NOVA IMS Information Management School

2024/2025

SPORTS LEAGUE OPTIMIZATION

Computational Intelligence for Optimization

GROUP S

Gustavo Veloso, 20240599 | Leonor Mira, 20240658
Martim Tavares, 20240508 | Santiago Taylor, 20240542

GitHub Link: https://github.com/leonormira/CIFO-Project

Table of Contents

1. Introduction

Genetic algorithms (GAs) are population-based metaheuristic techniques inspired by the principles of natural selection and genetics [1]. They are particularly effective for addressing complex combinatorial optimization problems where exhaustive methods are computationally impractical [2].
In this project, a Genetic Algorithm has been applied to the challenge of forming balanced sports leagues by minimizing the variance in average skill levels among teams. A grid search was performed to evaluate various combinations of genetic operators and hyperparameters. Despite the dataset's limited size and diversity, the algorithm consistently yielded valid and near-optimal solutions, showcasing its robustness and adaptability.

2. Problem Definition

We selected the problem of Sports League Optimization, which aims to create 5 teams, each consisting of seven players. The objectives are to ensure that the average skill level is balanced across all teams while keeping the budget for each team under €750 million. Additionally, each team must adhere to a specific formation: one goalkeeper (GK), two defenders (DEF), two midfielders (MID), and two forwards (FWD).

2.1. Problem Approach

To address the Sports League Optimization problem, we commenced by defining the population size, which dictates the number of distinct league configurations (individuals) present in each generation of the algorithm. The foundation of our solution rests on three primary classes: Player, Team, and League.

The Player class encapsulates the attributes of each player, including their name, position, skill rating, and salary. It also features comparison methods to ensure that no player is selected more than once across the teams.

The Team class is responsible for managing players based on their positions. Players are organized into separate lists according to their roles: Goalkeeper (GK), Defender (DEF), Midfielder (MID), and Forward (FWD). Additionally, this class computes essential metrics such as the team's average skill and budget expenditure, thereby enforcing the constraint that no team exceeds the €750 million budget.

The League class represents a complete solution with five valid teams. It controls how players are spread across the teams, ensuring that no player is duplicated while confirming each team has the correct composition of players. To accomplish this, we first categorize the full player pool into four distinct lists by position (GK, DEF, MID, FWD), and then methodically assign players to teams in order. This approach guarantees that each team is allotted exactly one goalkeeper, two defenders, two midfielders, and two forwards. For a visual representation, consult Figure 2.

Moreover, the League class incorporates methods for mutation and crossover, which are vital for the evolutionary process in the genetic algorithm. It also handles the evaluation of fitness, allowing us to assess how well-balanced and valid a league configuration is.

2.2. Search Space

In our implementation, each League object represents a unique point within the search space. The entire population, determined by the population size, is composed of multiple such leagues. Each league contains five distinct teams, which together create a valid configuration of all 35 players. By applying genetic operators such as crossover and mutation, these configurations evolve over time, exploring the search space in pursuit of a globally optimal solution where teams are both balanced in skill and adhere to budget constraints.

2.3. Fitness Function

To assess the quality of the solution, we computed the fitness by evaluating the standard deviation of the average skill levels across the five teams. Our objective is to establish a balanced league, which means we strive for the standard deviation to approach zero as closely as possible. This transforms our problem into one of minimization, where a lower standard deviation signifies reduced variation in team skill levels.

3. Selection Mechanism

During the selection phase of the genetic algorithm, we evaluated two strategies: tournament selection and fitness proportionate selection. Selection plays a crucial role as it determines which individuals will reproduce based on their fitness, ultimately guiding the algorithm toward better solutions over time.
In fitness proportionate selection (roulette selection), individuals are chosen according to their probability of selection and a randomly generated number. This method is particularly effective for minimization problems because individuals with lower fitness scores, indicating better solutions, are more likely to be selected. We implement this through an inverse fitness approach, where the selection probability is calculated as 1/fitness, as we can see in Figure 3.
In contrast, tournament selection identifies the best individual from a randomly selected subset of the population (Figure 4). This technique exerts stronger selection pressure and is less sensitive to variations in fitness.
Both methods were included in the grid search to evaluate their impact on the algorithm's performance. This analysis enabled us to identify which approach more effectively supports convergence and enhances solution quality for the specific problem at hand.

4. Mutation and Crossover Mechanisms

Mutations are an essential component of genetic algorithms, as they introduce diversity into the population and generate new, similar solutions. After selecting parent solutions and executing crossover, mutations can randomly modify small segments of each solution. This process allows the algorithm to explore a wider range of outcomes and helps prevent solutions from converging too quickly.

In our project, mutations are applied after parent selection but before the offspring are added to the new population. This method is easy to configure using our team structure, which consists of an array for each type of player.

We investigated three types of mutations. The first, shift mutation, involves randomly selecting a subset of players and shifting the entire subset one position to the right (**Figure 5**). The second, inversion mutation, entails randomly choosing a subset of players and reversing their order in place (**Figure 6**). Lastly, swap mutation involves selecting two random positions and exchanging the players in those positions (**Figure 7**).

Crossovers combine two parent solutions to create a new offspring that inherits traits from them, making it responsible for mixing team compositions for each generation. This is how the algorithm explores new combinations, always evaluating the best ones and keeping them on the next population while maintaining valid sets of players.

We used two crossovers. The first, swap crossover (**Figure 7**), starts by selecting a random position, then we locate where the player in that array position is on parent 1, then we swap those two players in the offspring. We then repeat it two generate the other solution using the other parent as reference. The second one, cycle crossover, starts on a random index and creates a cycle of positions by getting the values from parent 2 to their positions in parent 1, swapping the players on the offsprings, once the cycle is over, the other positions are inherited without changes.

5. Performance Analysis

5.1. Elitism Impact

Before undertaking a comprehensive grid search of the genetic algorithm parameters, an initial experiment was conducted to evaluate the impact of retaining the best individuals across generations. In one configuration, the genetic algorithm was allowed to evolve without restrictions, subjecting all individuals to crossover and mutation. Conversely, in the other configuration, the highest-performing individual from each generation was preserved and carried forward to the next generation, effectively safeguarding high-quality solutions from being lost in the randomness of genetic operations.

This straightforward modification yielded significant benefits. As illustrated in the accompanying graph bellow, the run that did not implement this preservation strategy exhibited considerably lower performance and an erratic fitness curve, likely due to the loss of strong individuals resulting from mutation or poorly executed crossovers. In contrast, the approach of maintaining the top individual facilitated a more stable and consistent improvement in fitness over time.

Given the clear advantages observed in terms of stability and solution quality, the elitism was integrated into all subsequent experiments. Consequently, it was not treated as a variable in the grid search but rather established as a fundamental aspect of the algorithm's design.

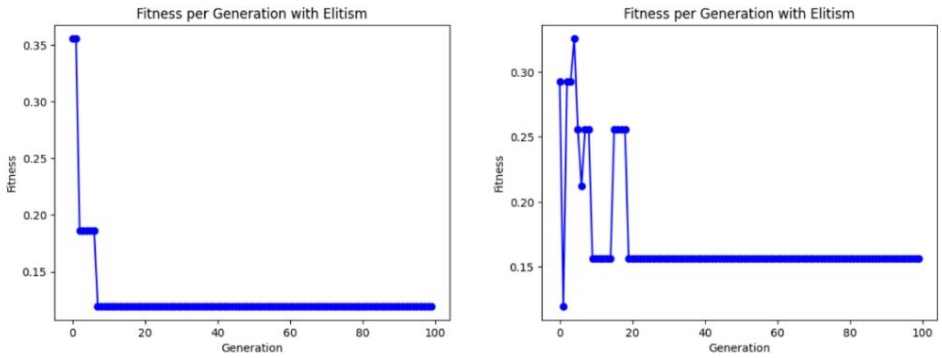

Figure 1- Impact of Elitism

5.2. Parameter Grid Search

To assess the impact of various crossover, mutation, and selection strategies, a thorough grid search was conducted. This search examined multiple combinations of genetic operators in conjunction with key parameters, including crossover probability, mutation probability, and tournament size. A complete list of the parameters tested can be found in Table 1, Table 2, and Table 3 in the appendix. In total, 72 distinct parameter configurations were evaluated.

To optimize overall computation time, the grid search was parallelized across four machines. Results from each machine were subsequently consolidated into a single dataframe for analysis. Each configuration was executed 30 times using a population size of 50 individuals over 100 generations (**Figure 9**). The mean performance across these 30 runs served as the representative metric for each configuration, helping to alleviate the effects of stochastic variability in the genetic algorithm.

Despite all seeming so similar, it is possible that different methods could have different impacts on the performance of the genetic algorithm. The Figure 10 in the appendix illustrate that, where each line represents the median fitness of the genetic algorithm given a certain crossover, mutation and selection function.

Upon analyzing the graph and comparing the various crossover functions, we can conclude that they do not significantly affect the algorithm's average performance. This finding is intriguing, as one might expect lower crossover probabilities to result in slower convergence. This phenomenon could be attributed to the random effects of the initial population or other influencing factors.

The mutation function appears to have the most substantial impact on fitness. While it facilitates convergence, the specific values at which this occurs vary. Notably, the inversion mutation with a mutation probability of 0.1 seems to perform the worst. However, given the scale of the values, the difference may not be statistically significant (a discussion of this is provided later). Similarly to crossover, the selection function also seems to have minimal impact on fitness. Tournament selection with k=3 achieved the lowest fitness level, likely due to its inability to effectively identify the best individuals for crossover.

Currently, there is limited evidence indicating that different configurations significantly impact the performance of genetic algorithms. To illustrate this visually, we have plotted the standard deviations of nine configurations, as shown in Figure 11. As expected, all lines fall within one standard deviation of each other, suggesting that the performance across all configurations is quite similar.

5.3. Statistics Test

To determine whether the differences in performance between configurations were statistically significant, the Wilcoxon signed-rank test was utilized. This non-parametric statistical test is commonly used to compare two related samples, particularly when the data does not follow a normal distribution. It assesses whether the distributions of the two samples differ in a statistically meaningful way.

The test was conducted for every possible pair of configurations, resulting in a total of 2,556 comparisons. For each pair, the p-value obtained from the Wilcoxon test was used to evaluate significance, with a threshold of 0.05. If the p-value for a given pair was below this threshold, the difference in performance was deemed statistically significant.

However, after performing all pairwise comparisons, none of the pairs of configurations showed a statistically significant difference in performance. This indicates that, despite variations in parameters and operator combinations, the configurations performed similarly within the considered statistical confidence level.

6. Final Genetic Algorithm

After evaluating all configurations and observing that no single setup demonstrated statistically superior performance, it was concluded that any of the tested configurations could theoretically be used to obtain a solution without significantly impacting the outcome. Based on this, a final configuration was selected to generate the optimal solution, with the goal of balancing simplicity and performance stability, consisting of a swap crossover function, swap mutation as the mutation method, a mutation probability of 0.4, an initial population size of 50, a total of 100 generations, and elitism enabled.

The resulting fitness curve is presented below, showing that the algorithm converged to a final fitness value of 0.12 standard deviations (Figure 9 and Figure 11). This low value indicates minimal variation in the average skill levels among the five teams, demonstrating that the generated league is well-balanced. The best solution identified by the algorithm can be found in Table 4 in the appendix.

7. Conclusion

In conclusion, the statistical analysis indicated that no configuration significantly outperformed or underperformed the others. While this outcome may be surprising, it can be attributed to the characteristics of the dataset, particularly the small number of players and the limited variation in their individual skill levels. Despite these constraints, the genetic algorithm developed in this project has proven to be both robust and adaptable. It is designed to scale effectively with larger and more diverse datasets that include additional players and teams. Therefore, it is reasonable to anticipate that differences in configuration performance may become more pronounced in more complex scenarios. However, in its current application, the algorithm consistently produced valid and near-optimal solutions, achieving the study's objectives and showcasing its potential for broader use.

References

[1] J. H. Holland, *Adaptation in Natural and Artificial Systems*, University of Michigan Press, 1975.

[2] D. E. Goldberg, *Genetic Algorithms in Search, Optimization, and Machine Learning*, Addison-Wesley, 1989.

Appendix

Figures

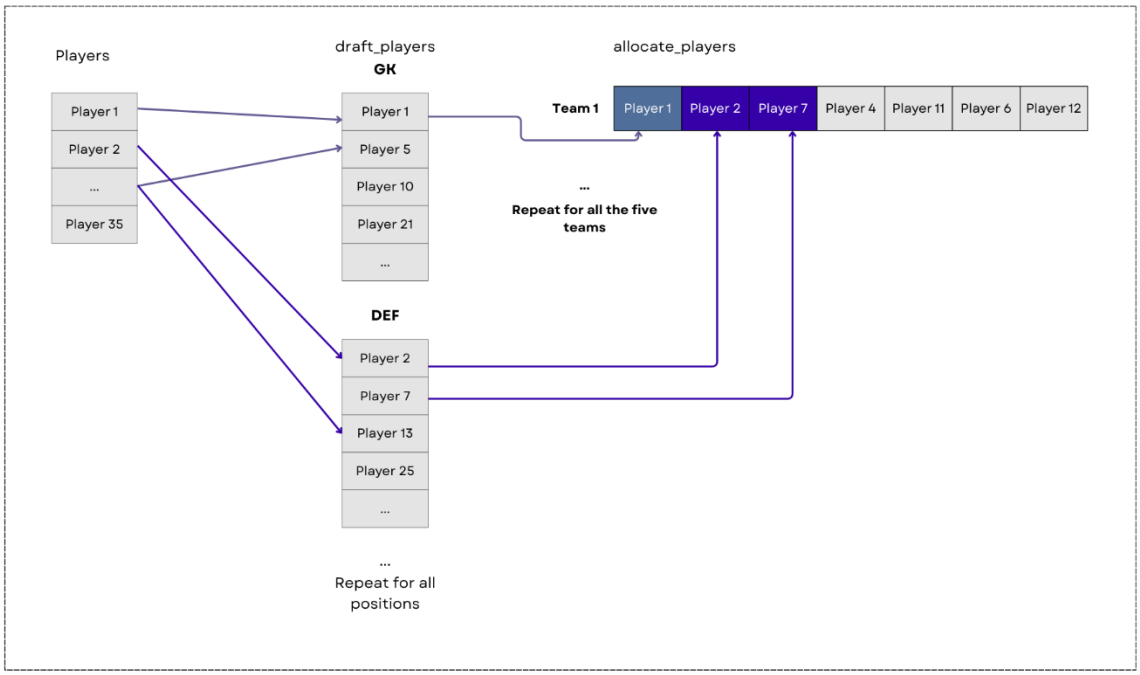

Figure 2- League Definition

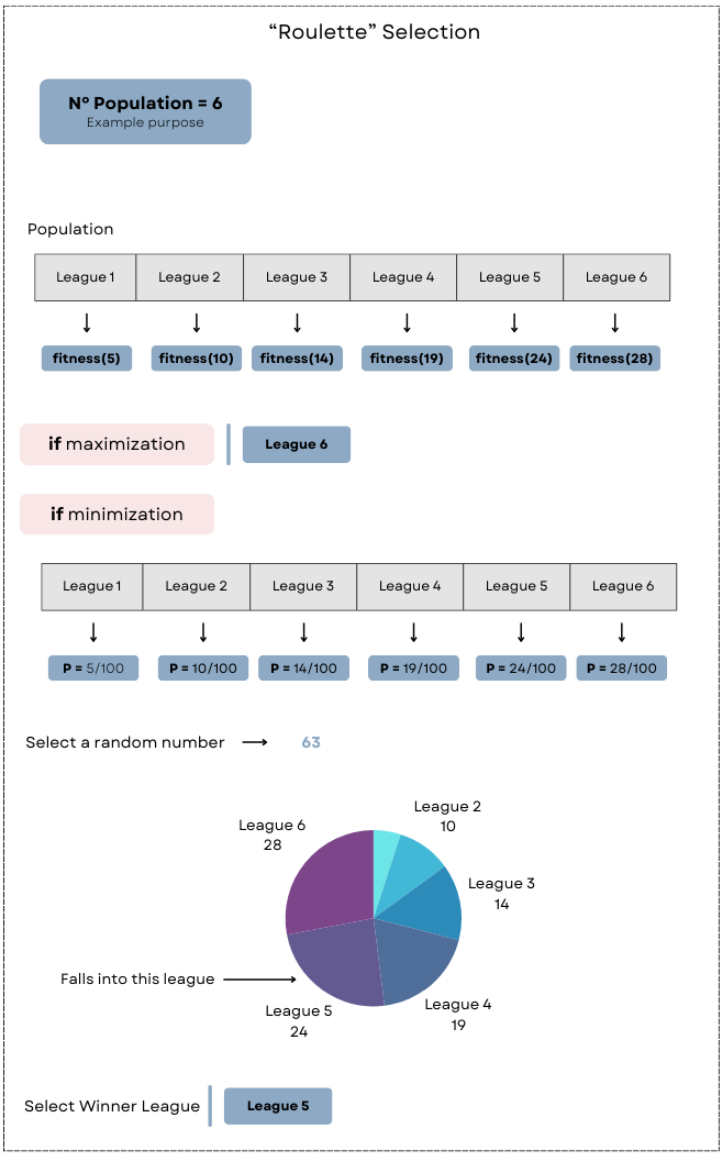

Figure 3- Roulette Selection

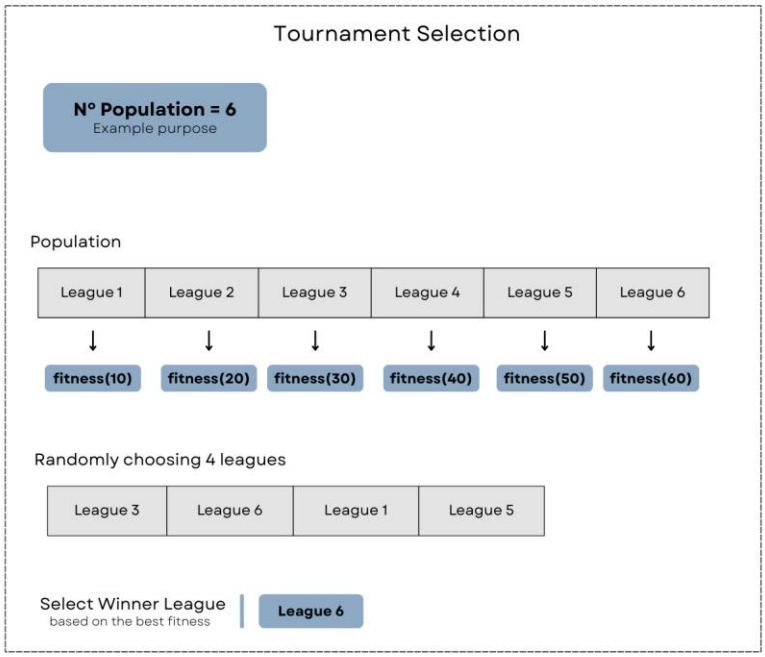

Figure 4- Tournament Selection

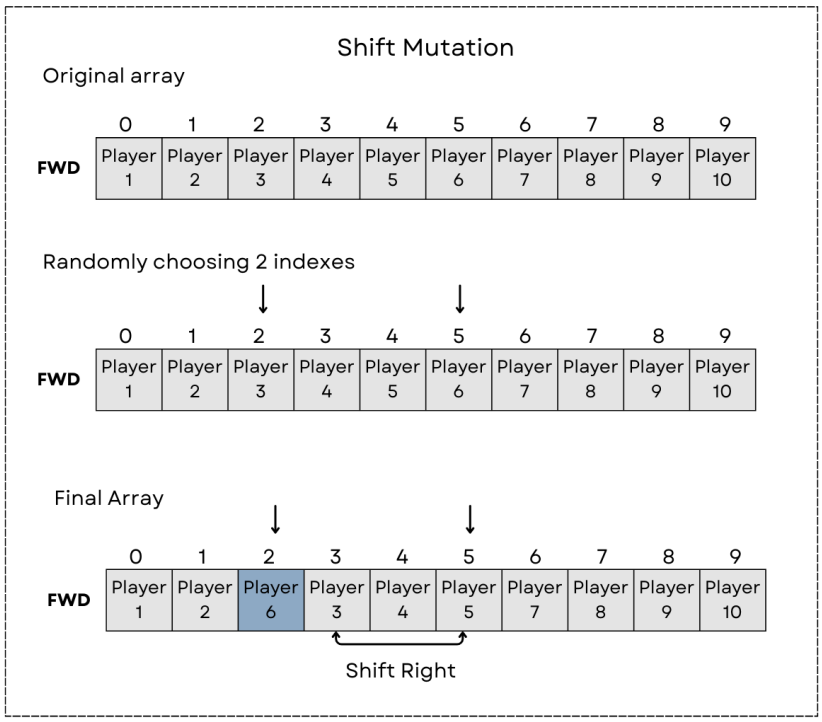

Figure 5- Shift Mutation

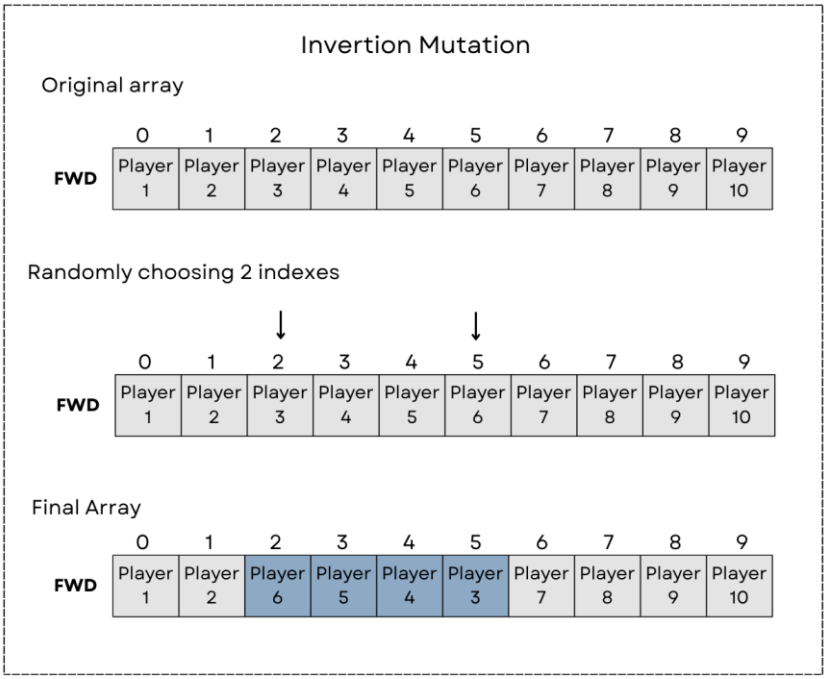

Figure 6- Inversion Mutation

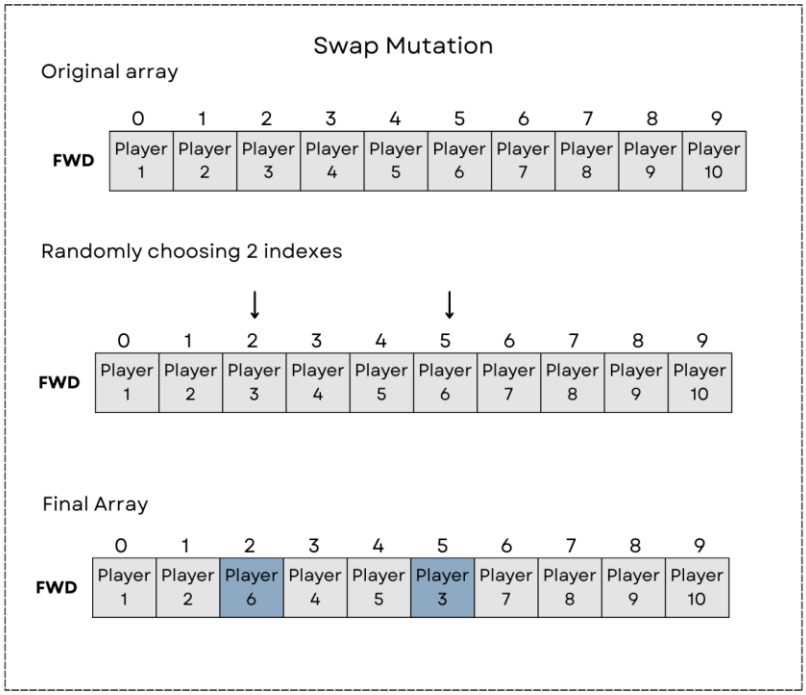

Figure 7- Swap Mutation

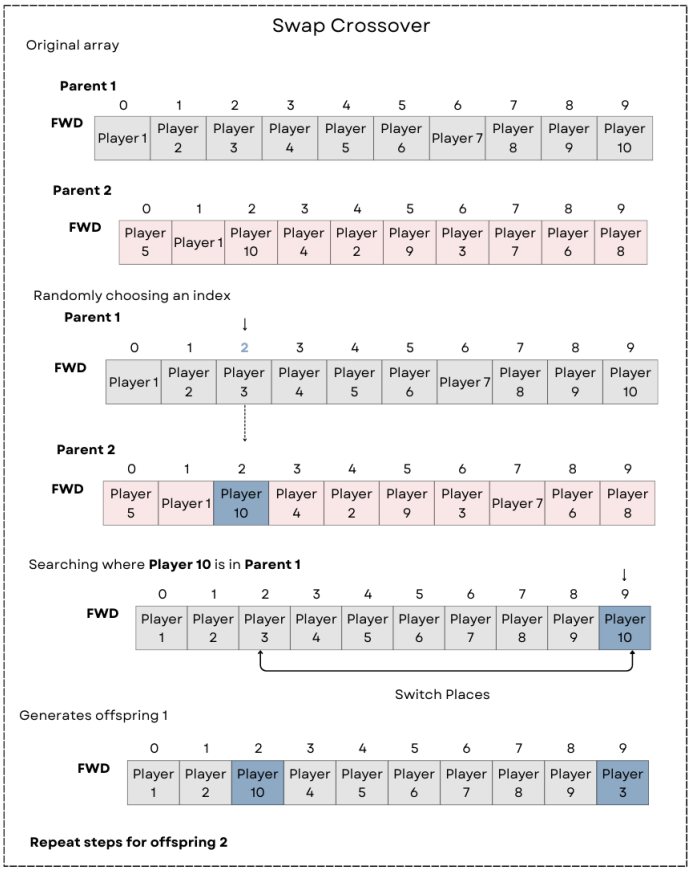

Figure 8- Swap Crossover

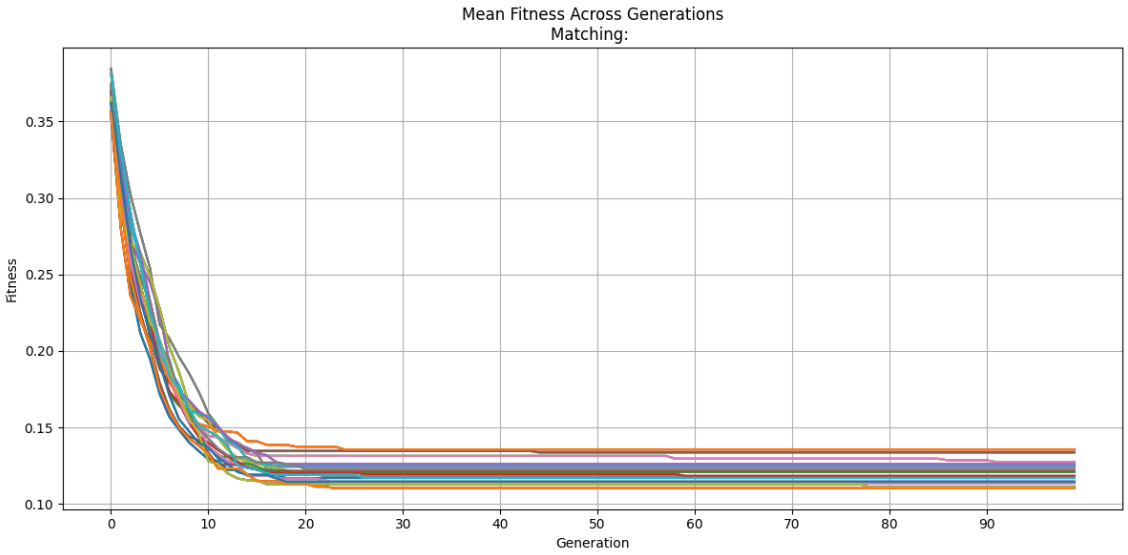

Figure 9- Grid Search

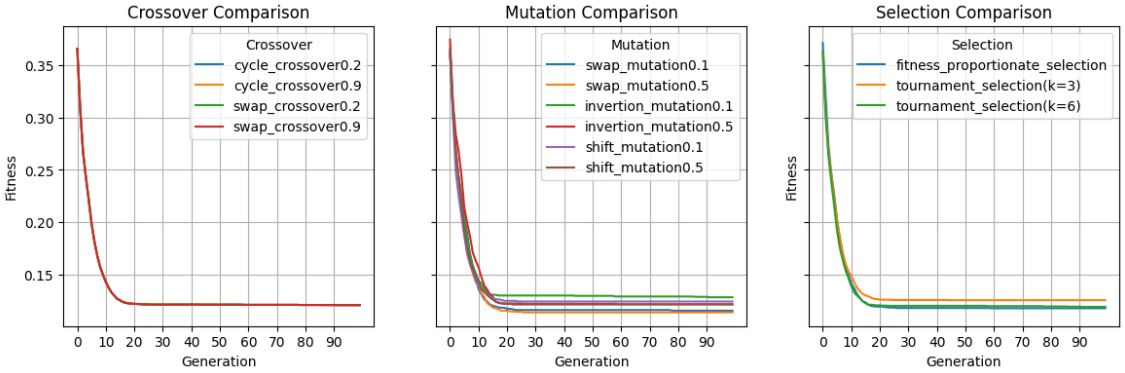

Figure 10- Crossover, Mutation and Selection Comparison

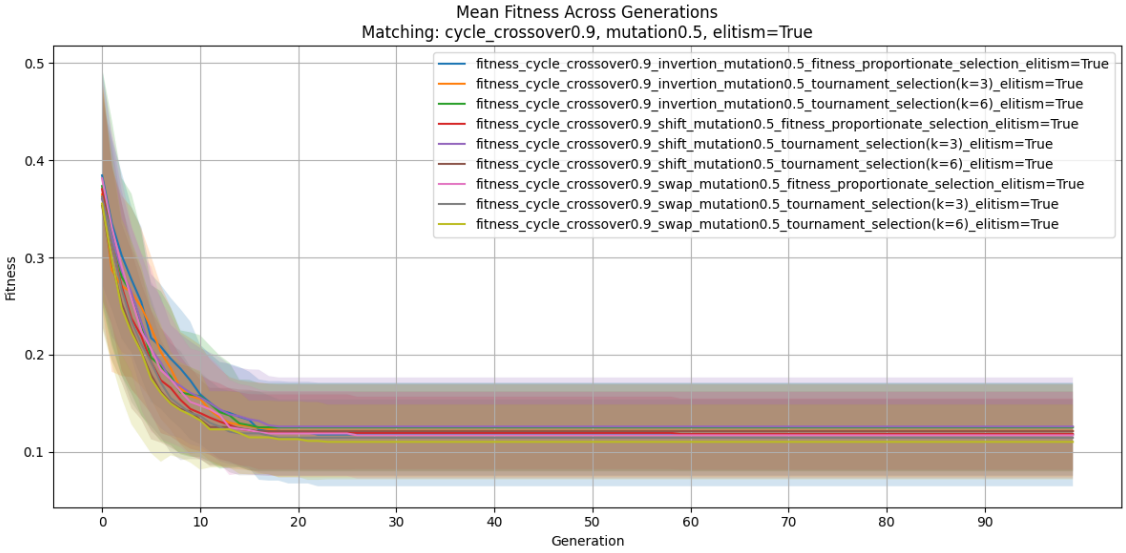

Figure 11- Mean Fitness Across Generations

Tables

Table 1- Crossover Parameters Tested

Crossover Type	Crossover Probability
Cycle Crossover	0.2
	0.9
Swap Crossover	0.2
	0.9

Table 2- Mutation Parameters Tested

Mutation Type	Mutation Probability
Shift Mutation	0.1
	0.5
Invertion Mutation	0.1
	0.5
Swap Mutation	0.1
	0.5

Table 3- Selection Parameters Tested

Selection Type	Maximization Parameter	Tournament Size
Fitness Proportionate Selection (Roulette)	False	Not Applicable
Tournament Selection	False	3
		6

Table 4- Best League Results

Team Number	GK	DEF	MID	FWD
1	Alex Carter	Mason Reed	Ashton Phillips	Zachary Nelson
		Daniel Foster	Nathan Wright	Tyler Jenkings
2	Blake Henderson	Jaxon Griffin	Dominic Bell	Elijah Sanders
		Ethan Howard	Bentley Rivera	Julian Scott
3	Chris Thompson	Logan Brooks	Dylan Morgan	Chase Murphy
		Brayden Hughes	Spencer Ward	Xavier Bryant
4	Ryan Mitchell	Caleb Fisher	Hunter Cooper	Landon Powell
		Lucas Bennett	Gavin Richardson	Sebastian Perry
5	Jordan Smith	Maxwell Flores	Austin Torres	Adrian Collins
		Owen Parker	Connor Hayes	Colton Gray