School of Computer Science and Informatics

CM3203 – Individual Project Report, 08/05/2025

Solving the Travelling Tournament Problem

Author: Yuv Raikhy

Supervisor: Nico Potyka

Abstract

This project investigates the application of a genetic algorithm to the classical Travelling Tournament Problem (TTP), a well-known combinatorial optimization challenge in sports scheduling. The TTP involves generating a double round-robin tournament schedule for a given set of teams while minimizing total travel distance and satisfying strict feasibility constraints related to game sequencing and venue assignments. Although genetic algorithms have been applied to this problem previously, the flexible nature of evolutionary approaches allows for continued innovation in their design. In this project, a customized genetic algorithm was implemented, featuring a problem-specific representation, a tailored crossover operator, and a case-based fitness function. The algorithm was tested on standard benchmark datasets (NL and CIRC instances) and compared against best-known results. It successfully produced optimal or near-optimal solutions for small problem instances (4–6 teams), and feasible but suboptimal solutions for medium instances (8 teams), while failing to find feasible solutions for larger instances (10 teams). Comparative evaluations also showed that the custom crossover outperformed standard k-point crossovers in both solution quality and constraint satisfaction. These findings demonstrate the potential and limitations of standalone genetic algorithms for solving the TTP and suggest avenues for future improvement through hybridization with other metaheuristic techniques.

Acknowledgements

I would like to thank my supervisor Dr. Nico Potyka for his advice and feedback throughout the project.

Artificial Intelligence was used in this project to assist with wording and the structure of this document and in assisting with programming. However, all research, conclusions, final code and analysis are my own.

Table of Contents

Abstract 2
Acknowledgements 3
Table of Contents 4
1. Introduction 6
2. Background 8
2.1 Problem definition 8
2.2 Literature Review 9
2.3 Complexity 10
2.4 Genetic Algorithms 10
3. Approach 13
3.1 Solution Overview 13
3.2 Algorithm 14
3.2.1 Genome representation 14
3.2.2 Population initialization 16
3.2.3 Selection 17
3.2.4 Crossover 17
3.2.4.1 Opponent-based Crossover 18
3.2.4.2 Structure-based Crossover 18
3.2.5 Mutation 20
3.2.6 Fitness Function 21
4. Evaluation 23
4.1 Overall Results 23
4.2 Custom Crossover vs K-point crossover 24
4.3 Configuration Testing 26
4.3.1 Population Size 26
4.3.2 Other 29
4.3.2.1 Tournament Size 29
5. Conclusion and Future Work 30

6. Reflection 31
Appendices 32
References 34

1. Introduction

Sports leagues are major commercial enterprises generating millions in revenue worldwide, with the UK's Premier League being particularly prominent due to significant income streams such as broadcasting rights and sponsorships. Crucial to maximizing revenue and audience engagement is the effective scheduling of matches. Optimal schedules ensure fairness by preventing excessive travel burdens and player fatigue, which can disadvantage visiting teams facing opponents on extended home stands. Additionally, attractive and competitive fixtures are strategically scheduled for prime viewing dates, such as Boxing Day in the Premier League, to justify high broadcasting investments and maintain fan interest, the principal revenue source for leagues. Not only this but the fans, the main source of income for the leagues, are also affected by the schedule of a league therefore this poses to be an interesting as well as a potentially useful problem.

The Travelling Tournament Problem (TTP) is a well-known combinatorial optimization problem that arises in the context of sports scheduling, originally introduced by Easton, Nemhauser, and Trick (2001) [1]. In this problem, given a set of teams we must output a double round robin tournament such that the travel distance is minimized given that the length of each home stand and road trip is at least L and at most is U (L and U inclusive). A double round robin tournament is a competition format in which each contestant meets every other participant twice. The length of the home stand or road trip refers to the amount of home games or away games that consecutively

Addressing the Travelling Tournament Problem is not only practically beneficial for leagues aiming to optimize schedules but also academically significant, as advancements in solving TTP can lead to improvements in combinatorial optimization methods broadly used in logistics, transportation, and operations management. Since it was introduced, TTP has gained significant attention in operations research and artificial intelligence communities due to its real-world relevance and computational complexity.

Numerous optimization techniques have been applied to address the Travelling Tournament Problem including heuristic approaches, exact approaches and evolutionary algorithms. Although evolutionary algorithm approaches have previously been applied to the Travelling Tournament Problem, the inherent flexibility and diversity in genetic algorithm methodologies provide ample opportunity for further exploration and experimentation. This project aims to produce a solution to the classic variation of the TTP using a genetic algorithm approach.

This report discusses the solution that has been produced as part of this project. Section 2 of this report provides the background and formal definition of the Travelling Tournament Problem, accompanied by a literature review examining previous approaches and methodologies. Section 3 outlines the genetic algorithm approach developed in this project, detailing the components and procedures utilized. Finally, Section 4 presents the evaluation methodology and discusses the experimental results obtained, highlighting the effectiveness and efficiency of the proposed

solution. Section 5 presents the project's conclusion and highlights parts of the problem where future work can be done.

2. Background

2.1 Problem definition

The Traveling Tournament Problem (TTP) was introduced by Easton, Nemhauser, and Trick in 2001, the TTP attempts to create an optimal schedule for a set number of teams where the objective is to minimize the total distance traveled by all teams throughout the tournament while adhering to various constraints related to time and venue [1]. TTP has many variations, however in this project the focus is on the classic variation.

The classic TTP can be defined as:

Let $T = \{t_1, t_2, \ldots, t_n\}$ be a set of n teams, where n is an even integer. The number of teams n must be even because in a round-robin tournament every team must play exactly one match per round, a round refers to a time slot / match day during which each team plays exactly one match, home or away, a match requires two teams and if n were odd then there would always be one team that is not playing in that round (this is called a bye), double round-robin tournaments are designed under the assumption that no byes are allowed. Let D be an $n \times n$ distance matrix where D_{ij} represents the distance between the home venues of teams $t_i t_j$. A double round-robin tournament requires that each team play against every other team exactly twice: once at its home venue and once at the opponent's venue.

In a round-robin tournament with n teams, each team plays every other team once hence each team plays $n-1$ games in total, if this is extended to a double round-robin then each team will play $2(n-1)$ games in total. Therefore, a double round-robin tournament schedule S consists of $n(n-1)$ matches played over $2(n-1)$ rounds (slots), where each team plays exactly one match per round. Each match is defined by an ordered pair (t_i, t_j) indicating that team t_i plays at home against the visiting team t_j.

For a valid tournament schedule S , the following constraints must be satisfied:

1. **Double Round-Robin Constraint**: For each pair of distinct teams $t_i, t_j \in T$, both matches (t_i, t_j) and (t_j, t_i) must appear exactly once in S .
2. **No-Repeater Constraint**: If team t_i plays against team t_j in round r , then it cannot play against it in round $r + 1$.
3. **Home-Away Pattern Constraint**: No team may play more than k consecutive home games or k consecutive away games, where k is a specified parameter (typically $k = 3$).

Let $dist(t_i, r)$ denote the distance that team t_i must travel to reach the venue where it plays its match in round r . Initially, each team starts at its home venue before round 1. If team t_i plays at home in round r , $dist(t_i, r)$is zero. Otherwise, $dist(t_i, r)$ is the distance from the team's

previous location (either home or the venue of the previous away match) to the venue of its opponent in round r . After the final match, if the team is not already at its home venue, an additional distance is incurred for returning home, and this is included in the total travel distance. The objective of the Traveling Tournament Problem is to find a valid tournament schedule S that minimizes the total distance traveled by all teams across the entire tournament.

$$\text{Objective: } \min S \ \sum_{i=1}^{n} \left(\left(\sum_{r=1}^{2(n-1)} dist(t_i, r) \right) + dist_h(t_i) \right)$$

where $dist_h(t_i)$ refers to the final travel back home if the final game was away, and 0 if it was home.

2.2 Literature Review

Research has extensively focused on optimizing solutions for the TTP through various computational strategies. Notable methods include integer programming, which formulates the problem in mathematical terms to derive optimal solutions and constraint programming, which addresses conflicts and enforces scheduling restrictions dynamically [2]. Metaheuristics, particularly Simulated Annealing (SA), have been widely applied [3] [7] [8] to the Traveling Tournament Problem (TTP) to effectively navigate the solution space. The work by Anagnostopoulos et al. [3] highlights the capabilities of SA, which adeptly explores both feasible and infeasible schedules. This algorithm employs a large neighborhood search, incorporating strategic oscillation techniques to escape local extremities and explore varied configurations while effectively managing hard and soft constraints of the tournament schedule.

Haldar et al. [4] discuss the limitations of genetic algorithms (GAs) within highly constrained environments, presenting empirical evidence indicating that under certain conditions, GAs may struggle to converge on satisfactory solutions. GAs utilises sophisticated representation and operations, crossover and mutation, to explore the solution space diversely, enhancing the algorithm's potential to navigate past local minima, a persistent issue in sports scheduling. Choubey elaborates on GAs by proposing a novel encoding scheme that explicitly encodes who is playing the match and where the game is being played (home / away). The encoding scheme optimizes the representation of solution instances, which may increase the algorithm’s effectiveness in managing the complex constraints [5]. Verduin et al, in their study [6], critically examine the limitations of standard genetic algorithms when confronted with the additional scheduling constraints of TTP like how many home or away games can be played in a row, add an extra challenge to the problem when compared to problems like the Travelling Salesman Problem where the goal is to simply minimize distance. The authors argue that while evolutionary computing is a powerful paradigm for many optimization problems, the dense constraints of the TTP can render these methods less effective unless significant adaptations or hybridizations are introduced. Khelifa et al [10] in their work present a significant advancement in solving the TTP through their enhanced genetic algorithm (E-GA), which introduces a novel crossover operator and integrates variable neighborhood search (VNS). The new crossover

operator shares the best partial path teams (BPPT) among schedules thereby preserving efficient sub-schedules and enhancing offspring quality. This tailored crossover improves the feasibility and performance of generated schedules compared to standard recombination techniques. Empirical results on standard benchmark instances (NL, CON, and CIRC) demonstrate that their E-GA consistently finds optimal or near-optimal solutions and matches several other state-of-the-art methods. This paper enriches the TTP literature by combining intelligent solution recombination with powerful local search, and by addressing practical scheduling flexibility thereby offering a robust and adaptable framework for tournament scheduling.

In conclusion, the TTP remains an engaging and intricate challenge in combinatorial optimization, where methodologies extend from straightforward integer programming to more complex approaches involving genetic algorithms. Significant contributions from past studies illuminate the contours of various TTP variants and associated solution strategies, establishing a foundation for both theoretical exploration and practical applications in sports scheduling.

2.3 Complexity

TTP is considered an NP-hard problem which means that the time required to solve it increases rapidly with the number of teams which makes exact approaches computationally infeasible for large instances. The TTP serves as a challenging benchmark for developing and testing optimization techniques. The combination of routing and scheduling constraints makes it more complex than classic problems like the Travelling Salesman Problem (TSP), to which it is closely related. While the TSP focuses solely on minimizing travel distance for a single agent, the TTP introduces multiple agents (teams) and temporal constraints (scheduling) increasing its difficulty significantly.

2.4 Genetic Algorithms

A genetic algorithm (GA) is an optimization method inspired by the principles of natural selection and biological evolution. Introduced by John Holland in the 1970s, GAs aim to find approximate solutions to problems by evolving a population of candidate solutions over successive generations [9].

A typical genetic algorithm begins with an initial population of individuals, each representing a potential solution. The quality of each individual is evaluated using a fitness function, which quantitatively measures its performance with respect to the objective. Individuals are selected for reproduction based on their fitness, favoring higher-quality individuals. New individuals are generated through genetic operators: crossover and mutation. In crossover, the two parents are selected from the pool of high-quality individuals, and they go through a function which recombines parts of two parent solutions to create two new offsprings. In mutation, some random alteration is introduced to some individuals in the population, this is done to increase genetic

diversity in the population. The idea is that through repeated application of selection, crossover and mutation, the population is expected to converge towards more optimal, higher fitness, solutions.

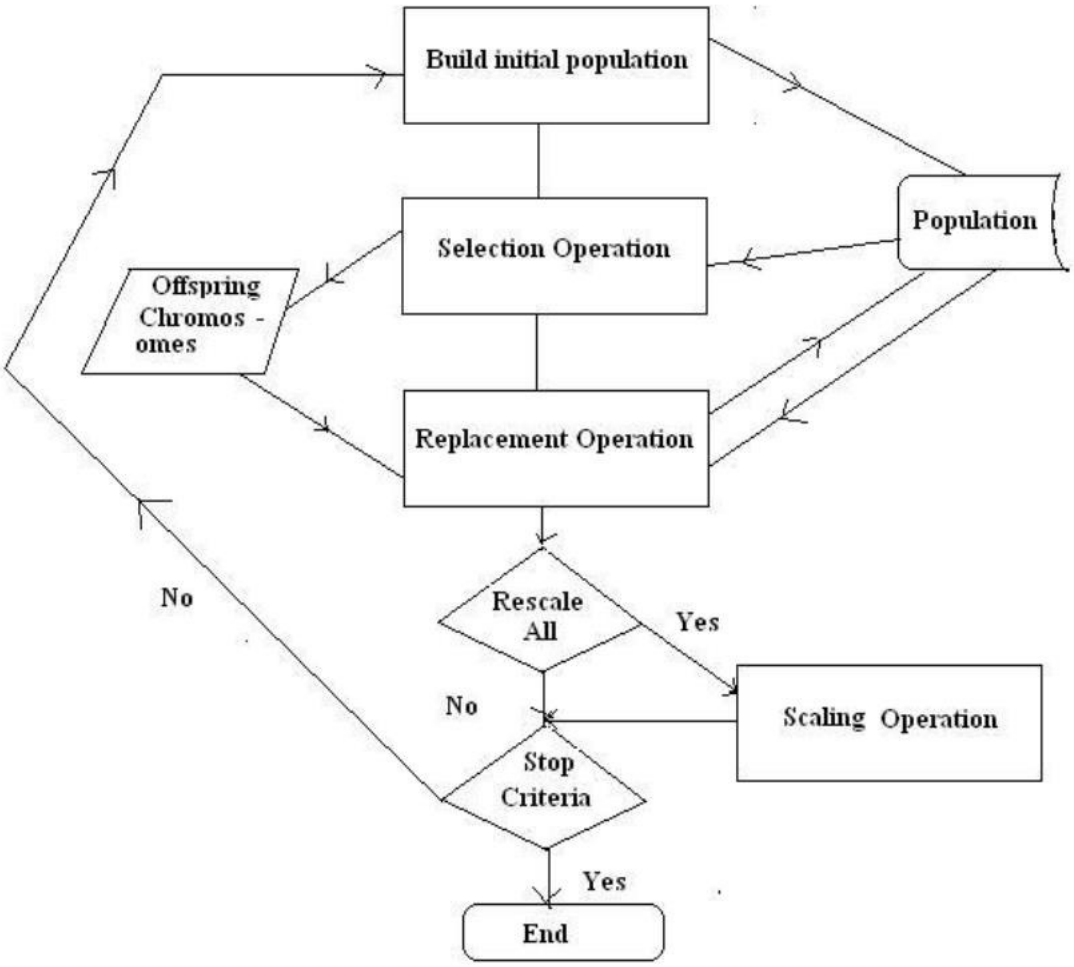

Figure 1 Example flow of a genetic algorithm [5]

A widely adopted selection method used in GAs is Tournament Selection. This is due to its simplicity and effectiveness. In each selection event, a subset of individuals is randomly sampled from the population. The individual with the highest fitness score within the tournament is then selected to contribute to the next generation. This process is repeated until the number of selected individuals equals the size of the original population. Tournament selection maintains a balance between selecting a diverse group of individuals and selecting the fittest individuals; by selecting the best individual among a small random group, it promotes fitter individuals while still allowing diversity through random sampling. The size of the tournament (tournament size) acts as a parameter controlling selection pressure. If the tournament size is too high then more of the same fittest individuals will enter the selection pool, if it is too low then more poor individuals

will also be selected. If the tournament size is just about right, then the fittest individuals will be selected but there will also be enough diversity present in the selection pool to explore various solution spaces.

For the crossover operation (the replacement operation in the flowchart), K-point crossover is a widely used genetic algorithm technique designed for combining genetic information from parent solutions to produce offspring that potentially inherit advantageous traits from both parents. The method involves selecting 'k' distinct crossover points within parent chromosomes, which divide the chromosomes into segments. Offspring are then generated by alternately exchanging these chromosome segments between two parents. This process effectively recombines genetic material, promoting genetic diversity and enabling exploration of the solution space. Commonly used values for 'k' include one-point and two-point crossover, as these provide a good balance between maintaining beneficial gene sequences and promoting sufficient genetic diversity. One-point crossover is often preferred for its simplicity and minimal disruption of gene sequences, while two-point crossover is favored when greater genetic recombination is desirable without overly fragmenting gene groups.

For the mutation operator, common approaches include bit-flip mutation, in which binary bits in chromosomes are randomly flipped. Swap mutation, where two positions in a chromosome are selected at random and their values exchanged. And inversion mutation, which involves selecting a segment of the chromosome and reversing its order. The choice of mutation method and mutation rate typically depends on the specific problem and the desired balance between exploration and exploitation.

3. Approach

3.1 Solution Overview

This section builds upon the general genetic algorithm flowchart introduced in Section 2 (Figure 1). The following subsections detail how each component—representation, initialization, selection, crossover, mutation, and fitness evaluation—has been used and adapted for solving the TTP.

In this genetic algorithm, each individual genome encodes a complete tournament schedule and through a combination of selection, crossover, and mutation, the population evolves toward schedules that minimize total travel distances while respecting feasibility requirements.

The initial population is generated through random initialization and are replaced by the children after crossover or the elite individuals from the selection pool if crossover could not be performed. This algorithm uses a problem-specific representation and a tailored crossover operator that makes use of the unique structure and constraints of the TTP. For selection and mutation, well-known methods such as tournament selection and swap mutation are used.

A case-based fitness function is used that evaluates individuals differently based on feasibility: feasible solutions are evaluated based on their total travel distance, while infeasible solutions are penalized according to the degree of constraint violation. This dual evaluation approach encourages the population to prioritize feasibility first and optimize travel distance once feasibility is achieved. Reaching a maximum number of generations is used as the stopping criterion for this algorithm.

The algorithm is implemented in Python and has been divided into several files divided based on their function. The files have been uploaded alongside this report on PATS. Figure 2 (See below) shows all the major Python files and classes that are used to implement the genetic algorithm, together with their methods, and inter-module relationships.

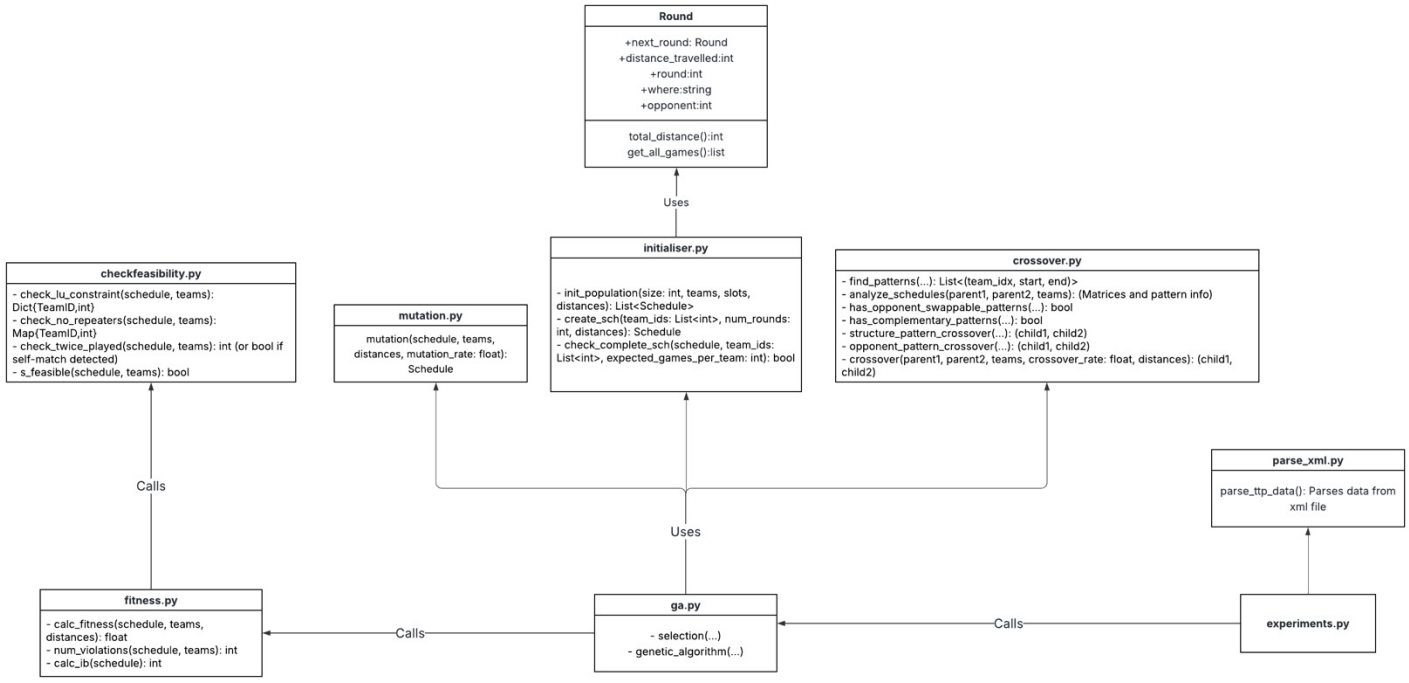

Figure 2: Shows the relationship between each of the files, their main functions, and attributes. (In the diagram, '...' is used where there are too many parameters to list and will not fit in the diagram)

The data, from the data sets, used to test and benchmark algorithms for TTP come in the form of an xml file, the 'parse_xml' file converts this data and puts it into dictionaries (data structure) that can be then used. Further information about the data used has been given in the evaluation section. The experiments.py file is where the genetic algorithm is called for the testing and evaluation purposes for this report.

3.2 Algorithm

3.2.1 Genome representation

In this genetic algorithm approach to solving the TTP, each individual solution encodes a complete tournament schedule. The genome structure is designed to represent the sequence of opponents across all rounds of the tournament for each team. Each team's schedule is stored as an ordered sequence of matches organized by rounds. A round refers to a time slot / match day during which each team plays exactly one match, home or away. To illustrate the structure of the genome representation, Table 1 presents an example schedule, for four teams, where the rows correspond to rounds and the columns correspond to teams. Each entry indicates the opponent a particular team faces in each round and the match's location (home or away).

Table 1:

Teams / Rounds	T1	T2	T3	T4

R1	T3 (home)	T4 (away)	T1 (away)	T2 (home)
R2	T2 (away)	T1 (home)	T4 (home)	T3 (away)
R3	T4 (home)	T3 (home)	T2 (away)	T1 (away)
R4	T3 (away)	T4 (home)	T1 (home)	T2 (away)
R5	T4 (away)	T3 (away)	T2 (home)	T1 (home)
R6	T2 (home)	T1 (away)	T4 (away)	T3 (home)

While the tournament schedule is conceptually visualized as a table, the genome representation is implemented as an array of linked lists. More specifically, the genome (schedule) is modeled as an array of linked lists with each array entry corresponding to a specific team in the tournament. The schedule array *S* stores the root nodes of all the teams' linked lists.

$$S = [t_1, t_2, \ldots, t_n] \text{ where } t_i \text{ is the root node of the singly linked list}$$

The linked list is constructed using a class called Round, as each node in the linked list corresponds to a round in the tournament schedule. Each node stores the round number, opponent's information, location (home or away), the distance travelled (if the game was away) and the pointer to the next round (See Figure 3). To interpret a genome and derive the corresponding tournament schedule, each team's linked list is traversed sequentially and that team's schedule can be constructed by noting the opponent and round information stored at each node. Thus, the entire tournament schedule emerges by iteratively decoding each team's linked list.

During population initialization a schedule array *S* is created and a root node representing round 0 of the schedules is added for each team. From then on, every time a new round is to be added, a new round node which stores the information for that new round can be created and the previous node can be made to point at that new node.

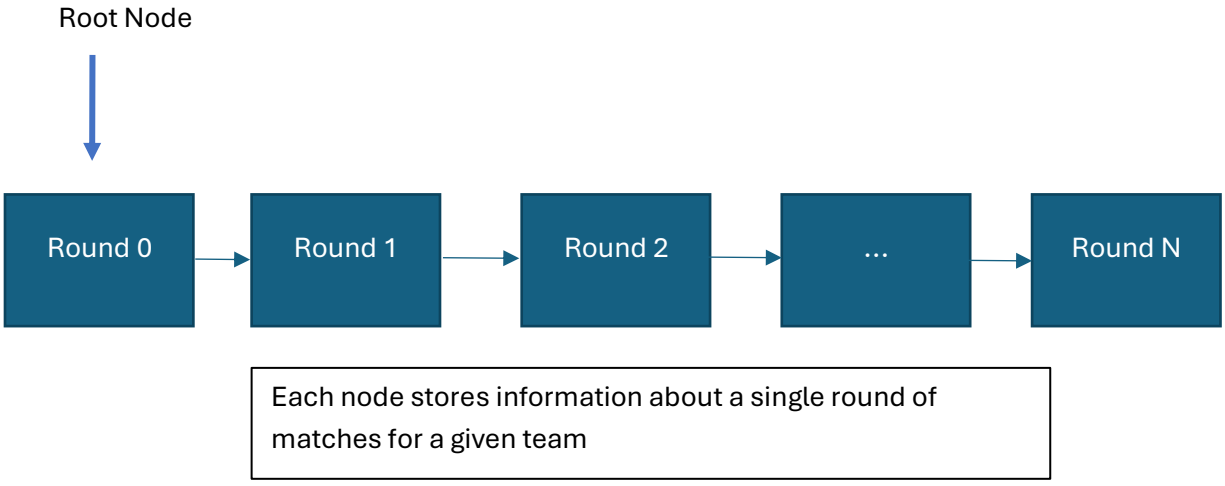

Figure 3: Visual representation of the linked list.

This representation based on linked lists is more computationally efficient compared to alternative structures such as a two-dimensional (2D) array. Although a 2D array might initially offer simplicity in implementation, the complexity increases significantly when operations such as swapping elements, iterating through multiple scheduling rounds, and dynamically modifying the schedules are required. Specifically, a 2D array necessitates extensive nested looping constructs to traverse rows and columns, thus introducing substantial overhead and increased algorithmic complexity. Consequently, the execution time for operations involving frequent insertions, deletions, and element swaps becomes higher. In contrast, a linked list-based representation simplifies dynamic data manipulation. Nodes representing individual rounds can be accessed directly, facilitating efficient movement and rearrangement of elements within the schedule. Such direct access and manipulation significantly reduce the computational overhead inherent to nested loop traversals required by 2D arrays. Additionally, linked lists inherently support dynamic memory allocation, allowing schedules to adapt flexibly without incurring substantial computational cost. Therefore, the linked list structure used in the genome representation provides a substantial long-term advantage in efficiency, particularly in operations involving dynamic modifications and frequent swapping of elements.

3.2.2 Population initialization

Population generation is performed through random initialization. Random initialization is a common strategy because it promotes genetic diversity at the beginning of the evolutionary process, helping to avoid premature convergence to suboptimal regions of the search space. By creating individuals with randomly assigned genes, the algorithm is better able to explore a broad range of potential solutions during the early generations. The process is handled in the ‘initialiser.py’ file.

For every team, a linked list is initialized with a root node that serves as a placeholder. All possible home-away matchups between teams are generated and randomly shuffled to introduce diversity. Matches are then sequentially assigned to rounds, ensuring that teams are available and that there are no invalid assignments i.e., no team has played home or away games against that same opponent more than once. As matches are inserted, new nodes are created using the Round class and a pointer is created to that new node at the last node of each team's linked list.

Due to the strict constraints of TTP, not all randomly generated solutions are feasible and if unchecked no feasible solutions may exist in the initial population. To fix this, initially, only feasible solutions are admitted into the population. This occurs with a certain number of predefined attempts which increases the runtime slightly. This may be the reason for larger TTP instances, instances with 8 or more teams, a larger initial population size is required for enough diversity to generate a feasible solution (See Section 4.3.1). During this phase, candidate schedules are generated randomly and then checked for feasibility; feasible solutions are added to the population while infeasible ones are discarded. If, after this predefined number of

attempts, the desired population size has not been reached, infeasible solutions are also admitted in order to complete the population. However, all members added to the population must be structurally complete i.e., they all must play the expected number of games, one match per team per round and they do not play a team more than once at home and away. This approach ensures that the initial population contains as many feasible schedules as possible while guaranteeing that the target population size is eventually met.

3.2.3 Selection

The selection mechanism used in this genetic algorithm is based on tournament selection (See Section 2.4). The selection mechanism has been implemented within the main ga.py file as the algorithm is concise enough that it does not need a separate file.

The algorithm works as follows, a random subset of the population—referred to as a "tournament"—is sampled. The size of this subset is determined by the ‘tournament_size’ parameter. From this subset, the individual with the highest fitness is selected and added to the selection pool. This process is repeated until the number of selected individuals equals the size of the original population.

Overview of the selection mechanism:

1. Sampling: A subset of individuals is randomly sampled from the population.
2. Feasibility Check: Each sampled individual is evaluated for feasibility based on its constraint satisfaction status.
3. Fitness Evaluation:
    a. If one or more individuals in the tournament subset are feasible, the one with the highest fitness among the feasible individuals is selected.
    b. If none are feasible, the individual with the highest fitness is chosen.

3.2.4 Crossover

The crossover operator in this genetic algorithm is designed to use the structural patterns within tournament schedules. Unlike standard random or fixed-point crossover techniques, this operator analyzes parental schedules to identify repeating patterns, complementary structures, and interesting sub-sequences. Based on this analysis, the algorithm selects between different crossover strategies: a structure-based crossover or an opponent-based crossover.

The process begins by analyzing the home/away patterns of each team's schedule in both parents. These schedules are first converted from their representation into matrix form to allow patterns

such as consecutive sequences of home or away games, alternating sequences, or repeating subsequences to be detected. A minimum length and width, based on the number of rounds and teams, is determined to ensure that only a subsection (submatrix) of the whole schedule (matrix) is considered for crossover.

Overview of the analysis procedure:

- Choose length of the submatrix based on number of rounds and teams.
- Build round/team matrices to map each round of matches to an (opponent, home/away) tuple
- Extract binary patterns ("home"/ "away") per team
- Locate runs of identical, alternating, or repeating subsequences by:
    - Slides a fixed-length window over each team's pattern
    - Flags windows that are all-same, alternating, or perfect repeats

After pattern analysis, the algorithm checks for two types of crossovers: structure-based and opponent-based. When both types of patterns are available, a randomized decision is made to select the crossover type. If no swappable patterns are present then the crossover operation is skipped for that pair of parents, and they are passed unchanged to the next generation.

3.2.4.1 Opponent-based Crossover

For the opponent-based crossover, the algorithm looks for regular patterns in the home / away games of both parents that meet the submatrix length and width. Once patterns that are repeating or alternating sequences of home and away games are found for a given length and width, teams corresponding to these identified sequences swap their assigned opponents. Importantly, this exchange respects the original home or away designation, meaning that teams continue to play at the same locations as previously scheduled i.e. if team A was playing at home against team B, then after the crossover it will still play at home against that new opponent. This allows for a targeted recombination of opponent matchups without disrupting the broader home/away scheduling structure.

This targeted recombination strategy enhances genetic diversity within the population as it explores different permutations within the given structure of the population allowing for the program to measure the change in fitness across many configurations which guides the algorithm towards improved solutions.

3.2.4.2 Structure-based Crossover

However, this is not enough to explore the search space as just by swapping the opponents only a given set of home / away structures, that were generated during initialization, will be present in the set of solutions. To solve this, there is also another type of crossover which swaps the home / away designations to allow more diversity to the schedules. This structure-based crossover

identifies complementary patterns between two parents, for that submatrix length and width, where one team's home games correspond to the other's away games for example if team A has the schedule: home, away, home, and team B has the schedule: away, home, away. In such cases, these segments of the schedule are swapped.

By doing this, more variety is introduced into the solutions as this crossover causes newer structural patterns to emerge that may not have been present during population initialization. This operator enhances the algorithm’s ability to explore structural aspects of the schedule and provides an essential mechanism for escaping local optima in the solution space by adding structural diversity within the population.

Figure 4 provides a flowchart that gives an overview of the whole crossover algorithm.

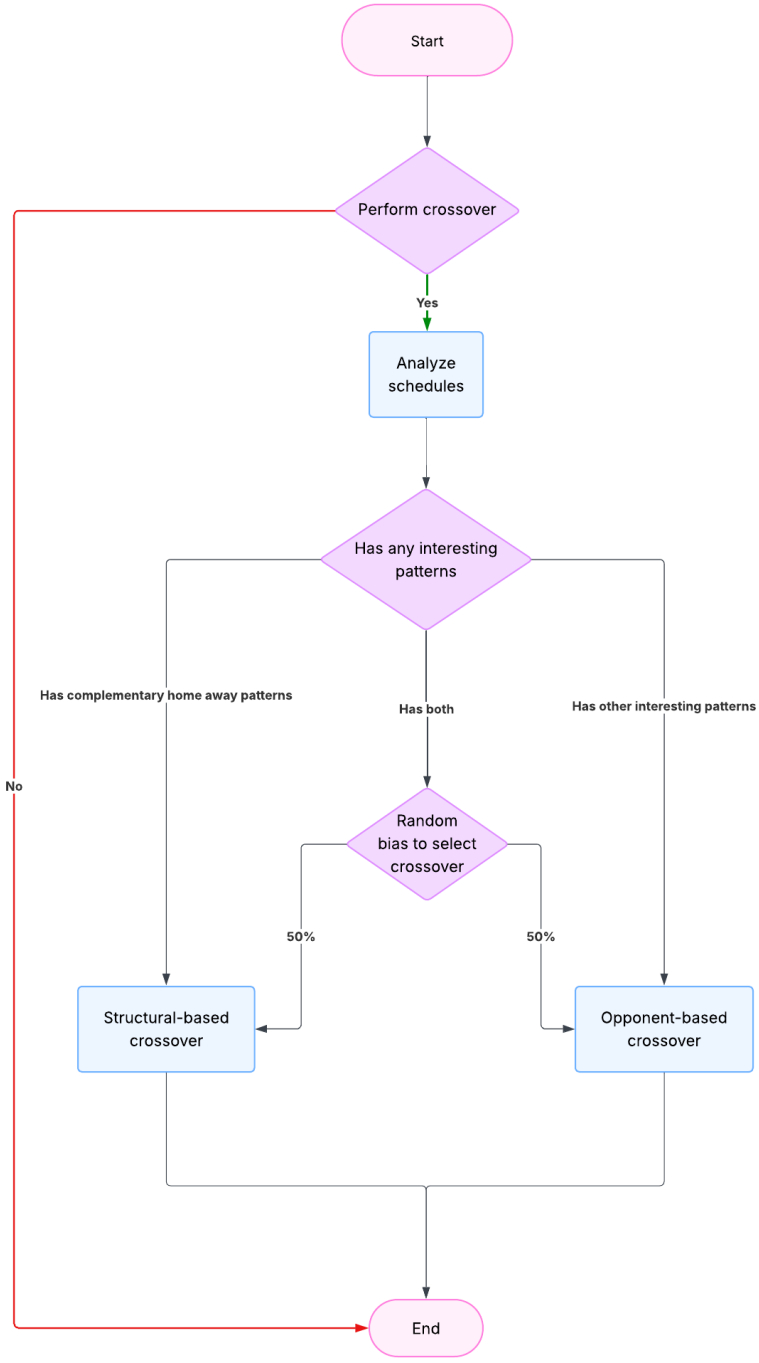

Figure 4: Flowchart of the algorithm

3.2.5 Mutation

This algorithm makes use of swap mutation; it is performed by swapping opponents between two teams in the same round. This mutation helps explore new areas of the solution space without completely disrupting the underlying structure of the schedule.

The algorithm selects one of the team's games at random, identifies another team playing in the same round, and attempts to swap their opponents. The swap is only accepted if it does not result in an invalid configuration, a team being scheduled to play against itself. After swapping, all corresponding mirrored games (i.e., the matches recorded from the opponent's perspective) are updated to maintain schedule consistency, and travel distances are recalculated.

3.2.6 Fitness Function

The fitness function in this genetic algorithm is designed using a case-based evaluation strategy. The approach distinguishes between feasible and infeasible schedules, applying different evaluation criteria depending on the feasibility status of the solution. A fixed threshold value is used to separate the evaluation scales. The threshold value is 100 for this algorithm. The fitness function first checks the feasibility for the schedule before assigning the fitness value to that given schedule. Infeasible schedules receive a fitness value below the threshold; their fitness score reflects the proportion of hard constraints satisfied relative to the maximum possible number of constraints. Specifically, the number of violations is calculated, and the fitness score is scaled proportionally: the fewer the violations, the closer the fitness score is to 100. Feasible schedules are evaluated primarily based on their total travel distance. For feasible solutions, the fitness score is greater than the threshold, encouraging the algorithm to minimize travel distances once feasibility has been achieved. Fitness is calculated by subtracting the total travel distance from the maximum distance that can be travelled and adding this value to the threshold. The maximum distance that can be travelled is an approximate upper bound calculated by doubling the sum of all the distances between each team; each team travels to their away game and returns home before another round of games. For this, we assume no chaining of consecutive away games i.e., the worst schedule possible. This is simple but effective for the purpose of this algorithm as we just require a simple guaranteed upper limit from which the actual distance travelled will be subtracted leading to the fitness being larger for schedules where the actual total travel distance will be lower.

Let:

- s be a candidate tournament schedule (an individual).
- th be the threshold; the fixed cutoff separating infeasible and feasible evaluations.
- $V(s)$ be the number of constraint violations in schedule s .
- V_{max} be the theoretical maximum number of possible violations.
- D_{max} be the maximum travel distance that can be travelled by the schedule.
- $D(s)$ be the total travel distance of schedule s .

Then let the fitness function $f(s)$ be:

$$f(s) = \begin{cases} \dfrac{V_{max} - V_s}{V_{max}} \times th, & \text{if the schedule is infeasible} \\ th + (D_{max} - D_s), & \text{if the schedule is feasible} \end{cases}$$

The implementation of this fitness function is handled by the 'fitness.py' file, which also heavily relies on 'checkfeasibility.py', which is responsible examining the schedules and counting the number of violations for every specific violation as well as declaring if a schedule is feasible of not. 'fitness.py' makes use of this information to return fitness values based on the equations in the fitness function *f(s)*.

4. Evaluation

For the evaluation of the genetic algorithm, standard and widely recognized benchmark datasets, the "NL" series (NL4 to NL10) and the "CIRC" series (CIRC4 to CIRC10), have been used. The NL-instances [1] are based on air distance between the city centers from teams in the National League of the Major League Baseball. The circular distance instances (CIRC) [1] is also another set of commonly used data sets to benchmark TTP. In this, teams' venues are placed on a circle and any two consecutive teams are connected by an edge and the distance between two teams is equal to the minimal number of edges that must be traversed to get to the other team. Although for traveling salesman problems with a circular distance matrix have a trivial solution, it remains challenging to solve circular traveling tournament instances. These datasets provide the benchmark for the best results that have been achieved in literature.

The metrics that have been used to evaluate these data sets are chiefly the feasibility and the total distance travelled. These are reflected in the benchmarks for these data sets (See Table 2). Other metrics that are measured are the runtime and fitness of the schedule.

All the evaluation code is in the ‘experiments.py’ file. The file has been submitted alongside this report. It contains functions for all the testing that has been talked about below.

4.1 Overall Results

Here the overall results are discussed and compared with the benchmarks. In the brackets ‘(a, b)’, ‘a’ refers to the number of constraint violations and ‘b’ refers to the total distance travelled. The table represents the best results found for each instance after all the experimentation. In this report, when a schedule referred to as optimal that means that schedule meets the benchmark result.

The configurations for all instances were the same except for the population sizes as for larger instances NL8 and CIRC8, a larger population size was required to yield positive results whereas for the instances containing 4 and 6 teams population size was not much a factor in getting feasible results. For all 4 instances, NL4, CIRC4, NL6 and CIRC6, all population sizes above 100 yielded the most positive results (For a more detailed discussion see Section 4.3.1).

Table 2: Best results found for each instance

Instance	Benchmark	Results
NL4	(0, 8276)	(0, 8276)
NL6	(0, 23916)	(0, 27484)
NL8	(0, 39721)	(0, 53028)
NL10	(0, 59436)	(8, 85981)
CIRC4	(0, 20)	(0, 20)
CIRC6	(0, 64)	(0, 76)

CIRC8	(0, 132)	(0, 204)
CIRC10	(0, 242)	(9, 375)

As we can see the genetic algorithm is able to find optimal feasible results that match the benchmarks for the NL4 and CIRC4 instances. The algorithm finds feasible but suboptimal results for NL6 and CIRC6, 27484 compared with the best-known result 23916, a 14.9% difference, and 76 compared with 64, an 18.75% difference.

The algorithm is able to find feasible results for the instances containing 8 teams however the results are far from the benchmarks as can be seen in the table. Also, it must be noted that a feasible solution is not always found and whether a feasible result is found or not depends upon the configurations with a high population size (500- 1000) with a normal number of generations (50-75) yielding the best results for these instances. (See Section 4.3.1)

For CIRC10 and NL10 the algorithm is unable to find a feasible solution. In an attempt to find any feasible solutions for NL10 and CIRC10, an experiment was run with population sizes above 1000, however it yielded no positive results and took long to run.

The results align with what was found in [5], where instances with 4 and 6 teams yielded results that match the best-known results (benchmarks) and instances with 8 teams yield feasible but suboptimal results. However, unlike [5], the results mentioned here, for instances with 6 teams, are feasible, however in terms of optimality they do not match the best-known results.

In terms of the runtime, the algorithm takes twice as long for every two teams added. For the configurations: population size – 100 and generations – 50, CIRC4 and NL4 take 2-4 seconds, CIRC6 and NL6 take 4-5 seconds, CIRC8 and NL8 take 8-10 seconds and CIRC10 and NL10 take 18-20 seconds. An example run of the experiment can be found in Appendices 3.

The schedules that got the optimal results for NL4 and CIRC4 can be found in Appendices 1 and 2.

4.2 Custom Crossover vs K-point crossover

Here the crossover, introduced above (Section 3.2.4), that was specifically made for this problem is compared and evaluated against a widely used general K-point crossover. The crossover is tested for the most used k values, 1 and 2. (See Section 2.4)

The configurations for the experiment were:

- Population size: 200
- Generations: 50
- Crossover rate: 0.8

- Mutation rate: 0.2
- Tournament Size: 3

Table 3: Results from the comparison experiment for K-point 1

Instance	Algorithm	Runtime (s)	Fitness	Feasible	Violations	Total Distance Travelled
NL4.xml	GA	2.41	4368.0	TRUE	0	8276
	K-pt	1.94	2343.0	TRUE	0	10301
NL6.xml	GA	5.46	9811.0	TRUE	0	29257
	K-pt	5.3	7657.0	TRUE	0	31411
NL8.xml	GA	11.32	98.44	FALSE	3	54183
	K-pt	10.43	83.85	FALSE	31	51244
NL10.xml	GA	29.35	98.12	FALSE	6	88191
	K-pt	29.63	84.06	FALSE	51	90365
CIRC4.xml	GA	2.39	112.0	TRUE	0	20
	K-pt	1.89	106.0	TRUE	0	26
CIRC6.xml	GA	5.83	124.0	TRUE	0	84
	K-pt	5.26	116.0	TRUE	0	92
CIRC8.xml	GA	10.92	98.96	FALSE	2	198
	K-pt	10.44	85.94	FALSE	27	194
CIRC10.xml	GA	29.69	98.44	FALSE	5	396
	K-pt	29.16	82.5	FALSE	56	416

When k = 1, as the results show (See the table above), the custom crossover performs better than the K-point crossover. For the solutions that are feasible, the difference in fitness and total difference travelled is sizeable and the difference in optimality between them can be seen. Where the solutions are infeasible, the difference between the number of violations for the custom crossover and k-point is very substantial e.g. for CIRC10 the custom crossover has only 5 violations whereas the K-pt has 56, approximately 10 times larger, this trend can be seen across the Table.

Next, the algorithm is tested with the same configurations just a change in k from 1 to 2.

Table 4: Results from the comparison experiment for K-point 2

Instance	Algorithm	Runtime (s)	Fitness	Feasible	Violations	Total Distance Travelled
NL4.xml	GA	2.31	4368.0	TRUE	0	8276
	K-pt	1.93	3710.0	TRUE	0	8934
NL6.xml	GA	5.51	11068.0	TRUE	0	28000
	K-pt	5.26	7127.0	TRUE	0	31941
NL8.xml	GA	10.88	99.48	FALSE	1	53765

	K-pt	10.38	82.81	FALSE	33	54176
NL10.xml	GA	29.42	97.19	FALSE	9	92823
	K-pt	28.46	87.5	FALSE	40	86282
CIRC4.xml	GA	2.27	112.0	TRUE	0	20
	K-pt	1.88	110.0	TRUE	0	22
CIRC6.xml	GA	5.69	126.0	TRUE	0	82
	K-pt	5.19	126.0	TRUE	0	82
CIRC8.xml	GA	11.02	98.44	FALSE	3	202
	K-pt	10.31	84.38	FALSE	30	218
CIRC10.xml	GA	29.7	98.44	FALSE	5	398
	K-pt	29.07	85.0	FALSE	48	392

From these findings, it can be seen that there is an instant improvement in performance of the K-point crossover when the number of k is 2 instead of 1. For feasible solutions, the total distance travelled is much closer to the value returned from the custom crossover. For infeasible solutions, the number of constraint violations is also lower but only slightly. Compare the values of the number of violations for NL8, NL10, CIRC8 and CIRC10 instances for k = 1: 31, 51, 27, 56, and k = 2: 33, 40, 30 and 48. For NL8 and CIRC8 instances the number of violations remains virtually the same however for NL10 and CIRC10 the number of violations sees a slight decrease.

Overall, it can be concluded that the custom crossover created in this project performs better than the k-point crossover for both k equals 1 and k equals 2. When k equals 2, the performance for feasible solutions improves vastly compared to when k equals 1, the total distance travelled comes closer to the custom crossover values. However, both suffer for infeasible solutions where the custom crossover performs much better than the k-point crossover; the number of violations is significantly higher for both values of k compared to the custom crossover.

4.3 Configuration Testing

4.3.1 Population Size

In this experiment, we test each instance against multiple population sizes ranging from 50 to 1000 (50, 100, 150, 200, 250, 500, 1000) and see which population sizes result in the best results for each instance (See Table 5). All other configurations were kept the same for all instances:

- Generations = 50
- Crossover rate = 0.8
- Mutation rate = 0.2
- Tournament Size = 3

For the NL4 and CIRC4, all population sizes gave feasible results. For NL4, the population sizes 50 and 100 yielded suboptimal results and the rest were optimal results i.e., they met the benchmark. For CIRC4, all results were optimal.

For NL6 and CIRC6, all population sizes also gave feasible results. The distance travelled varied between 27100-31000 for NL6 and 76-88 for CIRC6, the most optimal being 150 for both instances.

For NL8, only some population sizes gave feasible results, however it was not directly correlated with population size. Only population sizes above 200 gave feasible solutions but not always. CIRC8 was more directly correlated with population size only population sizes above 500 yielding feasible solutions. All feasible solutions were suboptimal.

For NL10 and CIRC10, no feasible solutions were yielded though number of violations did decrease as the population sizes increased.

On the whole, it can be said that as the number of teams increases, they require a higher population size to yield positive results. The CIRC datasets are more directly correlated with population size and NL datasets give a lot more varied results even where you'd expect them to yield all feasible / optimal solutions.

Table 5: Results from population size experiment

Instance	Population Size	Feasible	Distance Travelled
NL4	50	TRUE	8450.0
	100	TRUE	8313.0
	150	TRUE	8276.0
	200	TRUE	8276.0
	250	TRUE	8276.0
	500	TRUE	8276.0
	1000	TRUE	8276.0
CIRC4	50	TRUE	20.0
	100	TRUE	20.0
	150	TRUE	20.0
	200	TRUE	20.0
	250	TRUE	20.0
	500	TRUE	20.0
NL6	50	TRUE	31257.0
	100	TRUE	29213.0
	150	TRUE	27483.0
	200	TRUE	30075.0
	250	TRUE	29238.0

	500	TRUE	28731.0
	1000	TRUE	28110.0
CIRC6	50	TRUE	84.0
	100	TRUE	88.0
	150	TRUE	76.0
	200	TRUE	76.0
	250	TRUE	80.0
	500	TRUE	76.0
	1000	TRUE	80.0
CIRC8	50	FALSE	
	100	FALSE	
	150	FALSE	
	200	FALSE	
	250	FALSE	
	500	TRUE	208.0
	1000	TRUE	204.0
NL8	50	FALSE	
	100	FALSE	
	150	FALSE	
	200	TRUE	55087.0
	250	TRUE	53028.0
	500	FALSE	
	1000	TRUE	56885.0
NL10	50	FALSE	
	100	FALSE	
	150	FALSE	
	200	FALSE	
	250	FALSE	
	500	FALSE	
	1000	FALSE	
CIRC10	50	FALSE	
	100	FALSE	
	150	FALSE	
	200	FALSE	
	250	FALSE	
	500	FALSE	
	1000	FALSE	

*Although this experiment was ran several times to verify results, Table 5 is the result of a single full run of the experiment.

4.3.2 Other

Other experiments that took place were to do with manipulating other parts of the configurations such as crossover rate and mutation rate However, a change in the values of all these alone, when all other parts of the configurations remained the same, didn't make a substantial difference in the results unless there was a significant change in these values from what is considered normal.

4.3.2.1 Tournament Size

The tournament size makes a difference in exploring different solution spaces and because of this a tournament size that is too large limits the exploration and too low limits passing on the best genes. This can be noticeable in large TTP instances where it makes a difference in the fitness and feasibility of the schedules. However, the noticeability of this can depend on other factors like population size, the larger the population size the more difference the value of the tournament size makes, and number of generations, if there are more number of generations then the tournament size needs to be lower as large tournament sizes will lead to premature convergence before the end of all generations. When tournament size alone was measured, for the smaller instances for the TTP, the best tournament size was 3 and for the larger instances (8 and 10), tournament size 4 gave the best results.

5. Conclusion and Future Work

The main aims of this project were to gain a strong understanding of the literature surrounding the Travelling Tournament Problem, to implement a genetic algorithm solution and to evaluate its performance against established benchmarks. While genetic algorithms have previously been applied to the TTP, their diverse nature allows for continued exploration. In this project, the algorithm was tailored to the problem by customising genetic algorithms operators such as the genome representation, the fitness function, and the crossover. The implemented algorithm performed effectively on small instances, consistently finding feasible solutions for problems with 4 and 6 teams. For 4-team instances, the algorithm matched the best-known benchmark results and for 6-team instances, it produced suboptimal solutions that were reasonably close to the benchmarks. Feasible solutions were also found for 8-team instances, though they were further from the optimal. For larger instances, more than 8 teams, the algorithm was unable to find feasible solutions. Overall, the project successfully met its initial objectives by reviewing relevant literature, implementing an original genetic algorithm solution for the TTP, and evaluating its performance on benchmark instances.

While this project achieved its primary objectives, several opportunities remain for extending and improving the current approach in future work. From the results acquired in this project and in [5], it is evident that a standalone genetic algorithm struggles to match the benchmark results achieved by some other optimization techniques. However, there is substantial potential in exploring hybrid approaches, such as those discussed in [10], which have demonstrated greater potential. Hybridizations between genetic algorithms and other heuristic approaches, such as simulated annealing, tabu search, or ant colony optimization, are worth exploring, as they can combine the global search capabilities of evolutionary algorithms with the intensification strengths of local search methods. Through these hybrid approaches, future work can aim to develop more effective solutions to the Travelling Tournament Problem.

6. Reflection

Through conducting this project, I have expanded my theoretical knowledge in the area of combinatorial optimization. I gained hands-on experience with designing algorithms, evaluating their performance, and understanding the computational challenges that arise in optimization problems. I learnt to read theoretical research papers, this enhanced my ability to critically analyse complex academic material, extract relevant insights, and apply them to practical scenarios. Reading about the TTP deepened my appreciation for the theoretical and real-world significance of combinatorial optimization problems.

In addition to technical knowledge, I developed valuable academic skills such as working independently on a computer science problem, academic reading, and formal academic writing. I also improved my ability to compare and interpret results within the context of existing literature, helping me to better situate my work within the wider research landscape.

Alongside these technical and academic competencies, I cultivated important personal and professional skills. Managing the project over several months required effective time management, prioritisation, and adaptability, especially when unexpected issues such as bugs in the code arose. Learning to balance deadlines and workloads also helped me improve my stress management and resilience. Although the initial project plan provided a clear starting point and direction, the development process involved more iteration than originally anticipated. Frequent revisions to the algorithm, ongoing testing and refinements to the code led to a more fluid workflow, resembling an agile development approach rather than a strict linear progression such as in the waterfall workflow. This iterative method ultimately contributed to a better final solution and taught me the value of flexibility and continuous feedback in research and development.

Overall, I think that this project was a huge learning experience for me, and tackling such a big project on my own has taught me a lot.

Appendices

1.

```
yraikhy@Mac ga % python3 experiments.py
Best Schedule Found:
Is feasible? True
Best fitness: 4368
Total distance travelled: 8276

Schedule for Team 0 (ATL):
Round 1: Away at MON (Distance: 929)
Round 2: Away at NYM (Distance: 745)
Round 3: Away at PHI (Distance: 665)
Round 4: Home vs NYM
Round 5: Home vs MON
Round 6: Home vs PHI
Total distance traveled: 2011

Schedule for Team 1 (NYM):
Round 1: Home vs PHI
Round 2: Home vs ATL
Round 3: Away at MON (Distance: 337)
Round 4: Away at ATL (Distance: 745)
Round 5: Away at PHI (Distance: 80)
Round 6: Home vs MON
Total distance traveled: 2011

Schedule for Team 2 (PHI):
Round 1: Away at NYM (Distance: 80)
Round 2: Away at MON (Distance: 380)
Round 3: Home vs ATL
Round 4: Home vs MON
Round 5: Home vs NYM
Round 6: Away at ATL (Distance: 665)
Total distance traveled: 2127

Schedule for Team 3 (MON):
Round 1: Home vs ATL
Round 2: Home vs PHI
Round 3: Home vs NYM
Round 4: Away at PHI (Distance: 380)
Round 5: Away at ATL (Distance: 929)
Round 6: Away at NYM (Distance: 337)
Total distance traveled: 2127
```

```
yraikhy@Mac ga % python3 experiments.py
Best Schedule Found:
Is feasible? True
Best fitness: 112
Total distance travelled: 20

Schedule for Team 0 (T1):
Round 1: Home vs T2
Round 2: Away at T3 (Distance: 2)
Round 3: Away at T4 (Distance: 1)
Round 4: Away at T2 (Distance: 1)
Round 5: Home vs T3
Round 6: Home vs T4
Total distance traveled: 6

Schedule for Team 1 (T2):
Round 1: Away at T1 (Distance: 1)
Round 2: Away at T4 (Distance: 2)
Round 3: Away at T3 (Distance: 1)
Round 4: Home vs T1
Round 5: Home vs T4
Round 6: Home vs T3
Total distance traveled: 4

Schedule for Team 2 (T3):
Round 1: Home vs T4
Round 2: Home vs T1
Round 3: Home vs T2
Round 4: Away at T4 (Distance: 1)
Round 5: Away at T1 (Distance: 2)
Round 6: Away at T2 (Distance: 1)
Total distance traveled: 4

Schedule for Team 3 (T4):
Round 1: Away at T3 (Distance: 1)
Round 2: Home vs T2
Round 3: Home vs T1
Round 4: Home vs T3
Round 5: Away at T2 (Distance: 2)
Round 6: Away at T1 (Distance: 1)
Total distance traveled: 6
```

2.

3.

```
================================================================
                    SUMMARY OF RESULTS
================================================================
Instance   | Teams    | Runtime (s)  | Fitness    | Feasible | Number of Violations | Total Distance Travelled
----------------------------------------------------------------
NL4.xml    | 4        | 1.94         | 4331.00    | 1        | 0                    | 8313.00
NL6.xml    | 6        | 4.47         | 8471.00    | 1        | 0                    | 30597.00
NL8.xml    | 8        | 8.31         | 97.40      | 0        | 5                    | 56241.00
NL10.xml   | 10       | 18.45        | 97.81      | 0        | 7                    | 86869.00
CIRC4.xml  | 4        | 1.93         | 112.00     | 1        | 0                    | 20.00
CIRC6.xml  | 6        | 4.49         | 126.00     | 1        | 0                    | 82.00
CIRC8.xml  | 8        | 8.35         | 98.44      | 0        | 3                    | 212.00
CIRC10.xml | 10       | 20.09        | 96.88      | 0        | 10                   | 410.00
```

References

[1] K. Easton, G. Nemhauser, and M. Trick, “The traveling tournament problem description and benchmarks,” in *Proc. 7th Int. Conf. Principles and Practice of Constraint Programming (CP 2001)*, Paphos, Cyprus, 2001, pp. 580–584. doi: 10.1007/3-540-45578-7_43.

[2] K. Easton, G. Nemhauser, and M. Trick, “Solving the travelling tournament problem: A combined integer programming and constraint programming approach,” in *Proc. 4th Int. Conf. Practice and Theory of Automated Timetabling (PATAT 2002)*, Gent, Belgium, 2003, pp. 100–109. doi: 10.1007/978-3-540-45157-0_6.

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

[4] A. Haldar, S. Mondal, A. Mukherjee, and K. Chatterjee, “A comparative analysis of application of genetic algorithm and particle swarm optimization in solving traveling tournament problem (TTP),” *Int. J. Bioinfor. Intell. Comput.*, vol. 1, no. 2, pp. 1–15, 2022. doi: 10.61797/ijbic.v1i2.168.

[5] N. Choubey, “A novel encoding scheme for traveling tournament problem using genetic algorithm,” *Int. J. Comput. Appl.*, vol. ECOT, no. 2, pp. 79–82, 2010. doi: 10.5120/1536-139.

[6] K. Verduin, S. L. Thomson, and D. van den Berg, "Too Constrained for Genetic Algorithms too Hard for Evolutionary Computing the Traveling Tournament Problem," Dept. of Computer Science, Vrije Universiteit Amsterdam and Universiteit van Amsterdam, Amsterdam, The Netherlands, 2023.

[7] A. Lim, B. Rodrigues, and X. Zhang, “A simulated annealing and hill-climbing algorithm for the traveling tournament problem,” *European Journal of Operational Research*, vol. 174, no. 3, pp. 1459–1478, 2006. doi: 10.1016/j.ejor.2005.02.048.

[8] Nourollahi, S., Eshghi, K., & Razaghi, H. (2012). An efficient simulated annealing approach to the travelling tournament problem. American Journal of Operations Research, 02(03), 391-398. https://doi.org/10.4236/ajor.2012.23047

[9] J. H. Holland, *Adaptation in Natural and Artificial Systems*. Ann Arbor, MI: University of Michigan Press, 1975.

[10] M. Khelifa, D. Boughaci and E. Aïmeur, "An enhanced genetic algorithm with a new crossover operator for the traveling tournament problem," 2017 4th International Conference on Control, Decision and Information Technologies (CoDIT), Barcelona, Spain, 2017, pp. 1072-1077, doi: 10.1109/CoDIT.2017.8102741.