COMPUTATIONAL INTELLIGENCE FOR OPTIMIZATION

Master in Data Science and Advanced Analytics

NOVA Information Management School

Universidade Nova de Lisboa

Sports League Optimization

Group D

Tahiya Jahan Laboni (20240943)
Antonio Ramos (20240561)
Emir Kaan Kamiloglu (20240945)
Eden da Silva (20240740)
Joao Cardoso(20240529)

TABLE OF CONTENTS

1. Introduction

This project focuses on the challenge of creating balanced teams in a fantasy sports league. The task is to assign 35 players into 5 teams of 7 players each, ensuring that each team has exactly 1 goalkeeper, 2 defenders, 2 midfielders, and 2 forwards. Additionally, every team must stay within a strict budget of €750 million.

To solve this, we implemented a Genetic Algorithm (GA) with custom mutation, crossover, and selection operators adapted to the problem's constraints. In addition to GA, we also evaluated the performance of Hill Climbing (HC) and Simulated Annealing (SA) on the same problem. Our main focus was on evaluating how different operators influence solution quality, convergence speed, and overall balance of the league. We then visualized and analyzed the results to draw insights into the effectiveness of the best-performing configurations.

2. FORMAL DEFINITION OF THE PROBLEM

The goal of this optimization problem is to assign players to teams in a fantasy sports league such that the league is as balanced as possible in terms of average player skill, while respecting position constraints and a strict salary cap. Each team must:

- Include exactly 7 players: 1 Goalkeeper (GK), 2 Defenders (DEF), 2 Midfielders (MID), and 2 Forwards (FWD).
- Not exceed a budget of €750 million per team.
- Ensure each player is assigned to one and only one team.

Search Space: The search space consists of all valid ways to assign 35 players into 5 teams of 7 players each. In our implementation, we created the `***LeagueSolution***` class, which uses the `***_random_valid_assignment***` method. This method generates a shuffled list of team IDs (0 – 4), evenly distributed, ensuring an initial balanced distribution of players across teams. Each solution is stored as an assignment list, where the index corresponds to a player and the value indicates the team they belong to. To ensure the validity of a solution, we implemented the `***is_valid()***` method. This checks that each team contains exactly the required number of players, maintains the correct positional structure, and does not exceed the team budget. Only valid assignments are considered part of the search space.

Representation: Each solution is represented by an `***assignment***` list, where the index corresponds to a player and the value indicates the team to which the player is assigned (from 0 to 4 for 5 teams).

Fitness Function: We implemented a `***fitness***` function to evaluate how balanced each team configuration is. The function works by first grouping players into their assigned teams using the assignment list. For each team, it computes the average skill rating of its players. Then, it calculates the standard deviation of these team averages. A lower standard deviation indicates a more balanced distribution of talent across the teams, which is the objective.

If a solution violates any constraints—such as incorrect team size, improper role composition, or exceeding the €750 million budget—a penalty is applied by returning a very high fitness value ***(float("inf"))***. This condition helps us to explore invalid solutions and ensure feasible and fair team formations.

3. DESCRIPTION OF IMPLEMENTED SELECTION AND GENETIC OPERATORS

To evolve balanced team assignments, we designed custom mutation, crossover, and selection operators that respect the constraints of the problem.

Mutation Operators: In this project, we implemented **three** custom mutation operators to modify candidate solutions while preserving the problem constraints. These operators help to explore the search space and improve diversity during evolution.

1. ***`mutate_swap` :***

This operator randomly selects two players and swaps their team assignments. It creates small but impactful changes in the solution by redistributing players, enabling exploration of nearby configurations without significantly disrupting the team structure.

2. ***`mutate_team_shift` :***

This operator adds a random shift (between 1 and 4) to the team ID of every player in the assignment. The result is a cyclic shift of player assignments across teams. This mutation introduces global movement in the solution and helps escape local optima by rotating the entire team composition.

3. ***`mutate_shuffle_team` :***

This operator first randomly selects a team, then shuffles the player indices within that team. All selected players remain in the same team, but their assignment list indices change. It maintains the internal composition of a team while still introducing variation that affects crossover behavior.

Crossover Operators: To generate new offspring from parent solutions, we implemented two crossover operators that combine characteristics from two assignments while maintaining feasibility of team configurations.

1. ***`crossover_one_point` :***

This operator performs a one-point crossover. It selects a random cut-point in the assignment list and creates a child by combining the first segment from the first parent and the remaining segment from the second parent. This method promotes partial inheritance of structural patterns from both parents while introducing variety through recombination.

2. ***`crossover_uniform` :***

This operator applies uniform crossover across the entire assignment. For each player, it randomly chooses whether to inherit the team assignment from parent1 or parent2. This approach ensures high diversity in the offspring by mixing assignments at a finer level, while still preserving feasible team compositions from the parent solutions.

Selection Mechanisms: To guide the evolution of team configurations, we implemented two selection strategies that determine which individuals are chosen to reproduce. These strategies balance between choosing the best solutions (exploitation) and maintaining diversity (exploration).

1. ***`selection_tournament` :***

This method randomly selects a small group of individuals from the population and evaluates their fitness. The individual with the best fitness among this group is chosen for reproduction. This strategy encourages competitive selection while maintaining randomness, promoting a balance between high-performing and diverse solutions.

2. `selection_ranking` :

This method ranks the entire population based on fitness scores and assigns probabilities to individuals according to their rank. Higher-ranked individuals are more likely to be selected, but lower-ranked ones still have a chance. This supports gradual and stable evolution while avoiding premature convergence to local optima.

4. PERFORMANCE ANALYSIS

We tested four different GA configurations (mutations, crossovers, selections) and compared them to a Hill Climbing baseline. This analysis helps us determine which configuration finds the most balanced team setup, how fast it converges, and how stable the solutions are. It also allows us to assess whether GA or HC is more suitable for this problem under given constraints.

The primary metric used was Final Fitness, calculated as the standard deviation (SD) of the average skill levels across the five teams. A lower SD indicates a more balanced league. Additionally, we also tracked the number of iterations and processing time, indicating how quickly each method found its best solution.

Method	Final Fitness(SD)	Iterations	Processing Time (s)
Config 1	0.1400	30	9.5410
Config 2	0.2100	30	24.2313
Config 3	0.4371	30	10.0522
Config 4	0.1069	30	22.5404
Hill Climbing	0.1069	5	0.2078
Simulated Annealing	0.2770	1001	40.8123

Key Insights: *(Appendix 1)*

- **Config 4** and **Hill Climbing** both achieved the lowest SD (**0.1069**), indicating the most balanced team distributions. Additionally, **Hill Climbing** converged in only 5 iterations, demonstrating efficient local optimization with fewer steps. **Config 4**, while slower than HC, still demonstrates strong balance, showing that GA with appropriate operator combinations can approach HC's quality. Furthermore, **Configs 2 and 3** resulted in **higher SDs**, indicating less balanced teams and likely suboptimal parameter settings.

Simulated Annealing achieved a moderate SD of **0.2770**, but required **1001 iterations** and the longest processing time (40+ seconds), indicating a less efficient convergence path and more randomness in solution quality.

This analysis shows that although GA (Config 4) can match the solution quality of Hill Climbing (HC), HC achieves it faster and more efficiently. The trade-off lies in GA's broader search capability versus HC's speed and simplicity.

5. Implementation of Evolutionary Algorithms

To effectively compare optimization strategies, we designed the implementation of the Genetic Algorithm (GA), Hill Climbing (HC), and Simulated Annealing(SA), which allow us flexible configuration and experimentation with different operators. This structure made it easier to try out different combinations and see how each one affected the results, such as how fast the method worked and how balanced the teams were.

5.1 Genetic Algorithm

We designed a ***`genetic_algorithm`*** function to support experimentation with various population sizes, mutation rates, elite preservation, and operator choices. The goal was to study how different combinations affect convergence and final team balance. It begins by creating a valid initial population using generate_population, which repeatedly generates candidate solutions and filters them using the ***`is_valid()`*** check.
For each generation:

- The population is sorted by fitness (standard deviation).
- The top-performing individuals (elite) are preserved.
- Parent solutions are selected using the configured selection strategy.
- Crossover is applied to generate children.
- Mutation is optionally applied based on the mutation rate.
- Only valid children are added to the new population.

This evolutionary cycle runs for a defined number of generations (e.g., 30), tracking the best solution over time.

5.2 Hill Climbing

We implemented HC as a benchmark to evaluate the local search efficiency compared to GA's global search capability. It starts from a valid solution and iteratively generates neighboring solutions using the ***`get_neighbors`*** method from the ***`LeagueHillClimbingSolution`*** class. At each iteration:

- The best neighbor with a lower fitness is chosen.
- If no better neighbor exists, the process halts.

The HC method is more greedy and converges faster, but is prone to getting stuck in local optima compared to GA.

5.3 Simulated Annealing

We implemented a `***simulated_annealing***` to improve global exploration beyond local optima. We start from a valid solution, generate neighbors via `***mutate_swap()***` and decide acceptance based on fitness and a temperature-controlled probability. The temperature decreases over time, reducing the likelihood of accepting worse solutions. This allows SA to explore broadly early on and focus locally later. Though slower to converge, SA offers better global search compared to Hill Climbing.

5.4 Summary Comparison

This dual setup enabled us to compare GA's broad global search with HC's fast local convergence. GA explores more solutions with flexible operators, while HC quickly finds good local optima. Together, they highlight different strengths in achieving balanced team configurations.

6. Conclusion

In this project, we tackled the challenge of forming balanced teams in a fantasy sports league by applying three evolutionary optimization techniques: Genetic Algorithm (GA), Hill Climbing (HC) and Simulated Annealing(SA).GA gave the most flexible results by testing different settings and creating well-balanced teams. HC was the fastest method, but it sometimes got stuck in local solutions. SA used a smart way to accept worse solutions at first, helping it explore more and avoid getting stuck early. Although it was slower, SA found better results in the long run.
Overall, GA proved most versatile, HC was fastest, and SA offered stronger global exploration. The most suitable algorithm ultimately depends on the specific goal—whether prioritizing speed, solution diversity, or overall balance.

7. Bibliographical References

Books

● Goldberg, D. E. (1989). *Genetic Algorithms in Search, Optimization, and Machine Learning*. Addison-Wesley.

● Mitchell, M. (1998). *An Introduction to Genetic Algorithms*. MIT Press.

Research Articles

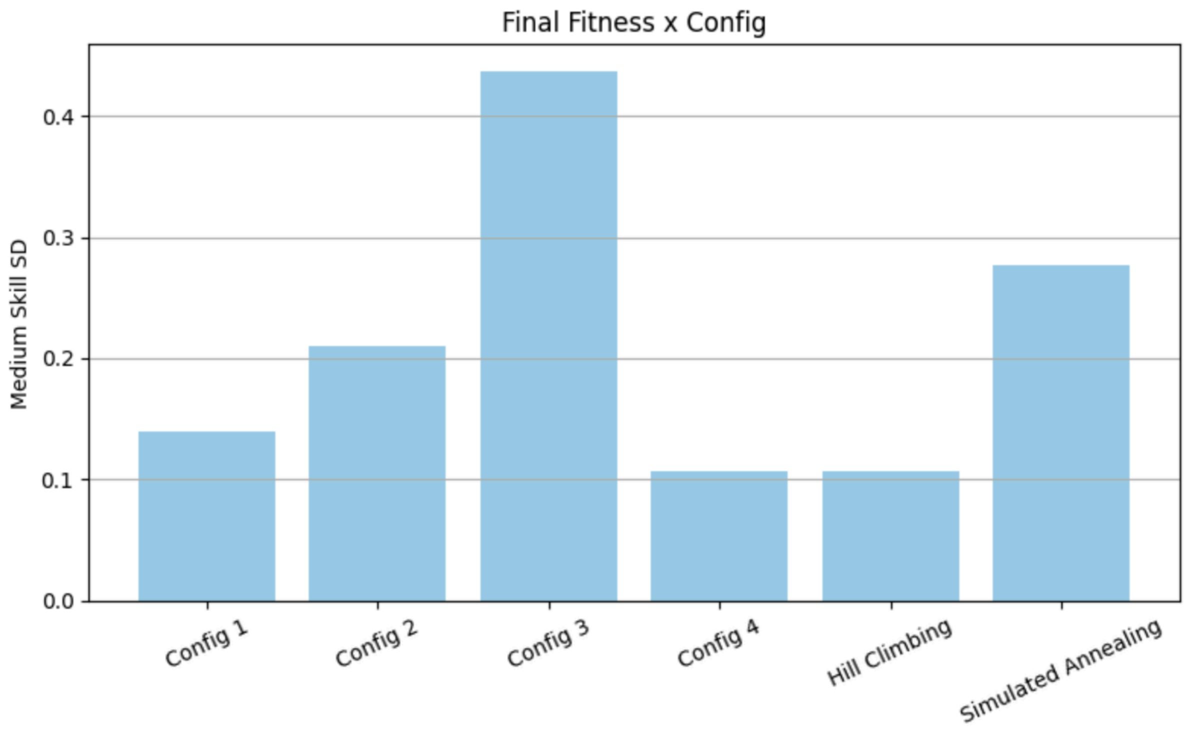
● Whitley, D. (1994). A Genetic Algorithm Tutorial. *Statistics and Computing*, 4(2), 65–85.

● Haupt, R. L., & Haupt, S. E. (2004). *Practical Genetic Algorithms*. Wiley.

AI Research Blogs

● Google AI Blog. https://ai.googleblog.com

● OpenAI Blog. https://openai.com/blog



Appendix 1 – Final Fitness per GA Configuration