République Algérienne Démocratique et Populaire
Ministère de l'Enseignement Supérieur et de la Recherche Scientifique
Université Kasdi Merbah - Ouargla
Faculté des Nouvelles Technologies de l'Information et de la Communication
Département d'Informatique et Technologie de l'Information

Mémoire de fin d'études
Pour l'obtention du diplôme du Master en Informatique
Option : Informatique Fondamentale

Réalisé par :

BENYAYA SOUMIA **BOULAHDJAR KHADIDJA**

Une nouvelle approche basée sur les heuristiques de graphe et une approche évolutionnaire pour le problème de compétition sportif

Soutenu publiquement le : 18/ 06/ 2022

Devant le jury composé de :

Dr. MERIEM KHELIFA	**UKM OUARGLA**	**Encadreur**
Dr. RANDA BENKHELIFA	**UKM OUARGLA**	**Président**
Dr. ABDELHAKIM CHERIET	**UKM OUARGLA**	**Examinateur**

Promotion : 2021/2022

Contents

List of Figures

List of Tables

Acknowledgement

First of all, we thank “God” who gave us the wisdom and health to do this modest work. We thank our supervisor Ms khalifa merieme for her guidance and supervision in the preparation of this work. Our thanks also go to all members of the jury for having accepted to honor our work with their judgment, and for their jury participation

Dedication

I dedicate this modest work:
To my parents, for their encouragement
To my husband ABDOU I really appreciate his encouragement and his help.
To my daughter djoumana rahaf
To all my big family.
To my supervisor khalifa merieme
To my dear friend and partner soumia
To my favorite teatcher SAADI WAFA

KHADIDJA

Dedication

I dedicate this work to my familly, parents , brothers expecially
DERRAR and SAIF who i wish the best for them , without
forgetting my binom collegue and sister KHADIDJA
To my teacher supervisor MERIEME KHELIFA
To my teatcher SAADI WAFA
To all those who help and motivate me especially AMIRA ,SALAH ,
BELKACEM ,Kawther ,nassom , WAHIBA, LAYLA
To all my big familly
To my self because I suffered a lot lo live this day and I think that I
deserve this success

SOUMIA

Abstract

In this work, we are interested in the Traveling Tournament Problem (TTP). TTP is a challenging combinatorial optimization problem in sports scheduling that aims to construct a double round-robin tournament schedule minimizing the total distance traveled by the teams and satisfying, at the same time, the TTP-specific constraints. TTP is a well-known and important problem within the collective sports communities since poor optimization in TTP can cause losses in the budget of managing the leagues. We proposed an original enhanced Genetic Algorithm(E-GA) to solve TTP. We used a local search as a subroutine to improve the intensification mechanism in GA. Further, We propose a new technique called the aspiration technique to filter the search space and keep only the feasible configurations. The aspiration technique allows memorizing information on moves leading to feasible neighbor configurations, starting from a current configuration. The overall method is evaluated on publicly available standard benchmarks and compared with other techniques for TTP. The computational experiment shows that the proposed method could build interesting results comparable to other state-of-the-art approaches

Keywords: traveling tournament problem ,combinatorial optimization , Genetic Algorithm ,local search.

Résumé

Dans ce travail, nous nous intéressons au problème bien connu Problème de Compétition Sportif(TTP). Le TTP est un problème d'optimisation combinatoire difficile dans la programmation sportive qui vise à construire un calendrier de tournois à double ronde minimisant la distance totale parcourue par les équipes et satisfaisant, en même temps, les contraintes spécifiques au TTP. Le TTP est un problème bien connu et important au sein des communautés sportives collectives car une mauvaise optimisation du TTP peut entraîner des pertes dans le budget de gestion des ligues. Nous avons proposé un algorithme génétique amélioré original (E-GA) pour résoudre le TTP. Nous avons utilisé une recherche de voisinage variable comme sous-programme pour améliorer le mécanisme d'intensification dans GA. De plus, nous proposons une nouvelle technique appelée la technique d'aspiration pour filtrer l'espace de recherche et ne garder que les configurations réalisables. La technique d'aspiration permet de mémoriser des informations sur les mouvements conduisant à des configurations de voisinage réalisables, à partir à partir d'une configuration actuelle. La méthode globale est évaluée sur des références standard accessibles au public et comparée à d'autres techniques de TTP. L'expérience informatique montre que la méthode proposée pourrait générer des résultats intéressants comparables à d'autres approches de pointe.
Mots clés :compétitions sportives,optimisation combinatoire ,l'algorithme génétique, la recherche locale.

ملخص

في هذا العمل ، نحن مهتمون بمشكلة بطولة السفر TTP.(TTP) هي مشكلة تحسين اندماجية صعبة في جدولة الألعاب الرياضية التي تهدف إلى إنشاء جدول دورات مزدوج الجولة لتقليل المسافة الإجمالية التي تقطعها الفرق وإرضاء القيود الخاصة بـ TTP في نفس الوقت. تعتبر TTP مشكلة معروفة ومهمة داخل مجتمعات الرياضات الجماعية لأن ضعف التحسين في TTP يمكن أن يتسبب في خسائر في ميزانية إدارة البطولات. اقترحنا خوارزمية جينية محسنة أصلية ($E-GA$) لحل TTP . استخدمنا بحث حي متغير كإجراء فرعي لتحسين آلية التكثيف في GA . علاوة على ذلك ، نقترح تقنية جديدة تسمى تقنية الشفط لتصفية مساحة البحث والاحتفاظ بالتكوينات الممكنة فقط. تسمح تقنية الطموح بحفظ المعلومات حول التحركات التي تؤدي إلى تكوينات الجوار الممكنة ، بدءًا من التكوين الحالي. يتم تقييم الطريقة الشاملة بناءً على معايير معيارية متاحة للجمهور ومقارنتها بالتقنيات الأخرى الخاصة بـ TTP . تظهر التجربة الحسابية أن الطريقة المقترحة يمكن أن تؤدي إلى نتائج مثيرة يمكن مقارنتها بأحدث الأساليب الأخرى .

الكلمات المفتاحية : مشكلة في بطولة السفر ، الاتحسين الأمثل ، الخوارزمية الجينية ،بحث المحلي .

Introduction

Many important and practical problems can be expressed as optimization problems. Such problems involve finding the best of an exponentially large set of solutions. The obvious algorithm, considering each of the solutions, takes too much time because there are so many solutions. The optimization problem can be solved in an exact method or an approximate method. The exact method provides optimal solutions in a long time because it cannot solve NP-complete problems. As for approximate algorithms (or heuristic), they give high-quality solutions in a reasonable time, but there is no guarantee of finding a global optimal solution. In this work, we are interested in the well-known NP-hard traveling tournament problem (TTP); TTP is an interesting problem in both sports scheduling and combinatorial optimization. It is known to be an NP-hard problem which makes finding quality solutions in a short amount of time difficult. TTP has attracted significant interest recently since a favorable TTP schedule can generate large incomes in the budget of managing the league's sport. Many approaches have been applied to solve TTP, including exact or Approximate methods. In this work, we proposed an original enhanced Genetic Algorithm(E-GA) to solve TTP. We used a variable neighborhood search as a subroutine to improve the intensification mechanism in GA. Further, We propose a new aspiration technique to filter the search space and keep only the feasible configurations. The aspiration technique allows memorizing information on moves leading to feasible neighbor configurations, starting from a current configuration. The computational experiment shows that the proposed method could build interesting results comparable to other state-of-the-art approaches .

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 Wegiveareviewofthemaindefinitionandfundamentalissues 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

1.1 Introduction

In this chapter, we will learn about some basic concepts in optimization and introduce combinatorial optimization problems and method to solve them by exact or approximate method We also provided some examples of this problem and its complexity.

1.2 Optimization Problem

An optimization problem (OP) is a class of problems resolved by optimization methods that aim to find an optimal solution in a vast set of possible solutions.[3]

1.2.1 Definition

An optimization problem can be formalized in the following way :$P = (S, f, W)$ Where :
P: represents the optimization problem.
S: symbolizes the search space of the problem domain.
f : represents the objective function .
W: corresponds to the set of problem's constraints.
The search space S is defined through a set of variables $= X1, X2, ..., Xn$, generally called decision variables or design variables. The search space may be subject to a constraints W or restrictions. These restrictions are a set of definitions used to specify whether a solution is feasible or not. In other words, they regulate whether a solution can be accepted according to the rules of the problem domain. The f fitness function or objective function is used to assess and determine the quality of the solutions and guide the search process.[3]

A) The objective function :

A fitness function produces a real number (R), which is commonly referred to as fitness.

$$f : \mathcal{S} \rightarrow \mathbb{R}$$
$$\forall s \in \mathcal{S}, f(s) = fitness(\mathbb{R})$$

This number allows you to compare the results of all feasible solutions to a specific optimization problem.[3]

- **Local optimum:** A solution $s \in S$ is a local optimum if and only if there does not exist a solution
  $\forall s_0 \in v(s)$ $f(s) \leq f(s_0)$ In the case of a minimization problem
  $f(s) \geq f(s_0)$ In the case of a maximization problem
  With $V(s)$ the set of neighboring solutions of s.

- **Global Optimum:** A solution is a global optimum to an optimization problem if there are no other better solutions. The solution $s^* \in S$ is a global optimum if and only if:
  $\forall s \in S$ $f(s^*) \leq f(s)$ In the case of a minimization problem
  $f(s^*) \geq f(s)$ In the case of a maximization problem
  Therefore, the global optimum is the solution s^* which verifies the previous

property for all the neighborhood structures of the problem. Figure 1.1 schematizes the curve of an evaluation function by showing the global and local optimum .[4]

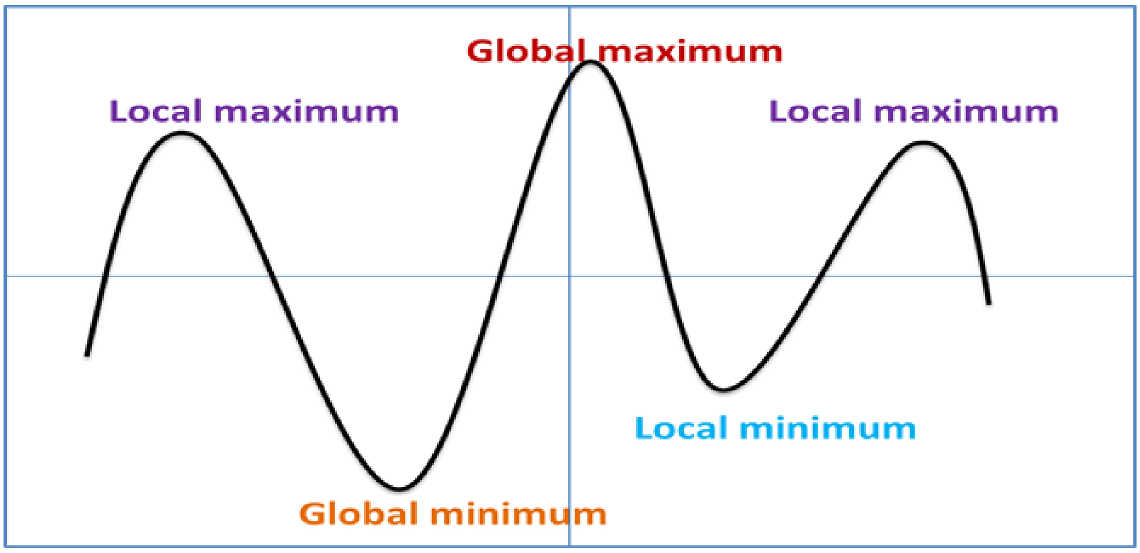


Figure 1.1: different minima and maxima

B) Combinatorial optimization problems (COP)

A combinatorial problem is a problem where it is a question of finding the best possible combination, That problem can be either a decision problem(a decision problem is a problem where the resolution is limited to answering <yes> or <no> to the question of whether there is a solution to the problem). [5]

1.3 The Resolution Methods (a combinatorial optimization problem):

The complexity of the problem can be solved in an exact method or an approximate method. The exact method give an obtain optimal solutions in a long time because it cannot solve NP-complete problems . As for approximate algorithms (or heuristic), they give high-quality solutions in a reasonable time, but there is no guarantee of finding a global optimal solution , Figure 1.2 shows that.

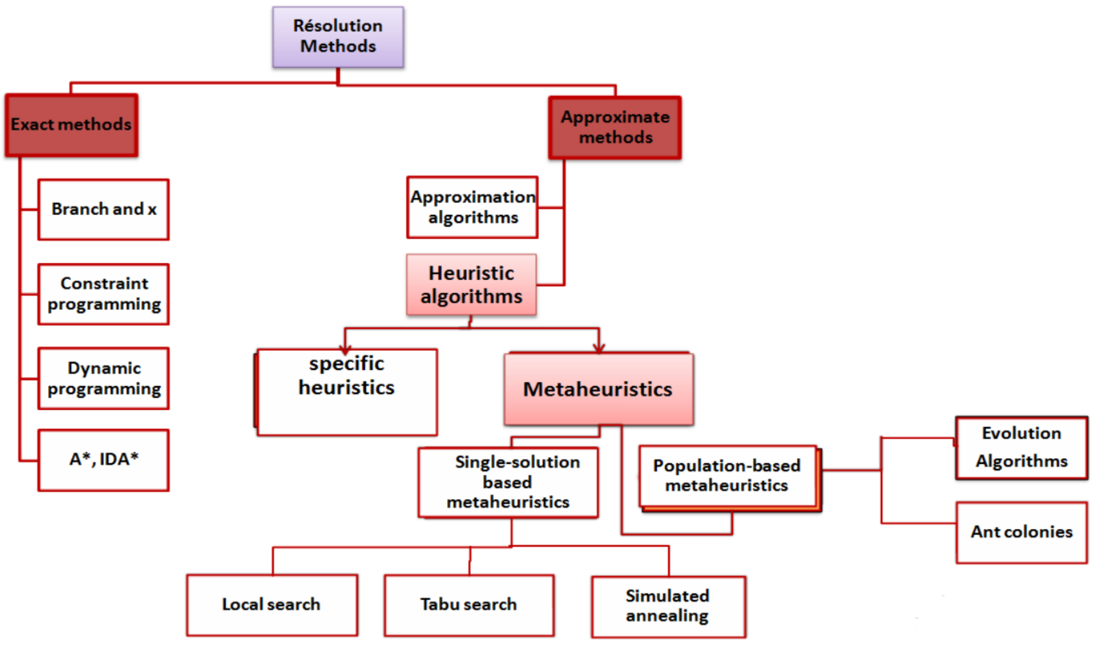


Figure 1.2: Classical Resolution Methods

1.3.1 Exact Methods

Exact methods seek to find the optimal solution with certainty by explicitly or implicitly examining the entire search space. They have the advantage of guaranteeing the optimal solution, however, the computational time necessary for reaching this solution becomes very excessive depending on the size of the problem and the number of objectives to optimize. What limits the use of this type of method for small size problems. These generic methods are: Branch & bound, Branch & cut and Branch & price, other methods are less general, such as: linear programming in integers, the algorithm of A *. Other methods are specific to a given problem like Johnson's algorithm for scheduling. In the class of exact methods one can find the following classical algorithms: dynamic programming, branch and X family of algorithms developed in the operations research community, constraint programming, and A * family of search algorithms developed in the artificial intelligence community.[2]

A) The branch and bound algorithm and A :

Are based on an implicit enumeration of all solutions of the considered optimization problem. The search space is explored by dynamically building a tree whose root node represents the problem being solved and its whole associated search space.The leaf nodes are the potential solutions and the internal nodes are sub-problems of the total solution space. The pruning of the search tree is based on a bounding function that prunes subtrees that do not contain any optimal solution. A more detailed description of dynamic programming and branch and bound algorithms.[2]

General algorithm

```
Start
     Place the start node of length 0 in a list.
  3: Repeat
  If
     the first branch contains the sought node then
  End successfully.
  Else
     - Delete the branch from the list and form new branches
    by extending the deleted branch by one step.
     - Calculate the cumulative costs of the branches and add
      them to the list so that the list is sorted in ascending order.
       Until (empty list or searched node found)
 End
```

Listing 1.1: branch and bound algorithm[4]

B) Dynamic programming

Is based on the recursive division of a problem into simpler sub-problems.
This procedure is based on the Bellman's principle that says that "the sub-policy of an optimal policy is itself optimal" .This stagewise optimization method is the result of a sequence of partial decisions. The procedure avoids a total enumeration of the search space by pruning partial decision sequences that cannot lead to the optimal solution.[2]

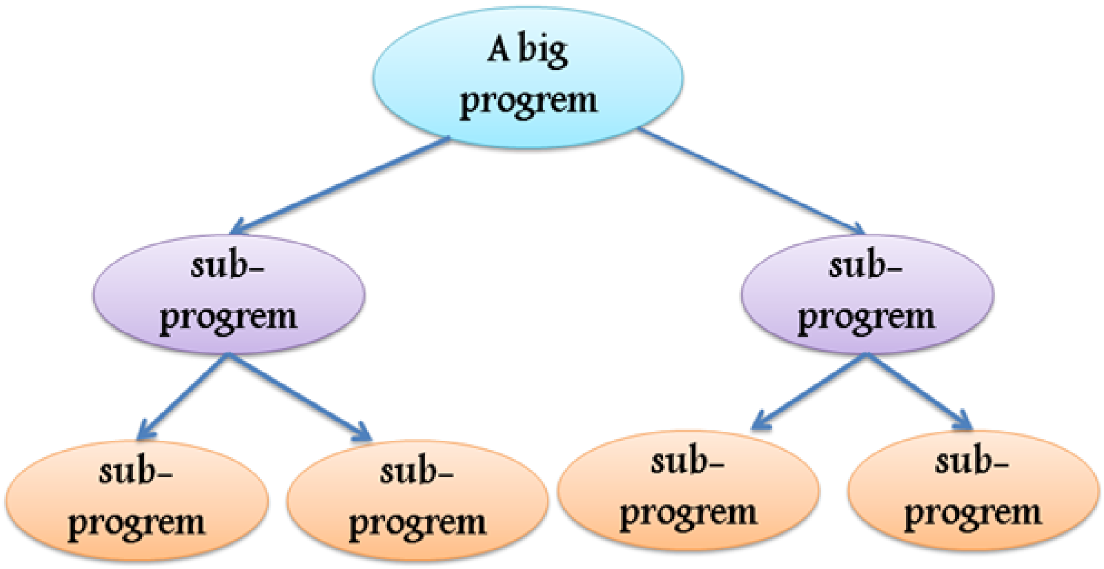


Figure 1.3: division a problem into simpler sub-problems

C) Linear programming

We can reduce the structure that characterizes linear programming problems (perhaps after several manipulations) into the following form:

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

In linear programming z, the expression being optimized, is called the objective function. The variables $x_1, x_2...x_n$ are called decision variables, and their values are subject to $m + 1$ constraints (every line ending with a bi, plus the nonnegativity constraint). A set of $x_1, x_2...x_n$ satisfying all the constraints is called a feasible point and the set of all such points is called the feasible region. The solution of the linear program must be a point $(x_1, x_2, ..., x_n)$ in the feasible region, or else not all the constraints would be satisfied.[6]

1.3.2 Approximate methods

In the class of approximate methods, two subclasses of algorithms may be distinguished:
Approximation algorithms and Heuristic algorithms.
Unlike heuristics which usually find reasonably "good" solutions in a reasonable time, Heuristics find "good" solutions on large-size problem instances.They may be classified into two families: specific heuristics and metaheuristics. The specific heuristics are tailored and designed to solve a specific problem and/or instance, Metaheuristics are general purpose algorithms that can be applied to solve almost any optimization problem.[2]

A) Approximation Algorithms

In approximation algorithms, there is a guarantee on the bound of the obtained solution from the global optimum . An ϱ-approximation algorithm generates an approximate solution a not less than a factor ϱ times the optimum solution s .

Definition : An algorithm has an approximation factor ϱ if its time complexity is polynomial and for any input instance it produces a solution a such that Where s is the global optimal solution ,and the factor and defines the relative performance guarantee. The ϱ factor can be a constant or a function of the size of the input instance. An ϱ-approximation algorithm generates an absolute performance guarantee21 ϱ, if the following property is proven:

$$(s - \varrho) \leq a \leq (s + \varrho)$$

$$a \leq \epsilon * sif\epsilon\, 1$$

$$\epsilon * s \leq aif\epsilon < 1$$

B) Heuristic algorithms

B.1) Metaheuristics Unlike exact methods, metaheuristics allow to tackle large-size problem instances by delivering satisfactory solutions in a reasonable time. There is no guarantee to find global optimal solutions or even bounded solutions. Metaheuristics have received more and more popularity in the past 20 years. Their use in many applications shows their efficiency and effectiveness to solve large and complex problems.

Optimization is everywhere; optimization problems are often complex; then metaheuristics are everywhere. Even in the research community, the number of sessions, workshops, and conferences dealing with metaheuristics is growing significantly!, Figure1.4 shows the genealogy of the numerous metaheuristics.[2]

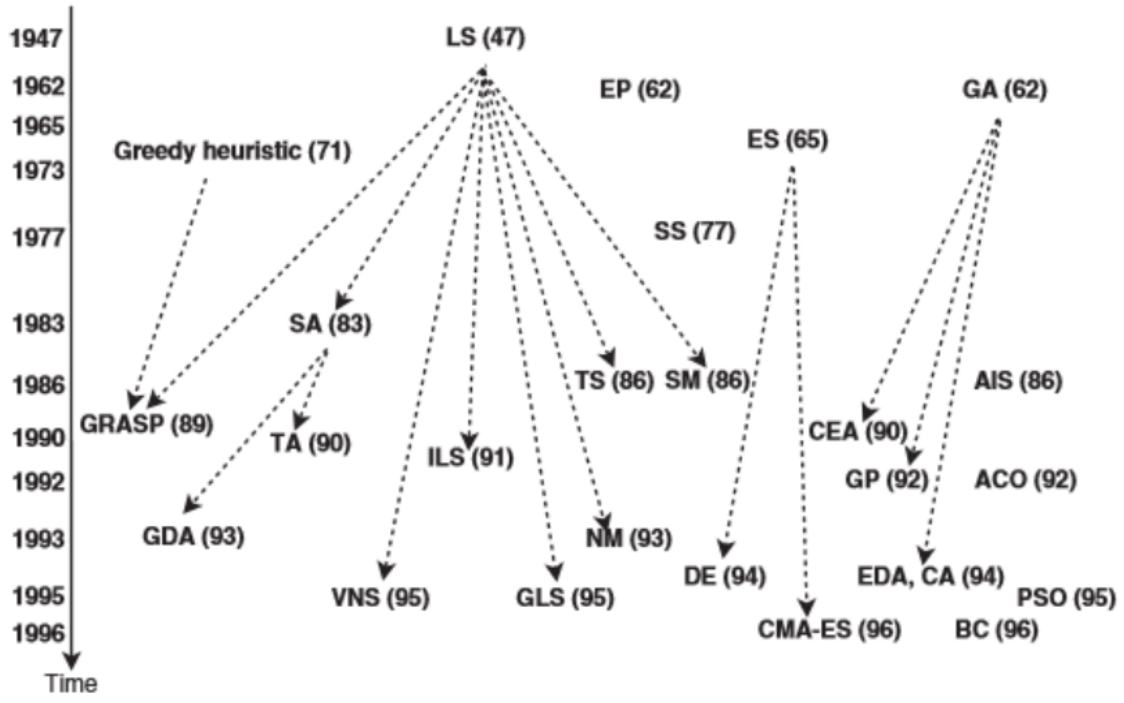


Figure 1.4: Genealogy of metaheuristics

B.2) Population-based metaheuristics Working on a set of points in the search space starting with an initial solution population and then improving it over time iterations, characterized by a tendency to explore and diversify . These methods are able to generate "good" solutions, evenly distributed in the search space. We will cite, among others, evolutionary algorithms, ant colonies and scatter search.[7]

B.3) Evolutionary algorithms Evolutionary algorithms are based on two fundamental principles of the evolutionary process of living beings: the survival of the best adapted individuals and genetic recombination. The principle is to simulate the evolution of a population of solutions, in consideration of these two rules, with a view to converging towards a set of solutions particularly well suited to the environment in which the problem being treated is identified. Holland uses the term adaptive plan (survival of the best individuals, elimination of the others), denoting an algorithm for the adaptation of a species of individuals in its environment. By its nature, although stochastic, it should make the population evolve towards beings which are better and better adapted to it. Evolutionary algorithms (EA) group together all the stochastic techniques based on the simulation of the adaptation process in natural environments, and bearing the names of genetic algorithms , evolutionary strategies , evolutionary programming and genetic programming.[2]

B.3.1) Genetic algorithm The genetic algorithm represents a famous evolutionary metaheuristic. It was proposed by Jhon Holland in 1975 . The genetic algorithm is inspired by biological mechanisms such as Mendel's laws and the theory of evolution proposed by Charles Darwin [Darwin, 1859]. Its process of finding solutions to a given problem mimics that of living beings in their evolution. It uses the same vocabulary as that of biology and classical genetics, so we speak of: gene, chromosome, individual, population and generation.[7]

Basic Principle:

```
formulate initial population
randomly initialize population
repeat
    evaluate objective function
    find fitness function
    apply genetic operators
         reproduction
         crossover
         mutation
    until stopping criteria
```

Listing 1.2: the Working Principle of a Simple Genetic Algorithm[1]

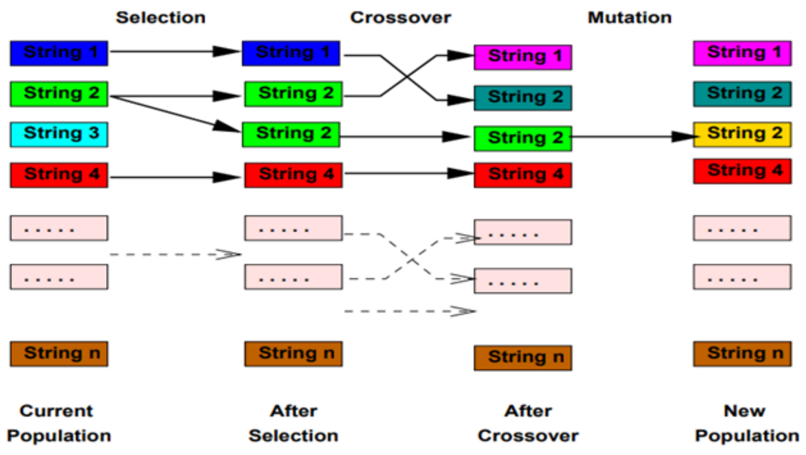


Figure 1.5: The basic GA operations [1]

These aspects are briefly described below. They are described in detail in the next subsection

one generation is broken down into a selection phase and recombination phase. Strings are assigned into adjacent slots during selection

Elements of genetic algorithms Genetic algorithms (GA) are inspired by classical genetics and use the same vocabulary. In general, a genetic algorithm consists of a population P of solutions called individuals, whose adaptation to their surroundings is measured using a fitness function g which returns a real value, called fitness which is the equivalent of the objective function in operations research. The general principle of a GA consists in simulating the evolution of a population of individuals using evolutionary operators until a stopping criterion is satisfied. Before explaining in detail the functioning of a genetic algorithm, we will present some vocabulary words relating to genetics. These words are often used to describe a genetic algorithm.

1. **Gene**

   :is a set of symbols representing the value of a variable. In most cases, a gene is represented by a single symbol (a bit, an integer, a real or a character).

2. **Chromosome**

   : is a set of genes, presented in a given order in a way that takes into consideration the constraints of the problem to be treated. For example, in the traveling salesman problem, the size of the chromosome equals the number of cities to travel. Its content represents the running order of different cities. In addition, care must be taken that a city (represented by a number or a character for example) must not appear in the chromosome more than once.

3. **Individual**

   :is composed of one or more chromosomes. It represents a possible solution to the problem addressed.

4. **Population**

   :is represented by a set of individuals (i.e. the set of solutions to the problem).

5. **Generation**

   is a succession of iterations composed of a set of operations allowing the transition from one population to another.[8]

```
   Start
  Generate a random population of n chromosomes.
  Evaluate the fitness of the chromosomes with the function f (x)
  Repeat
    Calculate the fitness function f (x), for any chromosome x
    Apply the selection operation
    Apply the crossover operation with a probability PC
    Apply the mutation operation with probability PM
    Add the new chromosomes to the new population
    Calculate the fitness function  f(x), for any chromosome x
    Apply the replacement operation
  Until satisfaction of termination conditions
End
```

Listing 1.3: Basic Genetic Algorithm[3]

B.3.2) Biogeography-based Optimization (BBO) Biogeography-based Optimization (BBO) is one of the new metaheuristics method inspired by the natural phenomenon, biogeography. Biogeography is science learning about the distribution of some specific species depends on the geographic condition . The BBO method was introduced by Simon to solve some continuous functions. Moreover, this method was made based on some principals that are how the species migrate from one island to another, how new species arise, and how the species become extinct. These are some concepts of BBO

1. **Biogeography**

   Based on the biogeography principals, island with the higher suitability have a large number of species, while the island with a low suitability have smaller number of species. Therefore, the solutions of the problems are analogous to those islands. The suitable island would have high Habitat Suitability Index (HSI). In other populationbased optimization algorithms (Genetic Algorithm, for example), HSI is usually called “fitness”. The variables that characterize the HSI are called Suitability Index Variable (SIV). SIV can be considered the independent variable of habitat, and the HSI can be considered the dependent variable .

2. **Migration**

   The habitats or islands that have high HSI or many species in it would have high emigration level and also low immigration level. Hence, the habitat with higher HSI would tend to be static. The species would tend to move to the nearest habitats since they have high emigration level, vice versa. Nevertheless, the species immigrating to another island would not completely disappear from their origins. Those species would appear in both islands at the same time. In general, the migration process would make the bad solution accept some features from the better solutions . The higher the emigration level of an island means the lower its immigration level, vice versa. Nevertheless, the emigration level of the island depend on the number of species lived in it. An island with high emigration level would have more species than the others that have lowest emigration level. Figure 1.6 shows the connection between emigration level, immigration level, and the number of species, also the comparison between two different solutions.

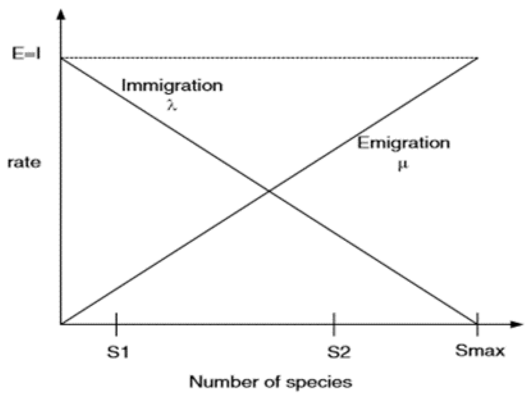


Figure 1.6: Illustration of two candidate solutions to some problem

3. **Mutation and Elitism**

   Besides the migration process, the mutation and elitism are also happen in the BBO method. Mutation is a cataclysmic event that happens on the habitat. The probability of mutation on some habitats are called mutation rate. Mutation rate of some habitats depend on the number of species lived in the habitats. Habitat with high HSI values would more likely have lower mutation rate compared to those that have low HSI values. Therefore, the good solution is rarely selected to be mutated so that it could last until the next generation. This mutation would bring new habitat to replace the old one that have low HSI values. If there is no mutation, the solutions with low HSI would be more dominant so that they could be trapped on the local optima. The mutation rate of every habitat could be formulated as:

$$m_k = m_{max}\left(\frac{1-P_k}{P_{max}}\right)$$

   where m_k is the mutation rate, m_{max} is the maximum mutation rate that is the user-defined parameter, P_k is the probability of the number of the species in the habitat, and P_{max} is the maximum probability might be happened. The new habitat from the mutation would replace the old habitat. And with the elitism, the best solutions found before would remain. The mutation could happen on all of the solutions except for the best solutions with the highest probability P_k. The mechanism of the mutation used in BBO could be varied just like the mechanism of the mutation that has been used on the Genetic Algorithm.

B.4) Ant colonies The Theory of Ant Algorithm The Ant Colony Optimization techniques has emerged recently as a relatively novel meta-heuristic for hard combinational optimization problems. It is designed to simulate the ability of ant colonies to determine shortest paths to food. Although individual ants posses few capabilities, their operation as a colony is capable of complex behavior. Real ants can indirectly communicate by pheromone information without using visual cues and are capable of finding the shortest path between food sources and their nests. The ant deposits pheromone on the trail while walking, and the other ants follow the pheromone trails with some probability which are proportioned to the density of the pheromone. The more ants walk on a trail, the more pheromone is deposited on it and more and more ants follow the trail. Through this mechanism, ants will eventually find the shortest path. Artificial ants imitate the behavior of real ants how they forage the food, but can solve much more complicated problems than real ants can. A search algorithm with such concept is called Ant Colony Optimization. Figure 1.7 shows how the ants find the shortest path [9].

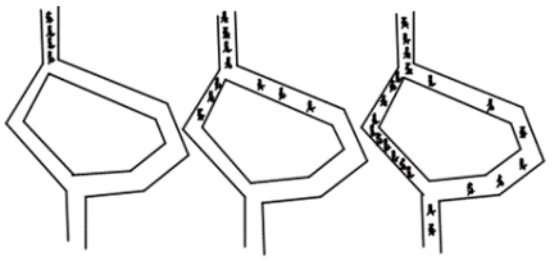


Figure 1.7: Sketch map of the ant theory

B.5 Local search In local search, we only keep track of present state. That is, current state is the single state we bother about, and we ignore keeping track of paths. That is why local search is very memory efficient. It is capable of finding reasonable solution in very large or even infinte state space. While we are in a current state, we can move only to a neighboring state - that is, we search in local neighborhood. So, given states and neighborhoods of the states, and given an objective function that evaluates a state, local search algorithms find a state that has optimum (maximum or minimum) value of objective function. In local search, every state is a solution - bad or good. A solution is good where higher number of constraints are statisifed, and bad where lower number of constraints are satisfied.[7]

```
    Step 1 (initialization)
            a) choose an initial solution s ∈ S.
            b) s*← s (i.e. s* memorizes the best solution found)
    Step 2 (choice and termination)
            a) choose s’ ∈ N(s)
            b) s ← s’ (i.e. replace s by s’)
            c) end if the stopping condition is verified
```

```
   Step 3  (update)
          a) s* ← s if f(s) < f(s*)
          b) go to Step 2
```

Listing 1.4: Simple local search algorithm [3]

B.6) Hill Climbing Algorithm As the name Hill Climbing suggests, it's concept is associated with climbing to the hill top. Following figure 1.8 shows a sample hill. When monuntaineers go for expeditions such as climbing the Mount Everest, how does one realize if she has rached at the summit or peak of the Mount Everest. They keep a GPS tracker or altitude meter that guides them in this regard. However, there can be many other peaks with little bit smaller height in the surroundings of summit which can mislead monuntaineers to believe that they have reached at the peak in the absense of such GPS tracker.

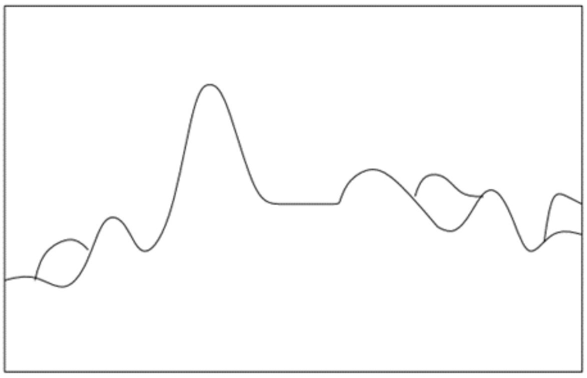


Figure 1.8: A Hill

Such things can also happen when we search for solution in the search space, where there are many local optimums and one global maximum. While doing the local search using hill climbing we may reach a local maximum, and we believe we have reached at the global maximum. In hill climbin algorithm, since an objective function is associated with every state, so given a current state, all it's neighborhoods are evaluated with the objective function. If a neighborhood state is found with value higher than the current state and it's the maximum value amongst the neighborhood, then this neighbor becomes the new current state. This is repeated. If none of the neighbor has higher value than the current state then terminating condition has been reached and the hill climbing algorithm returns the local maximum and then terminates. So, we can say that hill climbing algo. is the greedy local search. Current state may have many alternative for next state.[7]
Choose the next state which seems the best. Following algorithm 1.5 gives the maximum version of Hill Climbing algorithm

```
   Start
     Generate and evaluate an initial solution s
     While The stopping condition is not verified  do
       Modify s to obtain s' and evaluate s'
       If(s' is better than s)    then
         Replace s by s'
       End if
     End while
     return  s
   End
```

Listing 1.5: General scheme of Hill-Climbing

B.7) Tabou searsh In hill climbing if the search gets suck in a local optimum, there is no way to come out of that. One solution is to take steps back from local optimum and go down to reach at the bottom. Once the bottom is reached then the search is resumed afresh with the hope that better solution will be visited. This is continued. However, a limit on possible number of sideway moves to be placed to prevent infinte looping. This is the concept of tabu search. In tabu search we keep track of last node and this node is not repeated.[7]

```
function TabuSearch(problem)
Input : problem
Output: returns a state that is possibly global maximum
local variables: current_node,best_node
  current_node ← Random(InitialState(problem))
  best_node ← current_node
  Tabu_List ← current_node
  while (!Empty(Tabu_List))
    current_node ← Non_Tabu_SuccessorHighest(current)
    Tabu_List ← current_node
    if(Value(current_node)≤ Value(best_node))
      best_node ← current_node
  end of while loop
  return State(current_node)
end TabuSearch
```

Listing 1.6: Tabu Search Algorithm [3]

1.4 Computational Complexity

Sometimes, there are more than one way to solve a problem. We need to learn how to compare the performance of different algorithms and choose the best one to solve a particular problem. While analyzing an algorithm, we mostly consider the computational complexity which is divided into: *Time complexity* and *Space complexity.*The computational complexity of an algorithm is the amount of time and memory required to run it. [2]

1.4.1 Problem Classes

An algorithm to be polynomial "P" (polynomial running time) if for k>0 its running time on input of size n can be described at the worst with the formula O(nk), this includes to be : linear or quadratic, cubic and more. Algorithms with exponential running time are not polynomial. This classification includes the fundamental well-known classes:

- **P (polynomial time)** class of problems that are solvable in polynomial time with a deterministic Turing machine.
- **NP (non-deterministic polynomial time)** class of problems that can only be verified in polynomial time with a deterministic Turing machine.
- **NP-complete**problem that can be solved in P (but in a large polynomial time) and can be verified in NP and all problems of NP class can be reduced to it.

- **NP-hard**problem that have not a solution in P but can be verified in NP and all pbs of NP class can be reduced to it.

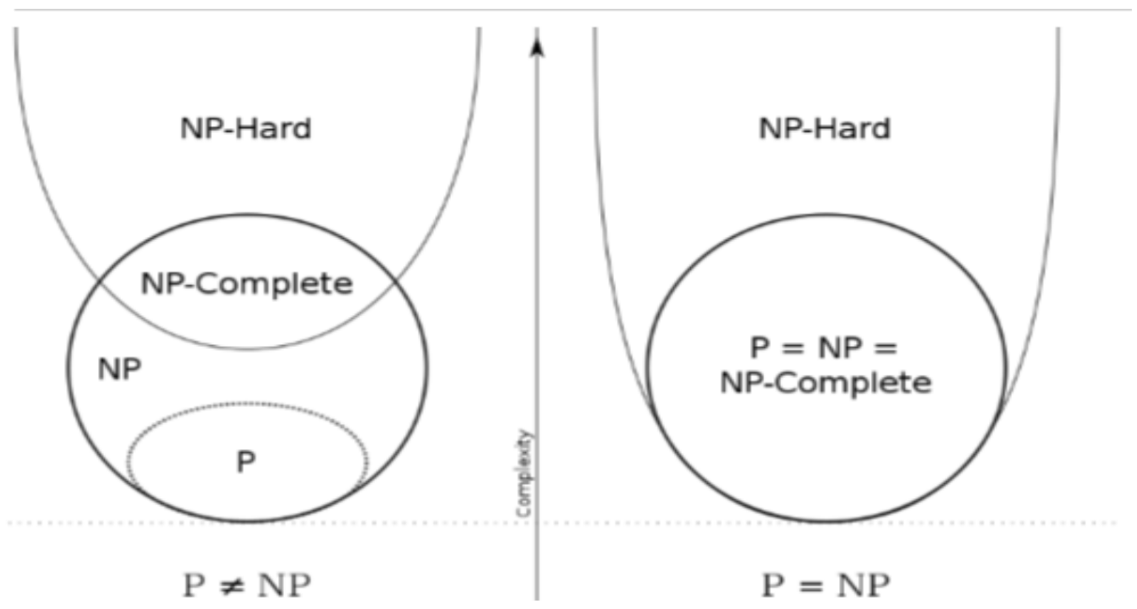


Figure 1.9: There are two possibilities depending whether P=NP Or P≠ NP [2]

1.4.2 Time Complexity

Time complexity is the number of operations an algorithm performs to accomplish its task, we assume each operation takes the same time. The algorithm that performs the task in the smallest number of operations is considered to be the most efficient regardless the kind of machine it runs on. In time complexity, it is necessary to decide what is an operation? the choice depends on the problem. We have to choose some small operations that the algorithm often does, and that you want to use as basic operations to measure complexity.When we compute the time complexity T(n) of an algorithm we rarely get an exact result, but just an estimate, that's why, in computer science we use ASYMPTOTIC notations, it means that is an approximation of the number of elementary operations. There are mainly three asymptotic notations:
Big Ω notation for best case;
Big Θ notation for average case;
Big O notation for worst case.[10]

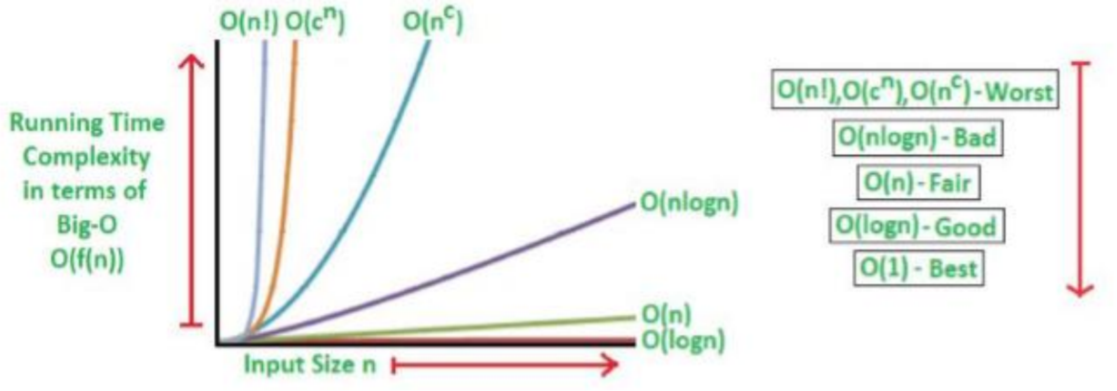


Figure 1.10: Comparison of complexity between functions

1.5 Some optimization problems

1.5.1 Traveling salesman problem (TSP)

Perhaps the most popular combinatorial optimization problem is the traveling salesman problem (TSP)

- Given a number of cities and the costs (distances) of traveling from any city to any other city...
- What is the least-cost round-trip route that visits each city exactly once and then returns to the starting city?[2]

1.5.2 Knapsack problem

We have n objects each having a weight pj and a value or utility vj with j = 1 $\cdots$ n it is a question of making a selection of these objects to put in the bag (choose a subset of objects) ; whose total weight is less than or equal to a certain value B so as to maximize the total utility of the bag. Defining for each object j, a decision variable x_j such that

if object j is chosen else $x_j = \begin{cases} 1 \\ 0 \end{cases}$

1.5.3 8-queen problems

In 8-queen problem, we need to place 8 queens in an 8×8 board such thatnot-woqueensattackeachother: thatis,thereshouldnotbetwoqueens in the same row or same column or same diagonal. The figure 1.11 shows a solution to the 8-queen problem.[7]

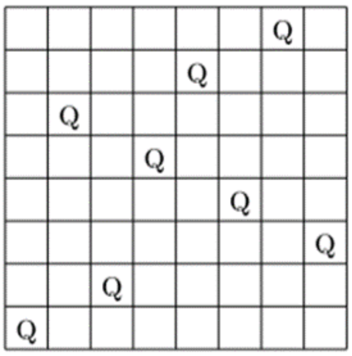


Figure 1.11: A Solution to 8-Queen Problem

1.6 Conclusion

In this chapter, we talked about the optimization problem , the combinatorial optimization and The Resolution Methods to these problems ,Exact Methods where we get a Exact solution in a long time perhaps, or the Approximate methods To find the search space in a less time , the time complexity and we presented some examples of optimization problems, Among these problems is the ttp problem, which we will talk about in the next chapter.

CHAPTER 2

The Traveling Tournament Problem

2.1 Introduction

In recent years, sports and computer technologies have witnessed a great development in terms of planning and organizing sports competitions, Many sports leagues (soccer, hockey, basketball) must deal with scheduling problems for tournaments he problem of scheduling , Traveling Tournament Problem (TTP) originated in Major League Baseball in the United States proposed by Kelly Stone Gorgo and Michael, it is one of the most complex problems classified as a Nb Heard problems, as broadcast and television networks are willing to pay a lot of dollars for sports leaks, in this Chapter you will learn about the Sports Scheduling Problem, the TTP its history, its variant.

2.2 The Sports Scheduling Problem

Sports Scheduling in general deals with the design of tournaments. A single round-robin tournament on n teams, where n is an even number, consists of (n 1) rounds (also called slots). In each round n/2 games, which are themselves ordered pairs of teams, take place. Every team has to participate in one game per round and must meet every other team exactly once. It is standard to assume n to be even since in sports leagues with n being odd, usually, a dummy team is introduced, and whoever plays it has a day off, which is called a bye. For scheduling single round-robin tournaments a rather general and useful scheme called the canonical schedule has been known in sports scheduling literature for at least 30 years . It is based on the polygon/circle method, which was first suggested by Kirkman in 1847 . One can think of Kirkman's method as a long table at which n players sit such that n/2 players on one side face the other players seated on the other side of the table. Every player plays a match against the person seated directly across the table. The next round of the schedule is obtained when everyone moves one chair to the right with the crucial exception that there exists one person at the end of the table who never moves and always maintains the seat from his or her first round. Note that this method only specifies who plays whom when and not where. The canonical schedule introduced by deWerra defines for each of the encounters specified by the method described above, at whose side they take place such that the number of successive home or away games is minimized . A double round-robin tournament on n teams consists of 2(n1) rounds and every team must meet every other team twice: once at its home venue (home game) and once at the other team's venue (away game). A popular policy in practice is to obtain a double round-robin tournament from a single round-robin tournament by mirroring, that is repeating the matches of round k for k = 1,...,n1 in round k + n1 with changed home field advantage. Consecutive home games are called a home stand and consecutive away games form a road trip. The length of a home stand or road trip is the number of opponents played (and not the distance traveled).
Tournaments may be represented by graphs, which offer a good model for scheduling formulations and algorithms, see e.g. de Werra (1980; 1981; 1988). The complete graph Kn may be used to represent a single round robin tournament or any of the phases of a compact double round robin tournament. Each of its nodes represents a team. Each game is represented by an edge, whose extremities are associated with the two opponent teams. Figure 2.1 displays an example illustrating the graph

representation of a single round robin tournament with n = 4 teams.[11] The problem

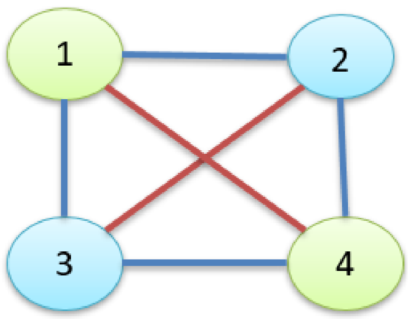


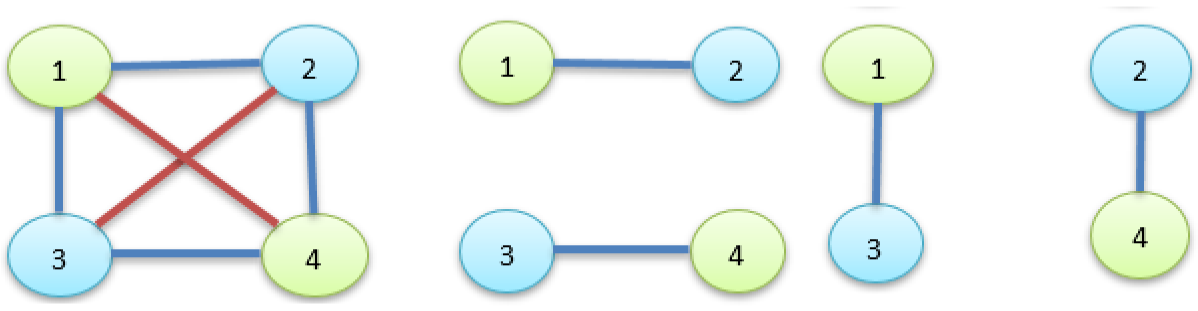
Figure 2.1: Example of a single round robin tournament with n = 4 teams represented by a complete graph

of scheduling a round robin tournament is often divided into two subproblems. The construction of the timetable consists in determining the round in which each game will be played. The home-away pattern (HAP) set determines in which condition (home or away) each team plays in each round. The HAP set for the previous example can be represented by a matrix as in Table 1, in which the cell corresponding to row k and column j indicates the playing condition of team j in round k. Together, the timetable and the home-away pattern set determine the tournament schedule [11]

2.3 A History of the Traveling Tournament Problem

The Traveling Tournament Problem In early 1999, Trick decided to write up some of his work on the MLB schedule. Work up to then has always been on solving the full MLB problem. But certain aspects of that work were confidential (team preferences most particularly), so Trick could not simply describe the whole problem. Instead, he decided to start with the simplest set of constraints: simply find a minimum travel schedule with limits on the length of home stands and road trips and no other constraints. Once this was solved, additional constraints could be added until the reason for the difficulty in solving could be identified. Presumably, those could eventually be overcome and the MLB problem could be solved. To Trick's enormous surprise, his methods, based on the stronger home-stand/road trip formulation were not sufficient to solve even the base model. Taking 30 teams and finding a minimum travel distance schedule did not seem doable. Even taking 16 or 14 teams (at the time MLB had two leagues with only weak interactions between them) seemed difficult. Trick looked at smaller and smaller instances of this simplified problem before realizing the only size instance he could solve had just 4 teams. It may have been small, but on May 1, 1999, the first instance of what would be the Traveling Tournament Problem was solved.

Trick sent the problem definition and data to Easton who had more computing power and more advanced methods, including constraint programming approaches. She was quickly able to confirm the optimality of Trick's four-team solution and provide the optimal solution to the six team instances. But the 8 team instances could not be solved to optimality with the methods of the time. It was not until 2002 that Easton finally found what was to be the optimal solution to the 8 team



(a) First round: 1-factor F (b) Second round: 1-factor F2 (c) Third round: 1-factor F3

Figure 2.2: Timetable of a single round robin tournament with n = 4 reams represented one of its 1-factorizations: (a) 1-factor F1 associated with the first round, (b) 1-factor F2 associated with the second round, and (c) 1-factor F3 associated with the third round.

instance, and not until 2008 that it was confirmed optimal by the lower bounding method of Irnich and Schrempp. At this point, the problem did not have a name. Trick suggested something like "The minimum travel for a sports league problem". He was wisely overruled by his coauthors and the Traveling Tournament Problem was born.[12]

2.4 The Traveling Tournament Problem

Given n teams, n even, a double round robin tournament (DRRT) is a set of games in which each team plays each other team once at home and once away. A schedule for a DRRT is a mapping of games to slots, or time periods, such that each team plays exactly once in each slot. A DRRT schedule covers exactly 2(n1) slots. The distances between the team venues are given by an n by n matrix D. For the distance calculations, it is important to note that each team starts and finishes the tournament at its home venue. Aroadtrip,ortrip,isdefinedasaseriesofconsecutiveawaygames.Similarly, a home stand is defined as a series of consecutive home games. The length of a road trip or home stand is the number of games in the series (not the travel distance). The TTP is defined as follows: [12]

- **Input** : n , the number of teams ; D an n by n integer distance matrix ; L , U integer parameters
- **Output** : A double round robin tournament on the n teams such that
  - The length of every home stand and road trip is between L and U inclusive
  - - The total distance traveled by the teams is minimized

  In addition to the basic constraints , there may be additional requirements . on the solution . These include :

  - **Mirrored** Mirrored :The double round robin tournament must have a round robin tournament in the first $n-1$ slots and then have the same tournament with venues reversed in the second $n-1$ slots .

- **No Repeaters** There are no teams i , j such that i plays at j and then j plays at i in the next slot

2.5 TTP Complexity:

Despite its simplicity, TTP is a difficult problem to solve, and it is known to be a strongly NP-hard problem . Until now there is no research could solve instance with more than ten teams optimally . The first NP-completeness proof has been given by Bhattacharyya for a variant of the original TTP, where the constraint on consecutive home-games and away-games is left out, which allows a reduction from TSP. The second attempt on TTPs complexity proof is made by Thielen and Westphal using a reduction from 3- satisfiability (3-SAT). They showed that the TTP is strongly NP-complete when the upper-bound of consecutive home games and away-games is fixed to 3. It still doesnt prove the original TTP, where u and l can be arbitrary integer numbers, but nonetheless, it is a big contribution in the analysis of the TTPs complexity. Furthermore, the authors also pointed out that with further refinement, their proof can be probably generalized for the original TTP.

2.6 the TTP and its variants

The Unconstrained Traveling Tournament Problem (UTTP) [77] is a variant of TTP in which constraints 1 and 2 are eliminated. The objective in UTTP is the same of TTP, find a schedule with minimum cost travel where the constraints 1 and 2 are not necessarily needed: UTTP is TTP without any constraint on the consecutive home and away matche. UTTP is a suitable model for some practical scheduling problem. Furthermore, eliminating the breaks constraints (1 and 2) reduces the budget constraint.
Thus a solution of the UTTP is an appropriate model for overseeing sports league with a small budget. mTTP is the mirrored version of TTP. mTTP requires that the games played in round R are exactly the same as those played in round R+n1, for R=1,2,....n1 with reversed venues (Table5.1). We called this additional constraint mirrored constraint. mTTP is then the problem of finding a schedule for a double round robin tournament with minimum cost satisfying the same constraints plus an additional constraint" the mirrored constraint.

The Bipartite Traveling Tournament Problem (BTTP): let there be 2n teams, with n teams in each league. Let X and Y be the two leagues, with X = x1, x2, x3..., x2n and Y = y1, y2, y3..., yn. Let D be the 2n $\times$ 2n distance matrix, where entry Dp,q is the distance between the home stadiums of teams p and q. By definition, Dp,q = Dq,p for all p, q X Y , and all diagonal entries Dp,p are zero. Similar to the original TTP, we require the following conditions: that each team play one game per day; that no team has a home stand or road trip lasting more than three games; that no team play against the same opponent in two consecutive games; and that for all i n, j n, teams xi and yj play twice, once in each others home venue. To illustrate, Table 2.1 provides two examples of a feasible tournament satisfying all of the above conditions for the case n = 3. In this table, as in all other schedules that will be subsequently presented, home games are marked in bold.

Teams	1	2	3	4	5	6
x_1	y_1	y_2	y_3	Y_1	Y_2	Y_3
x_2	y_2	y_3	y_1	Y_2	Y_3	Y_1
x_3	y_3	y_2	y_1	Y_3	Y_2	Y_1
y_1	X_1	X_3	X_2	x_1	x_3	x_2
y_2	X_2	X_1	X_3	x_2	x_1	x_3
y_3	X_3	X_2	X_1	x_3	x_2	x_1

Table 2.1: A feasible inter league tournament for n=3

The Traveling Tournament Problem with Predefined Venues(TTPPV) is a single round robin variant of the TTP, in which the venue of each game to be played is known beforehand (i.e., the venue where each game takes place is known beforehand) while the total distance traveled by the teams is minimized. In three integer programming formulations for the Traveling Tournament Problem with Predefined Venues are proposed. These formulations are compared with respect to the bounds provided by their linear relaxations. Let Q be a set of games, represented by ordered pairs of teams determined by the predefined HAA. The game between teams i and j is represented either by the ordered pair (i, j) or by the ordered pair (j, i). In the first case, the game between i and j takes place at the venue of team i; otherwise, at that of team j. For every two teams i and j, either (i, j) Q or (j, i) Q. TTPPV consists in finding a compact single round robin schedule compatible with Q, such that the total distance traveled by the teams is minimized and no team plays more than three consecutive home games or three consecutive away games.

2.7 Conclusion

In this chapter, we have provided a general explanation of The Sports Scheduling Problem and The Traveling Tournament , its variants, its inputs and outputs. Our main goal in this work is to find a solution that is closer to the optimal. In the next chapter, we will discuss how to apply the proposed approach to reach this solution.

CHAPTER 3

The proposed Approach (GA + LOCAL SEARCH)

3.1 Introduction :

In the previous chapter, we got acquainted with the The Sports Scheduling Problem , The Traveling Tournament Problem, its constraint and its variants. In this chapter, we applied the proposed heuristics approach to solve the problem of the The Traveling Tournament (TTP). First, we applied the local search method , then the genetic algorithm . We also applied the Neighborhood Structures and applied different swaps to the two approaches.

3.2 the proposed approach:

In order to solve the problem of scheduling The Traveling Tournament (TTP)., we applied the proposed approach, first we started by defining the inputs and outputs, then the algorithms apply the local search and the genetic algorithm as shown in these steps

- **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 3.1: 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
Team 4	-1	–3	–3	1	3	3

Table 3.2: Schedule for n = 4 teams

3.3 Genetic algorithm for TTP

A genetic algorithm is a type of searching algorithm. It searches a solution space for an optimal solution to a problem. The key characteristic of the genetic algorithm is how the searching is done. The algorithm creates a "population" of possible solutions to the problem and lets them "evolve" over multiple generations to find better and better solutions.

3.3.1 Population

is the collection of candidate solutions that we are considering during the course of the algorithm. Over the generations of the algorithm, new members are "born" into the population, while others "die" out of the population. A single solution in the population is referred to as an **individual. The fitness** of an individual is a measure of how "good" the solution represented by the individual is. The better the solution, the higher the fitness – obviously, this is dependent on the problem to be solved.

3.3.1.1 Application of proposed approach Local search

The Initial Configuration The search method starts with an initial configuration verifying the DRRT constraint. We create this configuration based on graph-theory modelling as follows: We have n/2 games per round and 2 · (n1) rounds. We number the vertices of the graph from 1 to n, where n is the number of teams. We put the top n in the center and the other vertices in a circle around the top n.[13]
Local Search Method for Feasible Schedules

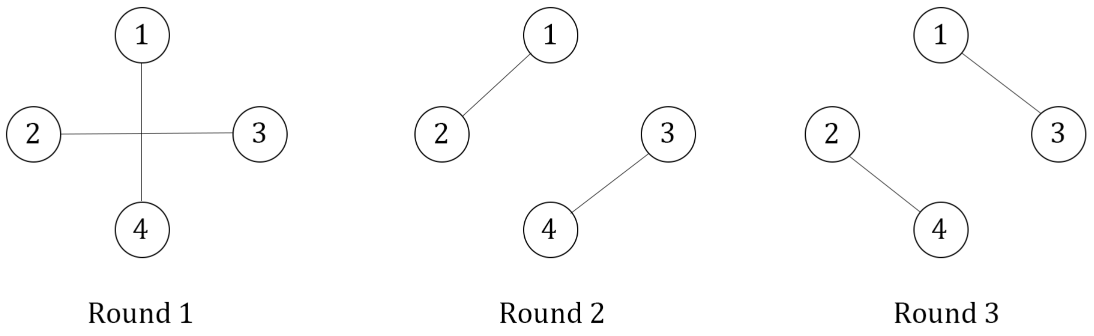


Figure 3.1: Local Search Method for Feasible Schedules

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)

Table 3.3: Local Search Method for Feasible Schedules

the Neighborhood Structures We use three neighborhood structures which are detailed in the following:

- **N1: Swap Home.** This move swaps the home/away roles of teams. For instance, when we take two teams ti and tj, the move Swap Home(S,ti,tj) swaps the home/away roles of a game involving the teams ti and tj. If team ti plays home against team tj at Round k and away against team tj at Round l then the move Swap Home (S,ti,tj) gives the same schedule as S, except that team ti plays away against team tj at Round k, and home against tj at Round l.

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

- **N2: Swap Round.** This move consists of swapping all games of a given pair of rounds. For example the move Swap Round (S,Roundk,Roundl) swaps two given rounds (Roundk and Roundl).

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,Round1,Round3):

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

- **N3 Swap Team.** This move corresponds to swapping all opponents of a given pair of teams over all rounds, For example the move Swap Team (S,ti,tj) corresponds to swapping all opponents of teams ti and tj over all rounds.
  After the application of the move Swap Team: N3 (S,(t2,t4))

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)

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

Aspiration Technique to Select the Best Neighbor We propose a new technique which we called aspiration technique to filter the search space and keep only the feasible configurations. The aspiration technique permits to memorize information on moves leading to feasible neighbor configurations, starting from a current configuration. First, we explore the search space to locate feasible configurations with zero-cost according to the cost function. Then among them, we take the best one having the minimum traveled distance.

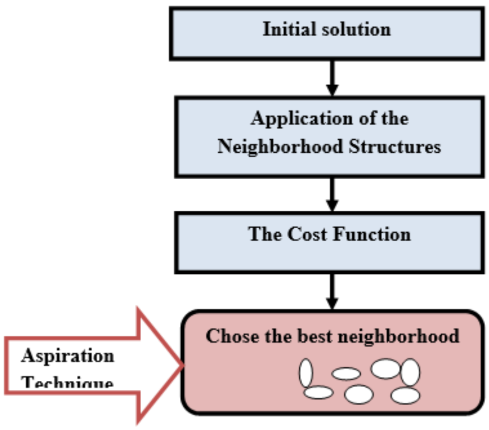

Initial solution
Application of the Neighborhood Structures
The Cost Function
Chose the best neighborhood
Aspiration Technique

Figure 3.2: The proposed approch local search

3.3.2 The selection

The selection process is analogous to the survival of the fittest in the natural world. Individuals are selected for "breeding" (or cross-over) based upon their fitness values –the fitter the individual, the more likely that individual will be able to reproduce. The cross-over occurs by mingling the two solutions together to produce two new individuals. During each generation, there is a small chance for each individual to mutate, which will change the individual in some small way.

3.3.2.1 The Cost Function:

The cost function consists of penalize configurations not satisfying the DRRT constraint. First, it is important to he following useful notations in Table 4 to represent the constraints.

Ti	is the team number i where i [1,n].
(ti,tj)	is the game ti, vs. tj in the home of ti.
Roundl	is the round number l where 1 l |Round|.
Ri,j	means that the match (ti,tj) is scheduled in a round R_i, j where $1 \leq R_i, j$|Round|, $\forall i, j \in |T|, i6 = j$. For example in Table 3, the match (t5,t1) is scheduled in Round8 at home of t5, R5,1 = 8, while the match (t1,t5) is scheduled in Round3, R1,5 = 3.
S	is a DRRT schedule

Table 3.4: Some useful notations and definitions

The Objective Function

$$Min \sum_{i,j}^{n} Distance + \sum_{i,j}^{n} no\ \ satisfate\ \ DDRT\ \ Constraint\ \ * 100$$

3.3.3 The Crossover

Cross over It is the process in which two chromosomes combine their genetic material to produce a new offspring which possesses both their characteristics. Two strings are picked from the mating pool at random to crossover .T he method chosen depends on the Encoding Method. The most widely used cross over methods are

1. **Single Point Crossover :**Single point crossover is the most commonly used crossover . A crossover site is selected randomly along the length of the mated strings and bits next to the cross-sites are exchanged. If appropriate site is chosen, better children can be obtained by combining good parents else it severely hampers string quality. In one point crossover the head and tail of one chromosome separates and if both head and tail contains the good genetic material then none of the offspring will obtained the both good features directly.

2. **N-Point Crossover** The N-point crossover was first implemented by De Jong in 1975 . It consists of more than one cross over sites but principle used is same as that of single point crossover . In 2-point crossover value of crossover sites is

2. Adding of the more crossover sites causes more disruptions of building blocks that sometimes reduce the performance of genetic algorithm. But it allows the head and tail portion of a chromosome to be passed together in the offspring.

3. **Uniform Crossover :** Uniform crossover do not fragments the chromosomes for recombination. Each gene in offspring is created by copying it from the parent chosen according to the corresponding bit in the binary crossover mask of same length as the length of the parent chromosomes . If the bit in crossover mask is 1, then he corresponding gene is copied from the first parent and if the bit in crossover mask is 1, then the corresponding gene is copied from the second parent. A new crossover mask is generated randomly for each pair of parent chromosomes. The number of crossover point is not fixed initially. So, the offspring contains a mixture of genes from both the parents.

4. **Partially mapped Crossover:**Partially Matched or Mapped Crossover (PMX) is the most commonly used crossover operator in permutation encoded chromosomes. It was proposed by Goldberg and Lingle for Travelling Salesman Problem. In Partially Matched Crossover, two chromosomes are aligned and two crossover sites are chosen randomly. The portion of chromosomes between the two crossover points gives a matching selection that undergoes the crossover process through position-by-position exchange operations . PMX tends to respect the absolute positions.

5. **Cycle Crossover (CX):**Cycle crossover is used for chromosomes with permutation encoding. During recombination in cyclic crossover there is a constraint that each gene either comes from the one parent or the other . The basic principle behind cycle crossover is that each allele comes from one parent together with its position. To make a cycle of alleles from parent1, start with the first allele of parent1. Then look at the allele at the same position in parent2 and go to the position with the same allele in Parent1.Add this allele to the cycle and repeat step the above until you arrive at the first allele of parent1. Put the alleles of the cycle in the first child on the positions they have in the first parent and the remaining alleles of first child come from the second parent along with their position. Generate next cycle from parent2

6. **TrajectoryCrossover:**This crossover produces children by exploring the trajectory that connects both parents. The procedure begins with one of the parents, called the initial solution. Then a random position is chosen for both parents. The elements located at this position are permuted which builds two other solutions. The fitness values of these two solutions are calculated and the bad solution is eliminated, to consider only the initial solution and the winning solution. The previous procedure is then repeated on the element of the following position until all the positions have been considered to obtain child 1. This crossing will be well illustrated in the following We applied the crossover to our work.

This crossing is envisaged with two parents and only one child. The child is produced by swapping alleles between the two parents starting at a random locus, the example shows the method .We chose a position randomly, in our example, we chose position 12 on the parents and then we crossing between them, then we change each team i to team j and each team j to team i.

	Round1		Round2		Round3		Round4		Round5		Round6	
Parent 1	(4 ,1)	(3,2)	(2,1)	(3,4)	(3,1)	(4,2)	(1,4)	(2,3)	(1 ,2)	(4,3)	(1 ,3)	(2,4)
Parent 2	(3 ,1)	(2,4)	(2,1)	(4,3)	(4,1)	(2,3)	(1,3)	(4,2)	(1 ,2)	(3,4)	(1 ,4)	(3,2)

Chase position 12 :

	Round1		Round2		Round3		Round4		Round5		Round6	
Parent 1	(4 ,1)	(3,2)	(2,1)	(3,4)	(3,1)	(4,2)	(1,4)	(2,3)	(1 ,2)	(4,3)	(1 ,3)	(2,4)
Parent 2	(3 ,1)	(2,4)	(2,1)	(4,3)	(4,1)	(2,3)	(1,3)	(4,2)	(1 ,2)	(3,4)	(1 ,4)	(3,2)

	Round1		Round2		Round3		Round4		Round5		Round6	
child 1	(4 ,1)	(2,3)	(3,1)	(2,4)	(2,1)	(4,3)	(1,4)	(3,2)	(1,3)	(4,2)	(1,2)	(3,4)
child 2	(2 ,1)	(3,4)	(3,1)	(4,2)	(4,1)	(3,2)	(1,2)	(4,3)	(1,3)	(2,4)	(1 ,4)	(2,3)

The fitness values of these two solutions Child 1 and Child 2 are calculated after we choose the best Child and we get rid of the worst solution, we repeat the crossover between parent 1 or 2 whit the best Child, in this example we chase the Child 2. Then the previous action is repeated on the next element (position 13).) the Parents 1and 2 are swapped to get solutions 3 and 4, after calculated The fitness values of these two solutions for rid of the worst solutions, etc

If the elements located at a selected position are identical in both solutions, the next position is considered. This procedure is repeated until all positions have been considered to obtain Child 1.
Child:

	Round1		Round2		Round3		Round4		Round5		Round6	
Parent 1	(4 ,1)	(3,2)	(2,1)	(3,4)	(3,1)	(4,2)	(1,4)	(2,3)	(1 ,2)	(4,3)	(1 ,3)	(2,4)
Parent 2	(4 ,1)	(2,3)	(3,1)	(2,4)	(2,1)	(4, 3)	(1,4)	(3,2)	(1 ,3)	(4,2)	(1 ,2)	(3,4)

	Round1		Round2		Round3		Round4		Round5		Round6	
Child 1	(4 ,1)	(2,3)	(3,1)	(2,4)	(2,1)	(4,3)	(1,4)	(3,2)	(1 ,3)	(4,2)	(1 ,2)	(3,4)
Child 2	(4 ,1)	(3,2)	(2,1)	(3,4)	(3,1)	(4, 2)	(1,4)	(2,3)	(1 ,2)	(4,3)	(1 ,3)	(2,4)

(4 ,1)	(2,3)	(3,1)	(2,4)	(2,1)	(4,3)	(1,4)	(3,2)	(1 ,3)	(4,2)	(1 ,2)	(3,4)

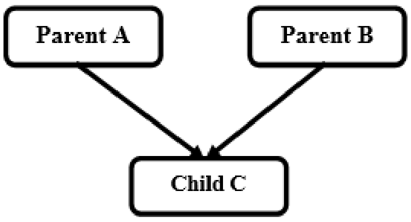


Figure 3.3: Trajectory Crossover

3.3.4 Mutation

After cross over, the chromosomes are subjected to mutation. It is the process by which a string is deliberately changed so as to maintain diversity in the population set. Mutation of a bit involves flipping it, changing 0 to 1 and vice-versa with a small mutation probability Pm. Mutation Probability determines how often the parts of a chromosome will be mutated. Types:

1. Flipping
2. Interchanging
3. Reversing Mutation Hence

mutation causes movement in the search space (local or global) and restores lost information to the population.
We applied the mutation to the new children by applying swaps and this using the local search algorithm, as we explained in the previous part (population part) with the use of the Aspiration Technique.

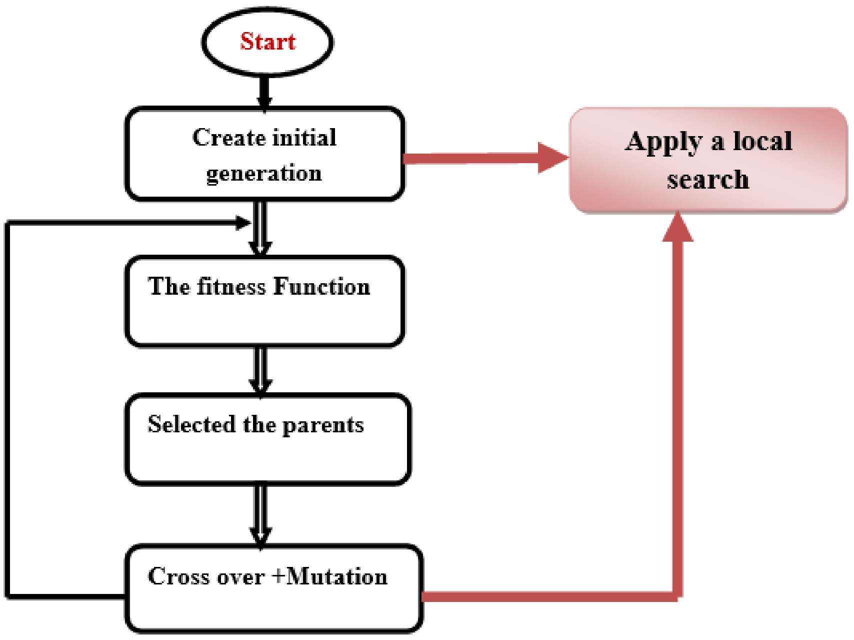


Figure 3.4: The proposed approch genetic algorithm

3.4 Conclusion

In this chapter we presented a detailed description of our method to solve the TTP by combining two approaches GA + LOCAL SEARCH with the application Aspiration Technique to Select the Best Neighbor. In the next chapter we will present the results and compare them with the previous results

CHAPTER 4

Numerical Results

4.1 Introduction

Finally in this last chapter we present the numerical results found by the proposed approach based on a approximate method and Evolutionary algorithms Genetic algorithm (GA) and Local search Our application was created to contribute to the solution of the Traveling Tournament problem (TTP) and then we analyze the results and compare the results with previous results .

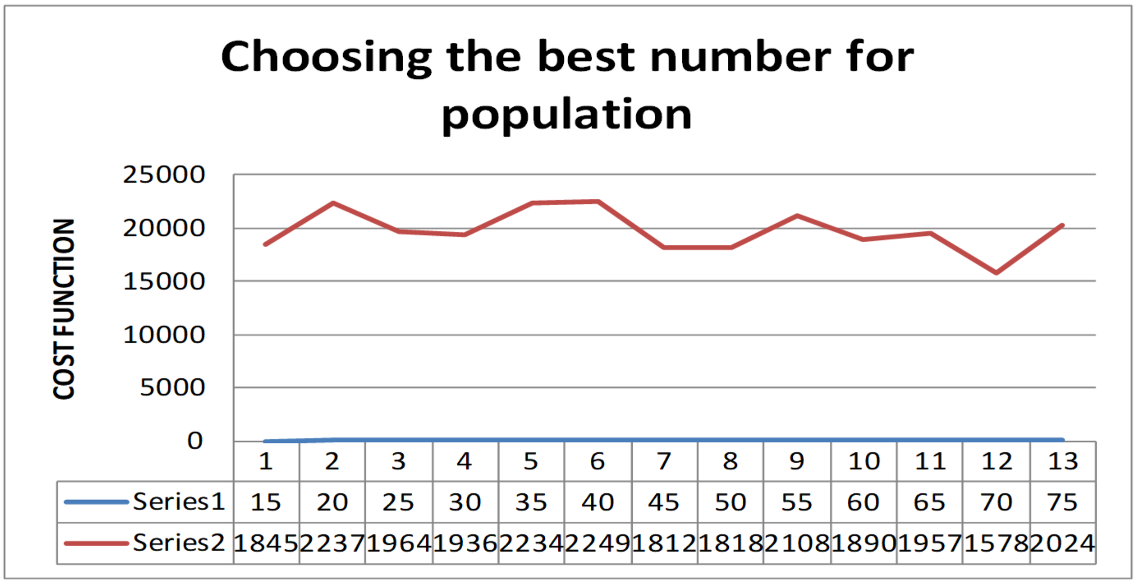
4.2 the programing language using

PYTHON In our work we use the python language is a programming language created by Guido van Rossum , It has many advantages compared to other methods or languages

- It is used to build websites and software and perform data analysis.
- It facilitates the implementation and testing of algorithms.
- It is allowed to draw graphs and develop applications with a graphical user interface.

4.3 Choosing the best Values for population

Choosing the best Values for population the parents in order to apply the genetic algorithm :
We have fixed the selection values in 0,7, and we implemented the algorithms with 100 iterations The results represented in the following curve:



	1	2	3	4	5	6	7	8	9	10	11	12	13
Series1	15	20	25	30	35	40	45	50	55	60	65	70	75
Series2	1845	2237	1964	1936	2234	2249	1812	1818	2108	1890	1957	1578	2024

Figure 4.1: Curve population change with respect to cost on the number of teams 6

4.4 Choosing the best Values for select the parents in order to apply the genetic algorithm

We have fixed the selection population in 100, and we implemented the algorithms with 100 iterations The results represented in the following curve:

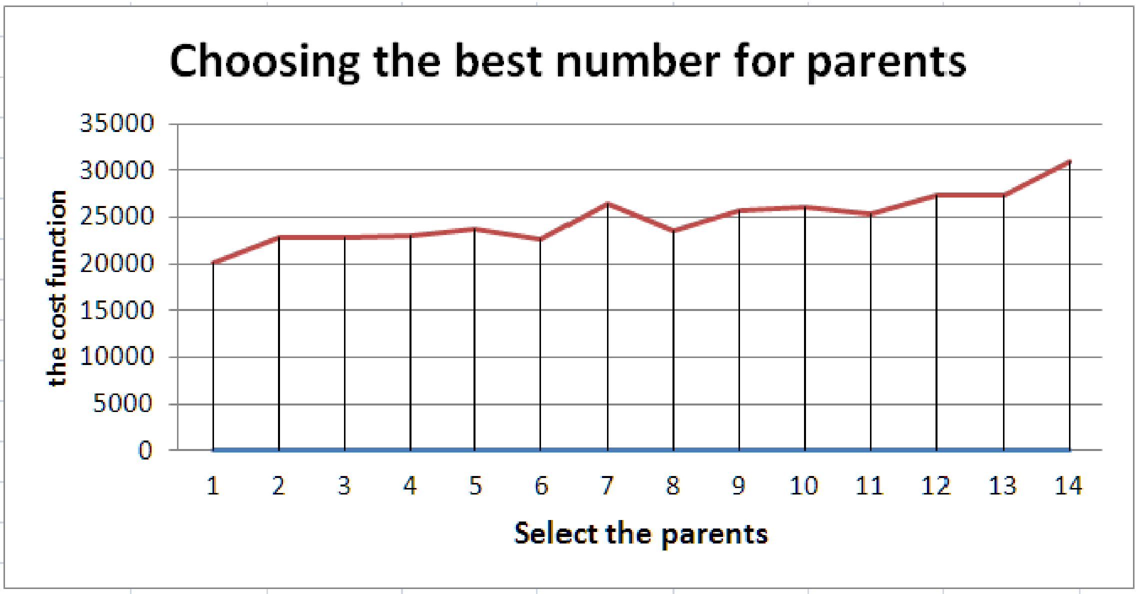


Figure 4.2: Curve population change with respect to cost on the number of teams 6

4.5 Results and analysis

The corresponding matrix and curve represent one of the best solutions we have obtained with the best population number and the best selection number and iteration=100 , it's minimize the distance to 21400

team 1	[[-6 -2 5 3 4 -3 -4 2 6 -5]
team 2	[-3 1 4 -5 6 5 -6 -1 3 4]
team 3	[2 4 6 -1 -5 1 5 -4 -2 6]
team 4	[5 -3 -2 -6 -1 6 1 3 5 -4]
team 5	[-4 -6 -1 2 3 -2 -3 -6 -4 1]
team 6	[1 5 -3 4 -2 -4 2 5 -1 -6]]

Table 4.1: Matrix represent one of the best solutions on the number of teams 6

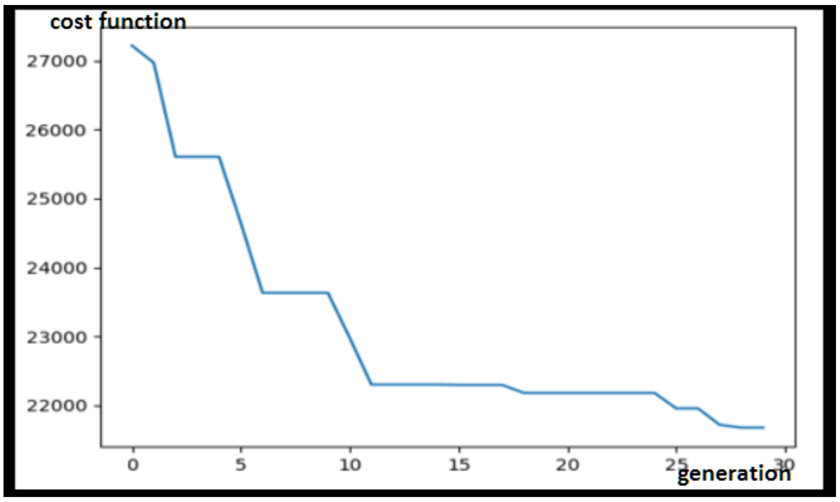


Figure 4.3: the best solutions we have obtained on the number of teams 6

4.6 Numerical Results

Table gives the numerical results found by the overall approach. We give the CPU time (Time) in seconds (the reported time is the time needed to find the best solution), the best (Best) and the average solution (AVERAGE) of twenty executions found by our method. We give the best known solutions (Feasible Solution) for each instance and the gap between Best and Best-known.The best results are in bold font. The proposed method SLS is compared with the best-known solutions Feasible Best-known for TTP in order to show its Best-known TTP [14]

Gap% = (Best-known - Best/ Best-known)*100

Instance	Best-known	Best	Worst	Average	CPU times	Gap %
NL4	——	8492	10433	9201	8492	
NL6	19900	21856	28800	25570	105.67	-0,000097
NL8	30700	32901	40568	36432	284.53	-7,169
NL10	45309	48324	55543	51432	395.33	-6,654
NL12	79623	80979	90982	86750	722.88	-1,703
NL14	125734	136398	206784	176891	1471.43	-8,481
NL16	154623	167484	218484	3316.34	186753	-8,316

Table 4.2: Numerical results found by the overall approach

4.7 Discussion:

After we chose the best value for population, the best value for choosing the new generation and mutation, we analyzed the results as shown in the curve graph Figure 4.3, after applying both the genetic algorithm and the local search, we found that the genetic algorithm contributed to Determine the minimum distance ,the local search contributed to building a schedule satisfies the double round robin tournament In the various Benchmarks.

4.8 Conclusion

In this chapter we applied the proposed model (GA+LOCAL SEARCH), then we compared our results with the Previous results. From the experimental results we conclude We conclude that this duality helps to achieve the best fitness while maintaining the health of the schedule

GENERAL CONCLUSION

The well-known NP-Hard TTP problem is difficult to solve optimally due to the problem formulation containing both optimization goal and integer feasibility constraints. This work proposes a novel hybrid approach of Genetic algorithm and Iterated local search heuristics to generate schedules for TTP. The proposed method starts with an initial population generated by using the polygon method. Then, the population of solutions is modified to a new population by applying three operators selection, crossover and mutation. Further, we apply the local search heuristic as an improvement strategy to enhance the solution quality. The results show that our approach could build interesting results comparable to other state-of-the-art approaches

Bibliography

[1] T. V.Mathew, "Genetic algorithm," *Indian institute pf technology Bombay, Mumbai.*

[2] E.-G. Talbi, "Metaheuristics from design to implementation," *University of Lille – CNRS – INRIA*, 2009.

[3] F. Peres and M. Castelli, "Combinatorial optimization problems and metaheuristics: Review, challenges, design, and development," *Appl. Sci*, no. 6449, 11 2021.

[4] A. LAYEB, "Utilisation des approches d'optimisation combinatoire pour la vérification des applications temps réel." *Thèse de Doctorat , Université Mentouri de Constantine*, 2010.

[5] S. K. Ichrak and D. Boutheina, "(aco+gpe) for solving the traveling salesman problem (tsp)," *Master's Thesis , University of Mohamed El Bachir El Ibrahimi of Bordj Bou Arreridj Faculty of Mathematics and Computer Science*, 2020/2021.

[6] C. Lewis, "Linear programming: Theory and applications," May 2008.

[7] A. Pattanayak, "Artificial intelligence local search," *Department of Computer Science, Raja N. L. Khan Women's College (Autonomous)*, May 2020.

[8] A. Gherboudj, "Méthodes de résolution de problèmes difficiles académiques," *Thèse de Doctorat, Université de Constantine2*, 2012/2013.

[9] . C. S. H. Zar and A. K. May, "Solving traveling salesman problem by using improved ant colony optimization algorithm," *International Journal of Information and Education Technology*, vol. 1, no. 5, December 2010.

[10] R. Lakehal-Ayat, "Computational complexity complexité algorithmique," *Msila university*, 2019.

[11] C. C. RIBEIRO, "Sports scheduling: Problems and applications."

[12] G. L. N. Kelly Easton1 and M. Trick3, "A history of the traveling tournament problem kelly easton1," *Kansas State University, Manhatten.*

[13] M. khelifa and dalila boughaci, "A cooperative local search method for solving the traveling tournament problem," *computer science department university of beb-ezzouar.*

[14] "Challenge traveling tournament instances." *http://mat.tepper.cmu.edu/TOURN/*, 2016.