# CIFO Project

MASTER DEGREE PROGRAM IN DATA SCIENCE AND ADVANCED ANALYTICS

## The Stigler Diet Problem: Genetic Algorithm

Group 9

Tomás Bourdain, 2022154

Diogo Morgado, 20221392

GitHub

May, 2023

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

# INDEX

# 1. Introduction

Our project aims to use a Genetic Algorithm for the Stigler's Diet Problem, offering an approach to optimize this problem. In the following chapters, we will delve into the details of the implementation of our Genetic Algorithm. GitHub

## 1.1. Statement of Contribution

Tomás Bourdain: Fitness Function, Rank Selection, Swap Mutation, Plotted all Configurations.

Diogo Morgado: Function to run multiple configurations over 30 iterations.

# 2. Genetic Algorithm Configuration

## 2.1. Representation

In our implementation of our Genetic Algorithm for the Stigler's diet problem, we chose a specific representation for the individuals in the population. The representation is based on a binary encoding, where each bit corresponds to whether a particular commodity is included or excluded from the diet plan.

We thought that by using this binary representation, our GA can effectively explore the solution space by toggling the selection status of individual commodities. The fitness function evaluates the selected commodities based on their prices, weights, and nutrient content, ensuring that the solutions satisfy the constraints imposed by the problem. The evolutionary process, including selection, crossover, and mutation, allows the algorithm to generate new individuals and converge towards better solutions over successive generations.

## 2.2. Fitness Function

The fitness function underwent several iterations as we aimed to find the most effective approach. Initially, our focus was on minimizing the cost, and we attempted to achieve this by using the inverse of the cost, represented as 1/cost. However, we encountered significant challenges related to underflow problems with this approach. Then we tried a different approach, we tried maximizing the cost instead, but one major drawback of maximizing the cost was that the highest cost would be attained by including all available elements. This might present a problem since it disregarded the capacity and nutrient constraints that we might need to consider, to get a better fitness function.

After analysis of the other projects that were implemented in class, we tried an approach where we would calculate the total weight of items in a bag while simultaneously checking if the capacity and nutrient constraints were exceeded. By focusing on the weight of the items, we were able to incorporate the important notion of capacity into the fitness evaluation.

Our revised fitness function allowed us to strike a balance between maximizing the cost and considering the limitations imposed by capacity and nutrient constraints. By examining the total weight of the items, we ensured that the bag's capacity was not exceeded, and by extension, the nutrient constraints were maintained. We ultimately decided that this approach provided a more practical and realistic representation of the problem at hand, because it let us consider their cost, weight, and the constraints imposed.

## 2.3. Selection

In our project we have explored three different selection methods: Fitness Proportionate Selection (FPS). These selection methods play a crucial role in determining the genetic diversity and convergence of the population throughout the evolutionary process. One reason for selecting these particular selection methods was our familiarity with them, as they had already been implemented in classes for our project. This familiarity allowed us to leverage existing code and resources, making the implementation process easier and more efficient. Ultimately depends on the specific characteristics of the problem at hand, in this case, we decided to go with these, optimizing the performance and effectiveness of our genetic algorithm.

## 2.4. Mutation

After conducting experiments and evaluating the results we decided that the mutation methods we would use for our genetic algorithm, would be Inversion Mutation and Swap Mutation. Even though, we have 3 mutation methods implemented, we noticed after our experiments that binary mutation doesn’t meet our requirements, meaning that by carefully selecting the most suitable methods and excluding the ones that do not meet our requirements, we can optimize the genetic algorithm's ability to effectively search for optimal solutions in the given problem domain.

## 2.5. Crossover

We explored several crossover methods within the genetic algorithm: single point crossover, cycle crossover, partially matched/mapped crossover, and arithmetic crossover. After conducting experiments and evaluating their performance, we concluded that the most effective methods were arithmetic crossover, single point crossover and pmx (used with tournament selection). Based on our experiments and analysis, we found that arithmetic crossover and single point crossover consistently yielded better results compared to the other crossover methods we investigated. Therefore, we have chosen to focus on combining these two methods with the selection and mutation methods in our project to achieve the best possible outcomes.

## 2.6. Configurations

These were the configurations we tested, these were already after some tests that didn't work out and some configurations were removed.

Rank/Swap/Single

Rank/Inversion/Arithmetic

Rank/Swap/Arithmetic

Rank/Inversion/Single

Tournament/Swap/Single

Tournament/Swap/Arithmetic

Tournament/Inversion/Arithmetic

Tournament/Inversion/Single

Note: Elitism was always True, we found it gave better results.

# 3. Results and Conclusion

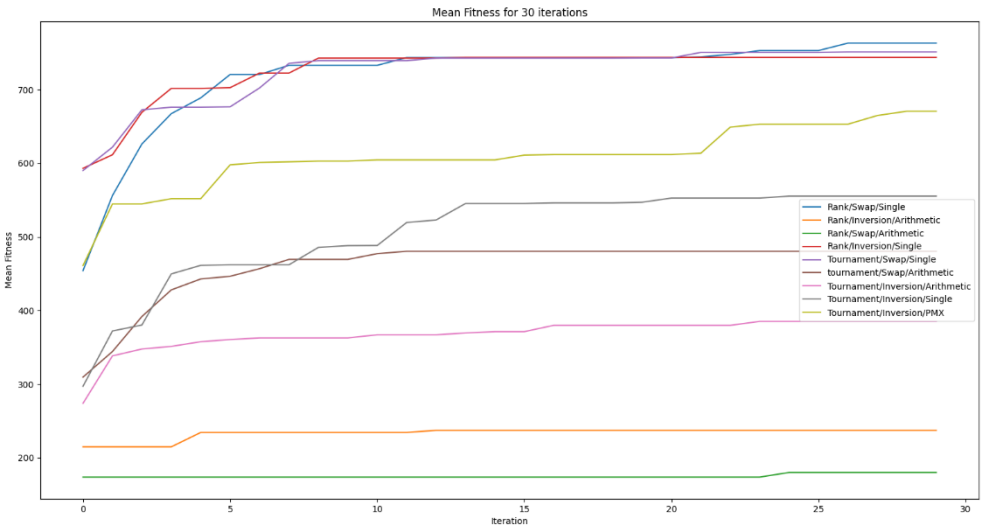


Figure 1 – Mean Fitness for Different Configurations over 30 Iterations

After looking at this graph and analyzing these different configurations over 30 iterations we can conclude that Rank/Swap/Single is the better one since it achieves the higher fitness, but some of the other configurations got close to it and some converged first to higher function than the best one, but in the long run Rank/Swap/Single was the best one.