NOVA IMS Information Management School

The Stigler's Diet Problem

Problem Optimization Using Genetic Algorithms

MSc. DATA SCIENCE AND ADVANCED ANALYTICS

Course: Computational Intelligence for Optimization

Professors: Leonardo Vanneschi | Berfin Sakallioglu

Group 14:

Henrique Marques (20220585)

Martim Santos (20220383)

Vasco Bargas (20220583)

Repository location:
https://github.com/henrymmarques/CI4O

May 2023

NOVA Information Management School
Instituto Superior de Estatística e Gestão de Informação
Universidade Nova de Lisboa

1. Introduction

In this project, we addressed the Stigler diet problem using genetic algorithms (GAs). GAs are optimization techniques inspired by natural selection, and they have shown great effectiveness in solving complex problems. The Stigler diet problem aims to find the most economical combination of food items that meet specific nutritional requirements. Our objective was to develop a GA-based solution that could efficiently explore the solution space and provide high-quality solutions. This report presents our approach, parameter configurations, and the analysis of experimental results, using as a baseline the *“Charles”* library developed in the classes.

2. Methodology

For the realization of this project, we used the library developed in class called *'charles'* where we made some changes to be adapted to our project. We tested the following selection methods, crossover and mutation:

Genetic Operation	Method Names
Selection Methods	1. Rank Selection
	2. Tournament Selection
	3. FPS
Crossover Methods	1. Arithmetic Crossover
	2. SBX
	3. Two-point Crossover
	4. Single-point Crossover
Mutation Methods	1. Swap Mutation
	2. Uniform Mutation
	3. Inversion Mutation

Table 1: Method used.

3. Fitness Function and Representation

The Stigler’s Diet Problem has a pretty straightforward objective: fulfil the minimum nutritional requirements while spending the least amount of money possible. To solve this problem, we have a table of 77 different foods and its correspondent nutritional values. The nutritional value of each food equates to 1 dollar worth of that food, meaning 1 dollar of a particular food will add to our diet the exact nutrients represented in the table. Since the objective is to find the lowest cost of our diet while complying with some constraints, the Stigler’s Diet is a **minimization problem**.

To address the problem, a **fitness function** was implemented to evaluate the individuals' fitness, with these individuals representing different combinations of money spent on different sets of food items. The fitness function initializes variables for cost, nutritional values, and a control parameter called "all_satisfied," which is initially set as "True". This parameter tracks if an individual fails to meet any nutritional requirements and will then be set to “False” if an individual does not meet any of the established nutritional requirements. After this step, the function iterates over the individual's representation and calculates their nutritional values and total cost based on the amount of money spent on each food item. In the end, if all nutritional requirements are satisfied, the fitness of each individual is the total cost of the foods it has in its list. If the individual does not satisfy the minimum nutritional requirements, the fitness will be the total cost plus the difference between the minimum value of each non-satisfied nutrient and the actual value of the individual’s list (for example, if an individual had a total cost of 100 and all the nutritional requirements were satisfied except the protein intake, which was 30 grams lower than the minimum required, its fitness would be 100+30=130).

Every **individual’s representation** is a vector with size 77 (number of foods) and each position represents the amount of money spent in each food, accordingly to the index. To ensure diversity and exploration, we initialize each individual with five non-zero positions in the vector, indicating initial spending on distinct food items.

To generate an individual, we developed a function that was initialized by creating an empty set called "selected_indices". This set is then populated with a specified number (in our case, 5) of unique numbers between 0 and 76, with each of these numbers representing a different kind of food that will be included in the generated individual.

Subsequently, we iterate through the selected food indices and assign them with a random spending amount between 1 and 15 dollars to each of these food items. The selection of these specific parameters was a result of extensive experimentation and fine-tuning, aiming to achieve optimal values that would enhance the overall performance of the algorithm.

4. Selection

Despite having tested the implementation of three selection methods the ultimately chosen selection method in this algorithm was **tournament selection**, due to its’ effectiveness and better results. To get to this result multiple tests consisting of 30 epochs were conducted using the three different methods while holding all the other factors fixed. Each of these tests involved a population of 100 individuals and 100 generations but despite having made these rigorous attempts, the obtained results were always very similar, as it is possible to see in Table 2, where algorithms achieved similar performances, with tournament selection performing slightly better than the others. Also, in Figure 1 it is possible to see the result of 1 epoch.

Selection method (keeping the rest constant)	**30 epochs average fitness**
Tournament Selection	60.12
Rank Selection	62.32
FPS	62.86

Table 2: Selection methods results.

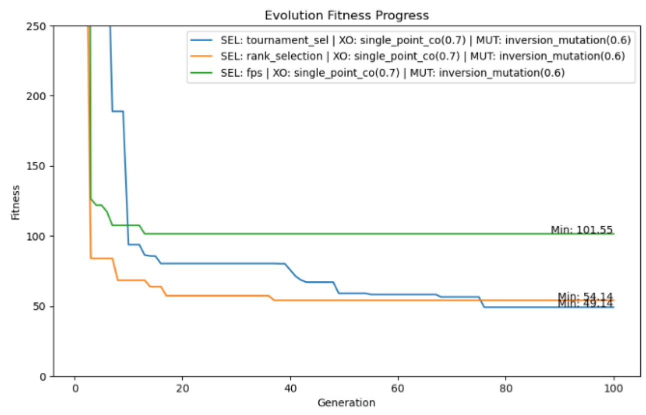


Figure 1: Selection methods plot.

Thus, to make a conclusive decision regarding the selection algorithm to be employed, further theoretical research was done. Tournament selection has been proved to be more effective than ranked selection and FPS in a variety of genetic algorithm problems. Hancock (1997)[1] states that "Freisleben and Härtfelder (1993) used a meta-level GA to tune the parameters of another GA. It was able to alter parameters such as population size, mutation rates and the selection method. The choice was tournament, rank and two forms of FPS. Tournament was the clear winner." – another advantage of this selection algorithm it is also really easy and simple to implement as it only requires that random individuals are selected from the population and that their fitness gets compared.

4.1. Tournament Size

After deciding to choose the tournament selection as the selection algorithm, the next step was to decide the **tournament size**: to choose the tournament size we decided to create three different configurations and see how these configurations changed with different tournament sizes. Initially we started with the values of 5, 10 and 15, with 5 achieving better results than the others. Thus, further trials were done with values that were closer to the previous best one, with these new values being 2 and 7, with 2 showing a worst performance but with 7 showing a clear improvement and so it was decided to use 7 as the tournament size for all the subsequent experimentations, as we can see in Table 3.

Tournament Size	30 epochs average fitness
2	61.25
5	59.84
10	60.04
15	60.40
7	57.99

Table 3: Tournament size results

Having made a final decision regarding tournament selection, our subsequent objective was to determine which were the most suitable crossover and mutation algorithms.

5. Crossover and Mutation Combinations

At this stage we decided to run several configurations at the same time, covering all possible combinations between mutation and crossover (still without changing the probabilities). And through Table 4 we can see that in terms of crossover, the two that stand out

[1]https://doc.lagout.org/science/0_Computer%20Science/2_Algorithms/Practical%20Handbook%20of%20GENETIC%20ALGORITHMS%2C%20Volume%20II/ganf3.pdf

are Single-point and Two-point. Regarding the arithmetic and SBX algorithms, it was realized that these might not be suitable for our problem as they had terrible performances that lead to fitness values 20 times bigger than the ones obtained with single-point and two-point crossovers. From the graphs in figures 2 and 3, we can see great results from 1 epoch using the two best crossover methods and the different mutations.

Crossover	Mutation	30 epochs average fitness
Single Point	Inversion	61.24
Single Point	Uniform	59.21
Single Point	Swap	65.75
Two Point	Inversion	57.79
Two Point	Uniform	64.76
Two Point	Swap	65.86
Arithmetic	Inversion	1151.01
Arithmetic	Uniform	100.57
Arithmetic	Swap	377.59
SBX	Inversion	145.19
SBX	Uniform	110.27
SBX	Swap	148.64

Table 4: All possible combination results.

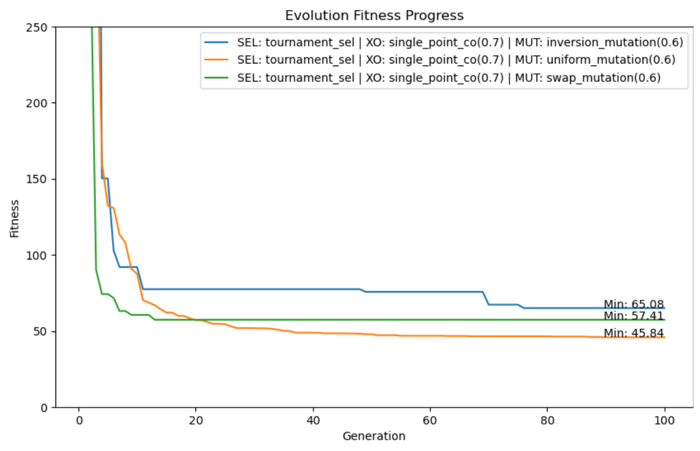


Figure 2: Single-point 1 epoch plot.

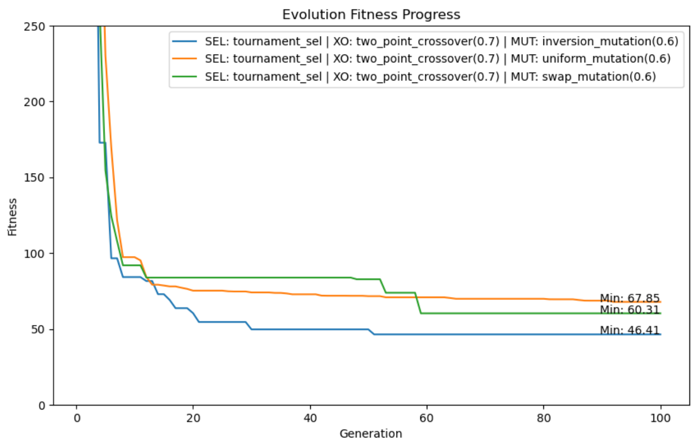


Figure 3: Two-point 1 epoch plot.

Although the values in table 4 for the 3 average fitness of the Single-point and Two-point crossovers are relatively close, through several tests and several plots, we could see that Single-point using uniform mutation converged faster to the final value than using another mutation and the same applied for Two-point with inversion mutation.

With these results, it was decided that we should stick with the two configurations highlighted in green in table 4 and run several tests with it changing the probabilities associated with its occurrence and with the occurrence of other factors.

6. Elitism impact in the GA results

Even though theoretically elitism is a technique that generally improves the performance of an algorithm, we decided to test some of our final configurations with elitism disabled, and as we can see in figure 4, the performance of our algorithms was worse when compared to the same algorithms using elitism.

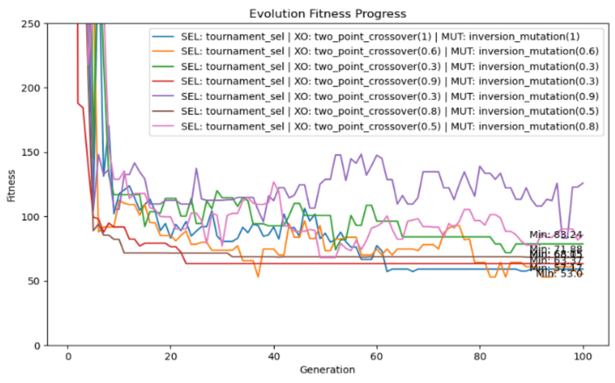


Figure 4: Configurations without Elitism.

7.Optimizing Mutation and Crossover Probabilities

The final step of the optimization of our algorithm was to define the best possible mutation and crossover probabilities. From early tests we knew these probabilities would have to be above 50 percent since we noticed that more constant changes would be beneficial to solve our problem. We performed 400 epochs with 3 different probabilities (0.5, 0.7, 1) while maintaining every other attribute the same. The results are shown in the table 5, where we can see that the probability of 1 for crossover and mutation results, on average, in the best fitness.

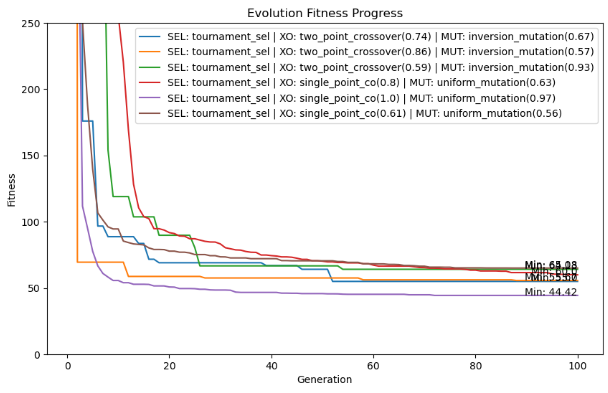
GA Configuration	Crossover Probability	Mutation Probability	400 epochs average fitness
Selection: Tournament	1	1	57.12
Crossover: Single-point	0.7	0.7	63.91
Mutation: Uniform	0.5	0.5	71.77

Table 5: 400 epochs average fitness results.

8. Stigler’s Problem Best Possible Result

To calculate the best result possible, we selected random probabilities between 0.5 and 1 for the crossover and mutation algorithms and used the two best configurations represented in Table 4. The best individual had a fitness of **<u>44.42</u>**, and was represented by this vector: *[9.119, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.426, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.936, 0.0, 0.0, 0.0, 0.0, 0.0, 2.829, 6.799, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 22.309, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0].* This means: 9.12$ of ***Wheat Flour (Enriched)***, 1.43$ of ***Rolled Oats***, 1.94$ of ***Green Beans***, 2.83$ of ***Potatoes***, 6.80$ of ***Spinach*** and 22.31$ of ***Navy Beans (dried).***

Even though the fitness of our best individual was not the global optimum of the Stigler’s Diet Problem, the difference in fitness between the global optimum and our individual was of about 5 (39.66[2] to 44.42).



[2] Reference result found here: https://developers.google.com/optimization/lp/stigler_diet.

Division of Labor:

All 3 of us worked equally on this project. The main ideas were discussed in group meetings and one of the members went ahead with the work as soon as they had time.