21] R. C. Daniel and R. J. Vanderbei, “Linear Programming: Foundations and Extensions.,” *J. Oper. Res. Soc.*, vol. 49, no. 1, p. 94, Jan. 1998, doi: 10.2307/3010658.

[22] A. Hasbani, “Contribution a la conception d’un environnement de specification formelle et de validation des systemes reactifs.” France, 1997.

[23] D. B. Fogel, “An introduction to simulated evolutionary optimization,” *IEEE Trans. Neural Netw.*, vol. 5, no. 1, pp. 3–14, Jan. 1994, doi: 10.1109/72.265956.

[24] P. K. Roy, S. P. Ghoshal, and S. S. Thakur, “Biogeography-based Optimization for Economic Load Dispatch Problems,” *Electr. Power Compon. Syst.*, vol. 38, no. 2, pp. 166–181, Dec. 2009, doi: 10.1080/15325000903273379.

[25] 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, doi: 10.1016/j.catena.2018.12.033.

[26] 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.

[27] M. K. Ahirwar, P. K. Shukla, and R. Singhai, “CBO-IE: A Data Mining Approach for Healthcare IoT Dataset Using Chaotic Biogeography-Based Optimization and Information Entropy,” *Sci. Program.*, vol. 2021, p. e8715668, Oct. 2021, doi: 10.1155/2021/8715668.

[28] J. Taghizadeh, M. Ghobaei-Arani, and A. Shahidinejad, “An efficient data replica placement mechanism using biogeography-based optimization technique in the fog computing environment,” *J. Ambient Intell. Humaniz. Comput.*, vol. 14, no. 4, pp. 3691–3711, Apr. 2023, doi: 10.1007/s12652-021-03495-0.

[29] D. Chatterjee and B. K. Roy, “An Improved Scheduling Algorithm for Traveling Tournament Problem with Maximum Trip Length Two.” Sep. 19, 2021. doi: 10.4230/OASIcs.ATMOS.2021.16.

[30] G. B. Dantzig and J. H. Ramser, “The Truck Dispatching Problem,” *Manag. Sci.*, vol. 6, no. 1, pp. 80–91, Oct. 1959, doi: 10.1287/mnsc.6.1.80.

[31] A. Anagnostopoulos, L. Michel, P. V. Hentenryck, and Y. Vergados, “A simulated annealing approach to the traveling tournament problem,” *J. Sched.*, vol. 9, no. 2, pp. 177–193, Apr. 2006, doi: 10.1007/s10951-006-7187-8.

[32] K. Easton, G. Nemhauser, and M. Trick, “Solving the Travelling Tournament Problem: A Combined Integer Programming and Constraint Programming Approach,” Aug. 2002, pp. 100–112. doi: 10.1007/978-3-540-45157-0_6.

[33] A. Anagnostopoulos, L. Michel, P. V. Hentenryck, and Y. Vergados, “A simulated annealing approach to the traveling tournament problem,” *J. Sched.*, vol. 9, no. 2, pp. 177–193, Apr. 2006, doi: 10.1007/s10951-006-7187-8.

[34] N. Frohner, B. Neumann, and G. R. Raidl, “A Beam Search Approach to the Traveling Tournament Problem,” in *Evolutionary Computation in Combinatorial Optimization*, in Lecture Notes in Computer Science. Cham: Springer International Publishing, 2020, pp. 67–82. doi: 10.1007/978-3-030-43680-3_5.

[35] J. Zhao, M. Xiao, and C. Xu, “Improved Approximation Algorithms for the Traveling Tournament Problem,” in *47th International Symposium on Mathematical Foundations of Computer Science (MFCS 2022)*, S. Szeider, R. Ganian, and A. Silva, Eds., in Leibniz International Proceedings in Informatics (LIPIcs), vol. 241. Dagstuhl, Germany: Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022, p. 83:1-83:15. doi: 10.4230/LIPIcs.MFCS.2022.83.

[36] M. Khelifa, D. Boughaci, and E. Aïmeur, “A new approach based on graph matching and evolutionary approach for sport scheduling problem,” *Intell. Decis. Technol.*, vol. 14, no. 4, pp. 565–580, Jan. 2020, doi: 10.3233/IDT-190114.

.