

NOVA

IMS

Information
Management
School

MDSAA

Master Degree Program in
Data Science and Advanced Analytics

Computational Intelligence for Optimization

Paulo Martins, r2015469
Beatriz Fonseca, r20201599

Group AI

NOVA Information Management School
Instituto Superior de Estatística e Gestão de Informação

Universidade Nova de Lisboa

Project available on [GitHub](#)

INDEX

1. Problem Definition.....	2
2. Genetic and Selection Operators	2
2.1. Mutation Operators.....	2
2.2. Crossover Operators	3
2.3. Selection Mechanisms	4
3. Design Choices.....	4
4. Performance analysis.....	4
5. Recommendations.....	6

1. PROBLEM DEFINITION

This project aims to create a **balanced sports fantasy league** using **genetic algorithms**. The league is composed of **five teams of seven players** each. Each team must have one goalkeeper, two defenders, two midfielders and two forwards. Each player must belong to one team only and each team's combined players salaries must not exceed **€750 million**.

To create a balanced league, the average player skill across all teams must be roughly the same. To achieve this, a **fitness function** was defined as the standard deviation of the average skill rating of each team in the league. However, if the solution is invalid, the fitness takes an infinite positive value, since this is a **minimization problem**.

Each league configuration is an **individual**, represented as a 5x7 matrix - five teams of seven players each - while the **population** is represented as a cube of N matrices, N being the number of individuals of the population.

The **search space** is all possible partitions of the players into five teams of seven players, satisfying the positional and cost constraints.

2. GENETIC AND SELECTION OPERATORS

2.1. MUTATION OPERATORS

The **mutation operators** that were used were **permute**, **roll** and **rubix**, a combination of roll and swap.

When using **permute**, a positional block is selected, and its elements are shuffled.

GK	DEF	DEF	MID	MID	FWD	FWD
0	5	10	15	20	25	30
1	6	11	16	21	26	31
2	7	12	17	22	27	32
3	8	13	18	23	28	33
4	9	14	19	24	29	34

GK	DEF	DEF	MID	MID	FWD	FWD
0	6	14	15	20	25	30
1	8	11	16	21	26	31
2	7	5	17	22	27	32
3	12	13	18	23	28	33
4	9	10	19	24	29	34


When using **roll**, the elements of a given column are shifted vertically by a given number of steps.

GK	DEF	DEF	MID	MID	FWD	FWD
0	5	10	15	20	25	30
1	6	11	16	21	26	31
2	7	12	17	22	27	32
3	8	13	18	23	28	33
4	9	14	19	24	29	34

GK	DEF	DEF	MID	MID	FWD	FWD
0	5	10	15	20	27	30
1	6	11	16	21	28	31
2	7	12	17	22	29	32
3	8	13	18	23	25	33
4	9	14	19	24	26	34

When using **rubix**, either a roll, a swap or a combination of both can be applied to an individual. Below is a representation of the swap operation.

	GK	DEF	DEF	MID	MID	FWD	FWD
0	5	10	15	20	25	30	
1	6	11	16	21	26	31	
2	7	12	17	22	27	32	
3	8	13	18	23	28	33	
4	9	14	19	24	29	34	

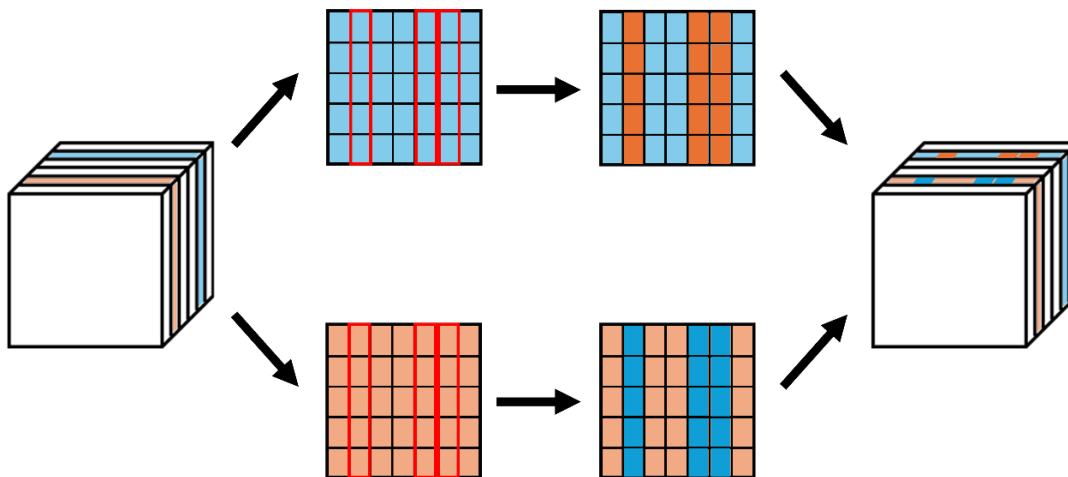


	GK	DEF	DEF	MID	MID	FWD	FWD
0	5	10	15	20	25	30	
1	11	6	16	21	26	31	
2	12	7	17	22	27	32	
3	8	13	18	23	28	33	
4	14	9	19	24	29	34	

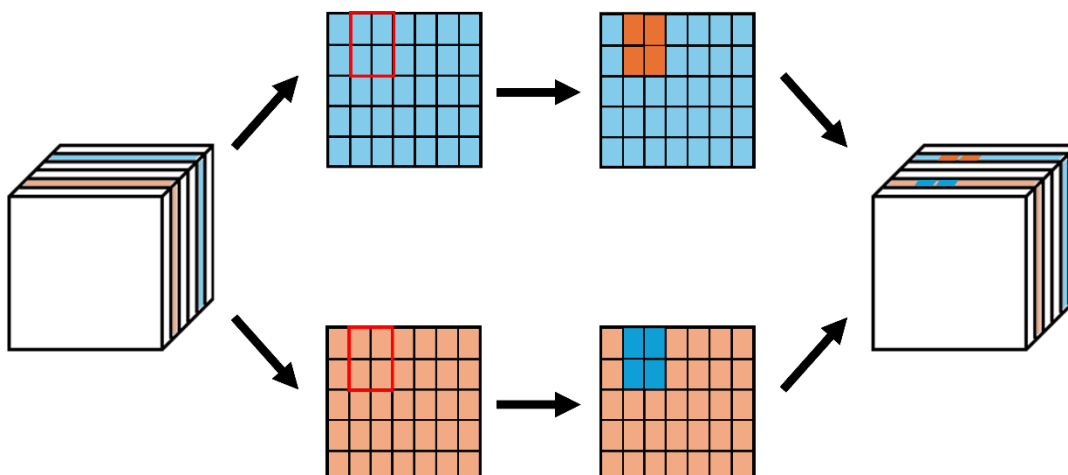
2.2. CROSSOVER OPERATORS

Regarding the **crossover operators**, two were implemented – **block crossover** and **cutoff crossover**.

The **block crossover** performs a column-wise crossover between pairs of individuals of the cube, based on a map which indicates which columns to swap for each pair. For each pair of individuals, the specified columns are swapped, resulting in a new set of individuals.



The **cutoff crossover** swaps rows between two individuals for a given positional block – these rows are defined by the cutoff parameter.



2.3. SELECTION MECHANISMS

Two selection mechanisms were implemented - **tournament** and **poison**.

The **tournament selection** uses fitnesses to select n individuals, based on **rank**, **fitness** itself or in a **probabilistic** fashion. When using **rank**, the individuals are assigned a numerical rank based on their fitness and the best ranked individual is chosen; when using **fitness**, the individual with lowest fitness (best individual) is directly selected; and when using **probabilistic** a softmax function is applied to the negative values of the individuals' fitnesses, effectively assigning a higher probability of choosing the individuals with better fitness, and then choosing one individual from the resulting distribution at random.

The **poison selection mechanism** simulates a lethal environment based on a smooth probability curve where only the fittest survive. Based on a fitness threshold, the death probability of an individual is calculated using a sigmoid function. In effect, the bigger the difference between the individual's fitness and the defined threshold, the more extreme the probability of death is, i.e., if the fitness is much larger than the threshold, the probability of dying is close to 1, whereas if it is much small, the probability tends to 0.

This probability is also affected by sharpness which represents the slope of the sigmoid function. A lower sharpness value promotes diversity by providing the individuals with fitnesses closer to the threshold a similar chance of survival; on the other hand, a higher sharpness value encourages elitism and favors only the individuals with the best fitnesses. Then, a random trial occurs and only the lucky survive.

3. DESIGN CHOICES

The **individuals** were represented as matrices of 5 teams of 7 players each, with each column index of the matrix representing the player position. This representation enables us to model the solution space as a categorical algebra, allowing nearly all of our operators to be applied directly as functors - acting as morphisms within the algebra.

Crucially, the choice of this representation ensures that the algebraic constraints of the space are inherently respected. Furthermore, although not fully optimized, the cube structure naturally supports tensor application, paving the way for enhanced optimization and parallelization.

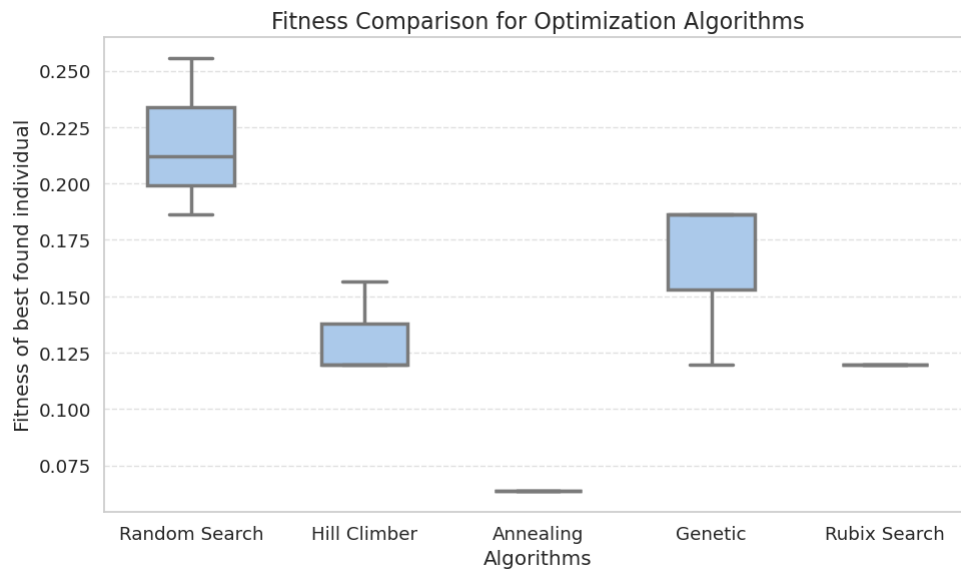
The **fitness function** was designed as the standard deviation of the teams' average player skill rating, which helped measure the model's success.

4. PERFORMANCE ANALYSIS

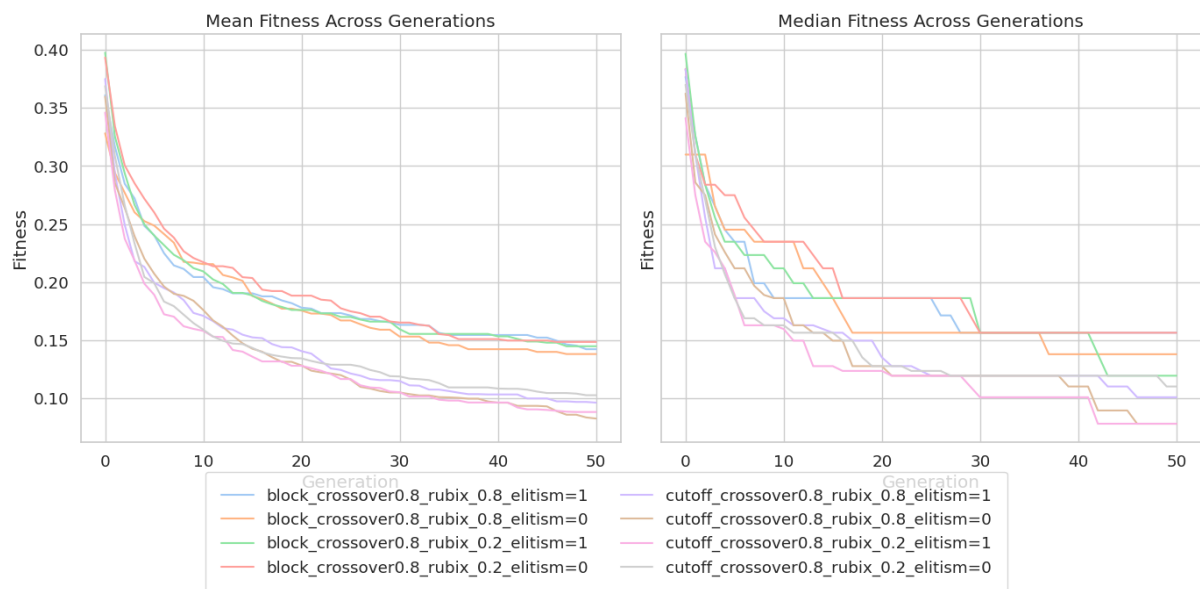
To bring confidence to the results of the genetic algorithm, it was compared against other optimization algorithms as well as against other versions of itself with different operators, by running these algorithms for 30 runs and averaging results.

In the end, the configuration of the genetic algorithm performed well against Hill Climbing, but did not perform any good vis the remaining algorithms, proving that it needs some extensive work. On the

other hand, Rubix search performed very consistently even though surpassed by the Annealing implementation, as it can be seen in the plot below.



Different operators of the genetic algorithm were also tested. As per the following image, one can conclude that the cutoff crossover gets better fitness results and clearly beats block crossover; however, implementations with the latter seem to converge faster. Also, elitism yielded mixed results –it shows better performance when having lower probability of mutation, whereas when using a higher probability, elitism actually produces worse results.



5. RECOMMENDATIONS

The codebase would benefit from further optimization, particularly through CUDA parallelization. Despite the simplicity of the current implementation, the Rubix search performed well and demonstrated efficient parallel execution. However, the implementation of the cutoff crossover operator proved to be challenging and would benefit from additional refinement and future enhancements.

The current implementation offers a rich testing ground for grid parameter exploration across all of the algorithms that were implemented with an easy interface via configuration files.