NOVA IMS Information Management School

MAY 2023

STIGLER'S DIET PROBLEM

PROJECT REPORT

Computational Intelligence for Optimization

Link to GitHub:
https://github.com/daila10/CIFO_Group21.git

Made by:
Daila Alexandre 20191182
Diogo Silva 20221393
Luís Fernandes 20221649
Skander Chaabini 20221041

Professors:
Leonardo Vanneschi
Berfin Sakallioglu

Index

Division of labor:
Data collection: *Diogo*
Algorithm design: *Daila, Luís, Diogo, Skander*
Algorithm implementation: *Daila, Luís, Diogo*
Fitness evaluation handling: *Daila, Luís, Diogo*
Experimentation and parameter tuning: *Daila, Luís, Diogo*
Result analysis and interpretation: *Daila, Luís, Diogo, Skander*
Visualizations: *Diogo, Daila, Luís*
Documentation and reporting: *Luis, Diogo, Daila, Skander*

1. Introduction

This report presents an application of genetic algorithms to the optimization of a dietary plan, exploring the effectiveness of different genetic operators and strategies in generating optimal dietary plans. The principles of selection, mutation, and crossover are leveraged in order to evolve a population of individuals towards solutions that strike a balance between nutritional adequacy and minimized cost.

2. Genetic Algorithm applied to the Problem

The Stifler's diet problem involves selecting a subset of food items from a given dataset that satisfies nutritional constraints while minimizing the total cost – the dataset gives the nutrients present in 1 gram of each ingredient and the price, in cents, were inputted by the group. From the dataset the essential macros and 12 essential vitamins, but due to almost non-existent values for vitamin-D it was deleted. Each individual is encoded as a set of genes, with each gene indicating a specific food item and its quantity, included in the diet.

Decision Variable

Quantity: The decision variable refers to the selection of food and its respective quantity to be included in the diet plan. There are 58 different food items available. Therefore, the decision variable can be represented as follows:

- ***quantity[i]:*** *denotes the quantity of the i-th food item to be included in the solution, where i ranges from 1 to 58. That quantity will be represented in grams unit (g).*

By manipulating the values of the decision variable, the GA explores different combinations of food items, seeking an optimal or near-optimal solution.

3. Problem Formulation

3.1. **Problem Formulation:** Diet Optimization

3.2. **Objective:** The objective of the diet optimization problem is to find an optimal or near-optimal combination of food items that meets specific nutritional requirements while minimizing the cost of the diet.

3.3. **Constraints:**

3.3.1. **Nutritional Requirements:** The diet must satisfy minimum nutritional requirements for various nutrients, including calories, carbohydrates, protein, total fat, vitamins, and minerals. These requirements are based on recommended daily intake values.

3.3.2. **Food Selection:** Each food item has a quantity associated with it, indicating the amount to include in the diet. The quantity of a food item must be a non-negative integer value.

3.3.3. **Available data:** Solutions are restricted to foods from which data have been obtained. In a broader context, there is a much larger pool of food options that would enhance the diversity of solutions.

3.4. **Representation:** In the genetic algorithm, an individual represents a potential diet plan. The individual is represented as a list of integers, where each element corresponds to a food item and its value represents the quantity of that item to include in the diet. A value of zero indicates that the food item is not included, while a positive integer represents the quantity in grams. This representation was chosen due to its simplicity and ease of applying various genetic operators.

In the given code, a specific design choice was made to address the scarcity of foods with protein in the problem domain. To mitigate the risk of losing good individuals in terms of protein content during the crossover operation, a modification was implemented. This modification involved rearranging the order of foods in the representation list. Foods with

higher protein content were placed together at the beginning of the list. By doing so, the probability of losing individuals with desirable protein characteristics during crossover was reduced.

3.5. **Fitness Function:** The fitness function evaluates the quality of a diet plan based on two criteria: meeting the nutritional requirements and minimizing the cost. The fitness value is calculated as the total cost of the selected food items plus a penalty for not meeting the minimum nutritional requirements or for passing the recommended maximum. The penalty is proportional to the shortfall in each nutrient from its minimum requirement and will be accessed later the report.

$$Fitness = Cost + Penalty$$

3.6. **Initialization:**

3.6.1. **Random Initialization**: Each individual in the population is randomly assigned to a set of quantities for the food items (generation of random integer values for each food item, representing the quantity in grams). This initialization introduces diversity in the initial population.

3.6.2. **Latin Hypercube Sampling variation:** a method that generates a diverse and representative set of initial solutions for optimization problems. It divides the value range into compartments and randomly selects one value from each, ensuring comprehensive exploration of the problem. This approach promotes solution space coverage and enables efficient discovery of promising solutions by optimization algorithms.

3.6.3. **Good foods:** As mentioned, the schemata of the individual were changed in an attempt to have a greater chance of better individuals surviving, in this vein, this initialization gives a random quantity to foods that are considered good and foods that are considered bad start with zeros.

In the context of this project, the allocation between a good food and a bad food can be considered subjective and is purely based on our common knowledge. Examples of good foods are fruit, chicken and rice, examples of bad foods are chocolate, champagne and ice cream.

3.7. **Genetic Operators:** Genetic operators such as mutation, crossover and parent selection will be explored in section 5 of this report.

3.8. **Termination Criteria:** The algorithm terminates when one or more of the following conditions are met:

- A maximum number of generations is reached.
- A predetermined number of generations pass without any improvement in the best fitness value.

4. Genetic Operations

4.1. Crossover

The following crossover operators were utilized in the study to facilitate the exploration and exploitation of the search space in the genetic algorithm.

- **Single-Point Crossover (single_point_co):** Selects a random crossover point and combines the genetic material of parents at that point to create two offspring.
- **Uniform Crossover (uniform_co):** Randomly selects genes from parents to create offspring, with a 50% probability of selecting a gene from each parent at each position.
- **Multi-Point Crossover (multi_point_co):** Selects multiple crossover points and alternates the genetic material between parents at those points to create two offspring.

- **Arithmetic Crossover (arithmetic_co):** Computes the weighted average of the corresponding genes from parents to generate offspring.
- **Geometric Crossover (geometric_co):** Performs geometric crossover by taking a weighted average of the corresponding genes from parents to produce offspring. These crossover methods offer different mechanisms for combining genetic material from parents to generate diverse offspring.

4.2. Mutation

A variety of mutation operators were applied to introduce diversity and explore different regions of the solution space.

- **Random Mutation (random_mutation):** Randomly changes the quantity of a food item in the individual. Each element of the individual has a probability (elem_mute_rate) of being mutated. If selected for mutation, a new random quantity between 0 and 50 is assigned to the food item.
- **Geometric Mutation (geometric_mutation):** Applies a geometric mutation to the individual. Each element of the individual has a probability (elem_mute_rate) of being mutated. If selected for mutation, the original value is scaled by a random factor between 1 - scale_factor and 1 + scale_factor.
- **Insert-Delete Mutation (insert_delete_mutation):** Performs an insert or delete mutation on the individual. It randomly chooses between inserting a new quantity for a food item with a quantity of 0 or deleting a quantity from a food item with a non-zero quantity. By adding or removing genes, insert-delete mutation introduces structural variations in the individuals, potentially leading to improved solutions.

These mutation methods provide different ways to introduce variations into the individuals of the population. Random mutation and geometric mutation modify the quantity of individual food items, while insert-delete mutation adds or removes items from the individual.

4.3. Parent Selection

The following selection methods were implemented and evaluated to assess their effectiveness in promoting desirable individuals and maintaining diversity in the population.

- **Fitness_Proportional_Selection (fps):** It calculates the total fitness of the population and selects an individual based on a random spin value. The probability of an individual being selected is proportional to its fitness compared to the total fitness of the population – individuals with higher fitness have higher probability of being selected.
- **Ranking Selection (ranking_selection):** The population is sorted based on fitness, and each individual is assigned a probability based on its rank. Individuals with higher fitness have a higher probability of being selected.
- **Tournament_Selection (tournament_selection):** It randomly selects a specified number of individuals (tournament size) from the population and compares their fitness. The individual with the best fitness is selected as the winner.

These selection methods offer different ways to choose individuals for reproduction in the evolutionary process. Fitness Proportional Selection and Ranking Selection consider the fitness values to determine the selection probabilities, while Tournament Selection compares fitness directly among participants.

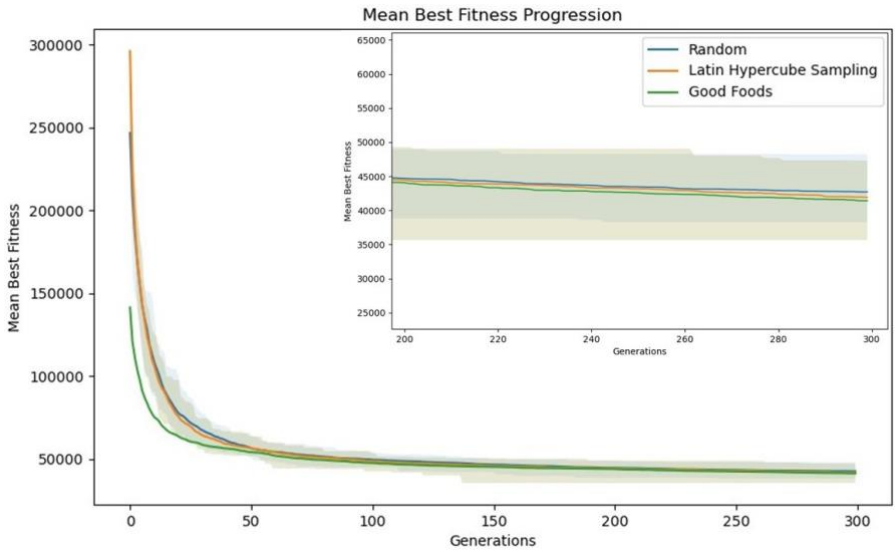
5. Experimental Setup and Analysis

As trying all combinations is computationally expensive and takes hours to run, another experimental approach was performed. First, the different methods of initialization and genetic

operations were tested while keeping all parameters fixed. The method giving the best results and meeting the highest number of requirements was selected.

5.1. **Genetic Operations**

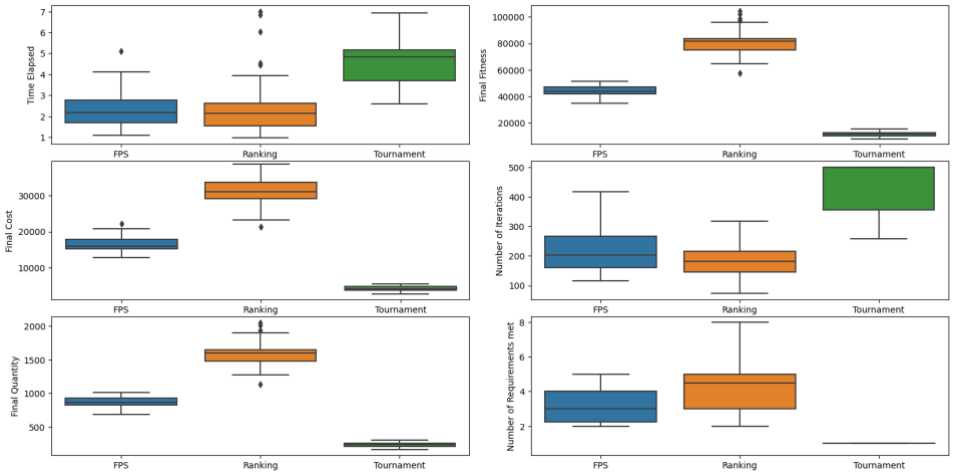
5.1.1. **Initialization**

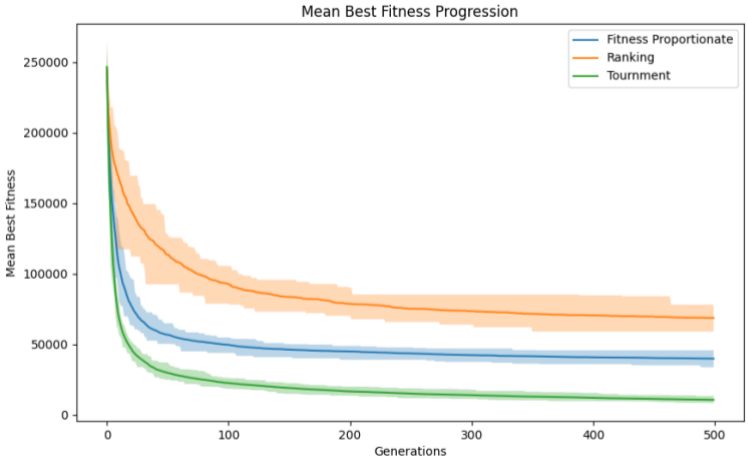


To evaluate performance, the mean best fitness achieved by each initialization method over multiple runs was performed and represented graphically. The x-axis represents the number of generations, and the y-axis represents the average fitness values. Upon careful examination of the graph, it was observed that there was no significant difference in the mean best fitness values among the three initialization methods. While the Good Foods method started with slightly lower mean best fitness values, the difference was not significant. Consequently, based on the graph and statistical analysis, it was concluded that there was no clear advantage in terms of fitness improvement by using any specific initialization method. Therefore, it was decided to proceed with the random initialization method for its simplicity and comparable performance to the other methods.

5.1.2. **Selection**

To assess the performance of different parent selection methods, namely Fitness Proportionate, Ranking, and Tournament, an evaluation was also carried out. The evaluation was conducted in conjunction with the fundamental genetic operators of random mutation and one point crossover. The experimentation involved running the algorithm for 50 independent trials, each consisting of 500 generations with a population size of 50 individuals – and these parameters were also used in mutation and crossover.

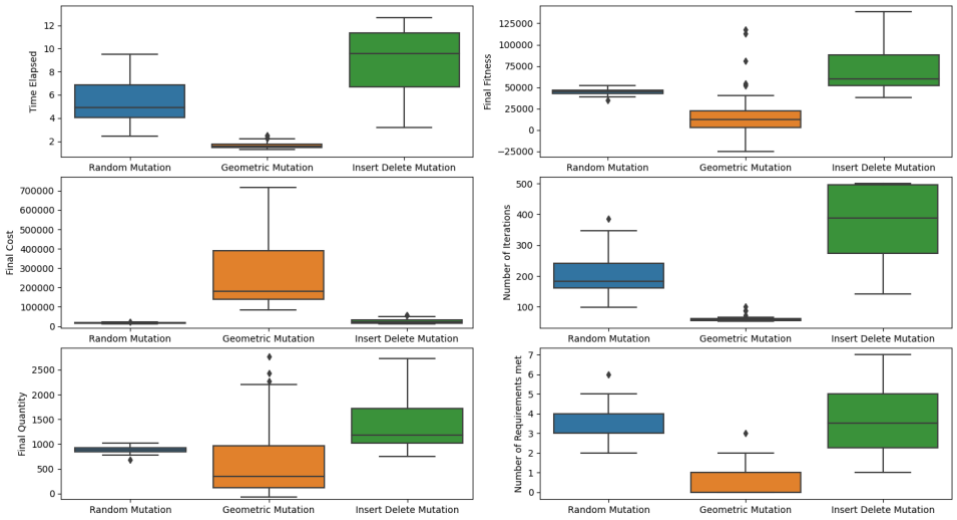


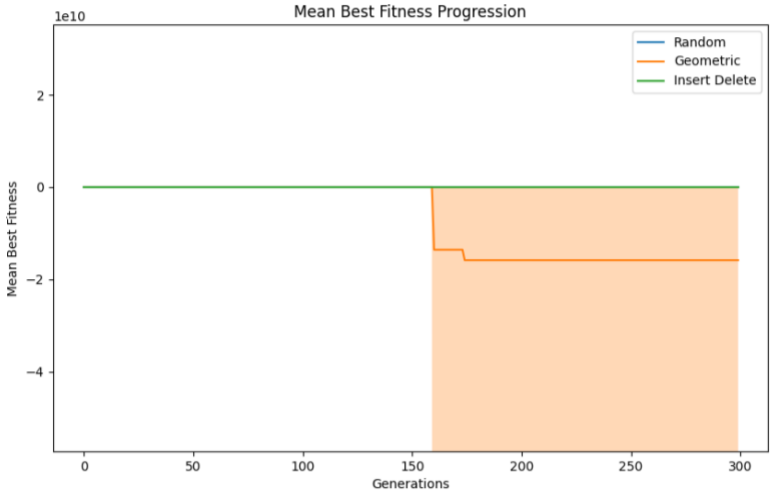


Even though the Tournament selection had the best fitness throughout all generations, it did not meet more than one requirement. Fitness Proportionate was the chosen operator as it provided better fitness and cost compared to the Ranking selection.

5.1.3. **Mutation**

The plot depicting the evolutionary progress of the algorithms with various mutation operators was performed with 500 generations, but the line plot is limited to 250 iterations because beyond this point, the fitness values exhibit no further improvement, restricting the plot to this range enhances clarity and enables easier interpretation of the evolutionary trends.

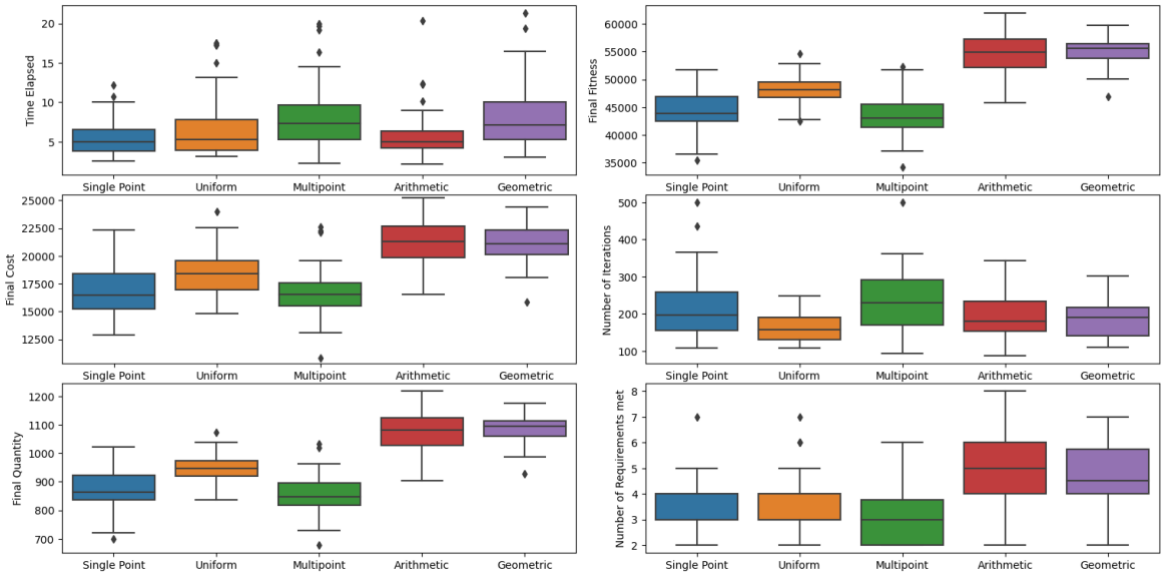


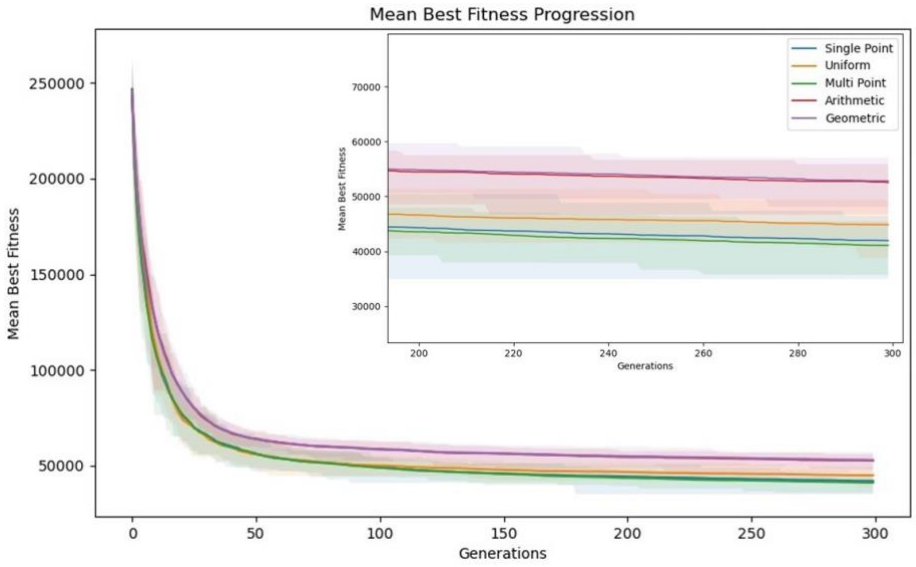


The Geometric Mutation had the best fitness and the less time consumed. However, Insert-Delete Mutation and Random Mutation methods met more requirements with less total cost. Thus random mutation is the chosen method.

5.1.4. **Crossover**

Similarly, to the comparison of the Mutation Operators, the plot is limited to 300 generations.



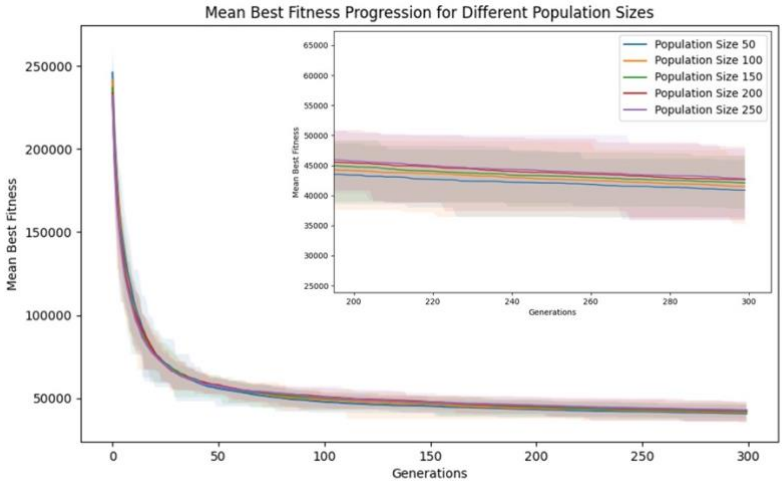


The Arithmetic and Geometric crossovers gave similar results, with a higher final cost, a worst fitness, but more requirements met. On the other hand, the three remaining methods gave better fitness, lower cost but met less requirements. The Multipoint crossover showed slightly better fitness and costs and so was chosen to be used in the final combination.

5.2. **Hyper Parameters**

5.2.1. **Population Size**

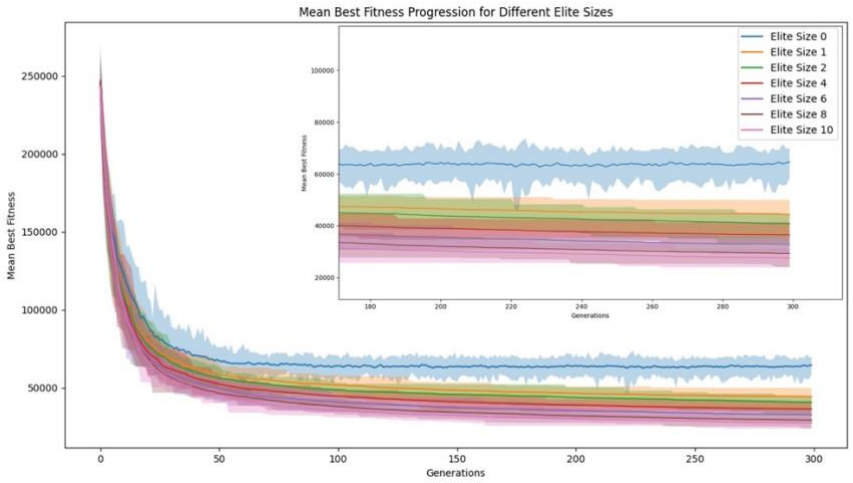
To investigate the effect of population size on the diet optimization problem, a series of experiments were conducted with varying population sizes. The population sizes chosen for testing vary from 50 to 250 individuals. The comparison was done using the genetic operators selected previously.



A smaller population size, such as 50 individuals, allowed for faster convergence to a better fitness due to fewer evaluations and less computational overhead enabling the algorithm to explore and exploit the solution space more efficiently. On the other hand, a larger population size enabled the algorithm to search a wider range of the solution space, therefore enhanced solution diversity. However, it led also to longer computation times. Throughout the experiments, a population of 50 individuals seemed to be enough to reach satisfactory solutions.

5.2.2. **Elite Size**

The elite size determines the number of top-performing individuals preserved from one generation to the next. During the experimentation with different elite sizes, a range of values from 0 to 10 were tested.



After evaluating the fitness values with different number of elite individuals from each generation, it was found that an elite size of 10 produced the best fitness results. It was also found that the bigger the elite size, the better the fitness was, however, having 10 elite individuals is not ideal for a population of size 50, as it decreases diversity. Therefore, the elite size was set on 6.

5.2.3. **Number of Generations**

Based on the experimental observations, it was consistently observed that the algorithm exhibits an "elbow" phenomenon after approximately 25-30 generations. This elbow point signifies a point of diminishing returns in terms of improvement. Beyond this point, the algorithm's progress in solution quality becomes less noticeable. Additionally, the findings consistently showed that continuing the algorithm for an extended number of generations, specifically beyond 250-300 generations, results in almost no improvement. Therefore, it was opted for 300 generations.

5.2.4. **Fitness Function**

The fitness function underwent several trials to achieve a suitable balance between cost and meeting requirements. Initially, a fixed penalty approach was employed for every requirement that was not met, but it failed to differentiate between a small and a significant deviation from meeting the requirements, such as 1% and 99%, and the GA had difficulty improving. The next attempt incorporated a penalty based to account for the deviation of range of each requirement either it was below the minimum or above the maximum. This fitness function led to a reasonable final price but not all the requirements were met.

Requirement Range = Maximum Requirement - Minimum Requirement

*Penalty = ((Minimum Requirement–Individual Nutrient value)/Requirement Range) *constant*

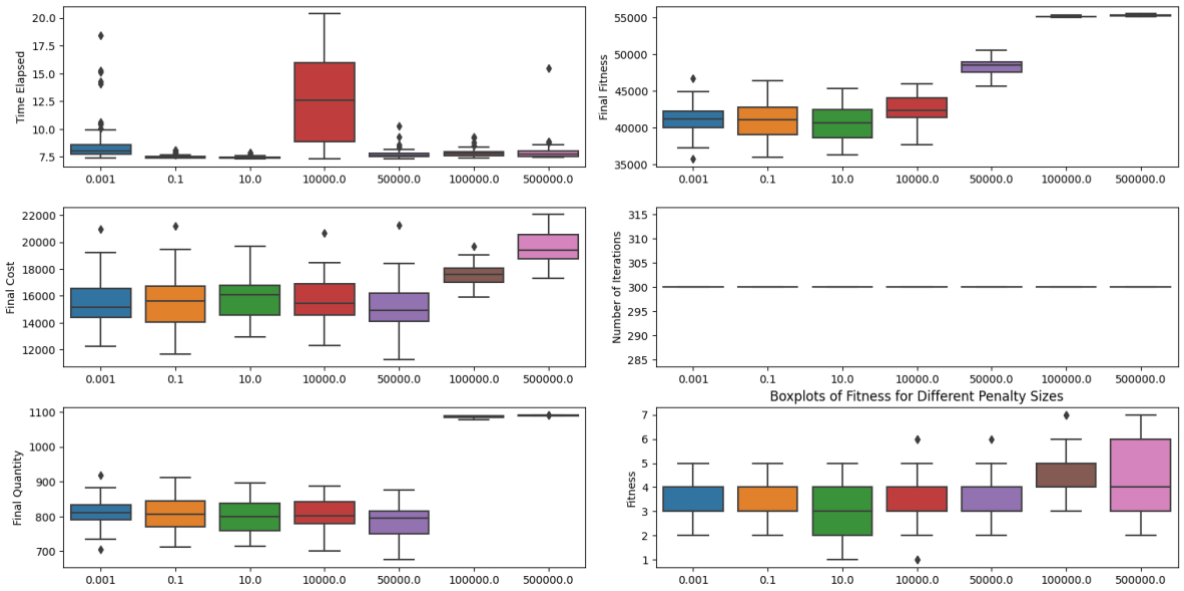
*Penalty = ((Individual Nutrient value–Maximum Requirement)/Requirement Range) *constant*

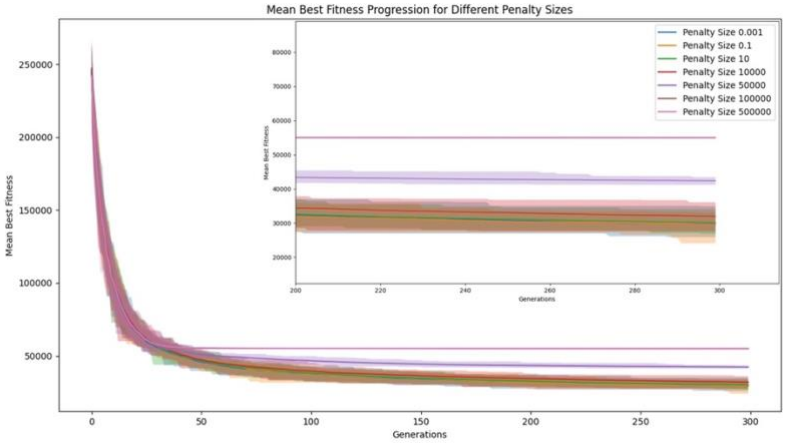
The fitness function went through further modifications, such as applying exponential penalties based on the extent of unfulfilled requirements. Although this led to all requirements being met, the cost was 70 euros, which is excessively high, with around 800% of the requirements being met. Attempts to increase the power of the surplus penalty did not bring significant improvements. Another approach involved adding an extra penalty for each euro over a pre-defined maximum. While this reduced the surplus to around 300%, a final cost of 50 euros was still not reasonable, and using this approach was somewhat biased towards the algorithm.

Ultimately, the chosen fitness function was the first improvement, which yielded reasonable results. It effectively balanced minimizing cost and meeting requirements.

5.2.5. **Penalty**

The penalty size parameter was created in order to make individuals meet nutritional requirements.





It was considered a penalty size associated with not meeting the nutritional requirements. The higher the penalty, the higher the fitness is, as expected, but reaching a certain threshold the penalty size doesn't have an effect.

Accessing the nutritional requirements, one can conclude that higher penalty values lead to higher number of requirements met but not strictly. The employed penalty size was 20 for not meeting the minimum requirements and 5 for exceeding the maximums.

5.2.6. **No Improvement Threshold**

No specific improvement threshold was explicitly defined for the algorithm to stop running if the improvement falls below that threshold. Instead, the focus was on establishing an optimal number of generations. It was perceived that 250 to 300 generations were enough to achieve satisfactory results, it was decided to use these values as a maximum number of generations rather than defining a no improvement threshold.

6. Results

As a setup it was used for a population size of 50 individuals, 300 generations, elite size of 6, and penalty sizes of 20 in case of not meeting the minimum requirements and 5 for exceeding the maximum.

The results obtained from running the genetic algorithm on the diet optimization problem, with the chosen genetic operators and parameters, are summarized in the following tables:

Best Fitness	11379.24	**Total Cost**	1209.9	**Ingredients**	57	**Requirements**	7/14

% of requirement met		
Calories (kcal):	2193.80 (1200)	182.82%
Protein (g):	57.06 (70)	81.51%;
Total Fat (g):	101.04 (44)	229.63%;
Vitamin B6 (mg):	1.34 (1.3)	102.78%;
Vitamin A (IU):	7190.83 (1000)	719.08%
Vitamin B12 (ug):	1.82 (2.4)	75.95%;
Vitamin C (mg):	70.18 (90	77.98%;
Vitamin E (IU):	7.46 (15)	49.76%;
Vitamin K (ug):	46.18 (120)	38.48%;
Thiamin (mg):	1.47 (1.2)	122.72%
Riboflavin (mg):	1.20 (1.3)	92.31%;
Niacin (mg):	18.56 (16)	115.98%;
Pantothenic Acid (mg):	4.30 (5)	86.08%

Optimal diet Composition (g)			
beef: 7	strawberry: 27	pineapple: 15	spaghetti: 2
chicken: 7	kiwifruit: 33	red apple: 20	bacon: 4
taco: 1	potato: 21	green apple: 17	hamburger: 13
burrito: 23	tomato: 11	pear: 1	french fries: 34
mushroom: 11	avocado: 12	peach: 28	pizza: 8
ice cream: 21	eggplant: 48	cherries: 7	hotdog: 34
milk: 39	corn: 50	popcorn: 7	croissant: 49
cheese: 5	hot pepper: 24	french bread: 22	rice crackers: 1
grapes: 31	cucumber: 12	pancakes: 20	fried shrimp:
melon: 44	carrot: 9	doughnut: 10	cookie: 25
watermelon: 2	peanuts: 1	cake: 4	chocolate bar:
tangerine: 9	chestnut: 4	candy: 25	custard flan: 32
lemon: 20	bread: 48	champagne: 12	red wine: 23
banana: 19	rice: 15	black tea: 12	sake: 22
beer: 4			

The optimized diet plan consists of a combination of 57 selected food ingredients. The overall fitness of the solution is 11379.24. However, upon analyzing the nutritional values it is observed that the plan falls short of meeting the minimum requirements for a few nutrients, 5 of the 7 unmet requirements have percentages above 75%, indicating that the obtained diet plan may not provide perfect nutrition, but it can still be adequate. The total cost of the selected food items is 12.10 euros, which seems like a reasonable price range.

7. Discussion and Conclusion

The optimization problem involved a large number of variables and constraints, making it computationally expensive and time-consuming to find optimal solutions.

The algorithm focuses on optimizing a single fitness function, which may not capture all aspects of the problem or adequately balance conflicting objectives.

For example, in addition to meeting the nutritional requirements, it may also be interesting to account for taste preferences, variety, sustainability, or specific dietary restrictions. These objectives may have conflicting trade-offs, where improving one aspect may come at the expense of another.

By solely relying on a single fitness function, the algorithm may prioritize one objective over others. For instance, it may optimize for cost reduction but not adequately consider nutritional balance or diversity of food choices. This limitation can prevent the algorithm from finding optimal solutions that strike a balance between various objectives.

The dataset consisted of only 58 different food items, which may not encompass the full range of dietary options available. This limited selection of foods posed challenges in meeting all the nutritional requirements satisfactorily.

One significant observation was the lack of sufficient sources of protein and vitamin D in the dataset. As a result, despite the optimization efforts of the genetic algorithm, it is understandable that some nutritional requirements were not fully met.

Furthermore, the potential applications of the genetic algorithm in the field of nutrition extend beyond individual diet planning. The algorithm can be utilized in menu optimization for institutions such as hospitals, schools, and cafeterias, where providing nutritious meals at a reasonable cost is essential. The algorithm can be also adapted in the future to accommodate specifically dietary requirements such as vegetarian, vegan, gluten-free among other options.

Further research and refinement of the algorithm could lead to enhanced optimization techniques and broader implementation in various domains, ultimately promoting healthier (nutritionally) and more cost-effective dietary practices.