Implementation of Genetic Algorithms to Solve Stigler's Diet Problem

20220662 Ariel Perez

20221015 Ehis Jegbefumwen

20221012 Ayotunde Aribo

20220661 Julio Vigueras

20220665 Miguelangel Mayuare

Group: Group 1

GitHub: https://github.com/mikemayuare/stiglers-diet-ga

Statement of Contribution:

Ariel Pérez: Power law, Gaussian and Sine mutations implementations, code revision, and project planning.

Ehis Jegbefumwen: Contribution to the implementation of fitness proportion selection and ranking selection operators, code review and drafting of the report

Ayotunde Aribo: Implementation of the tournament, fitness proportion selection and ranking selection operators, code revision, design and drafting of the report

J**ulio Vigueras**: Problem framing. Design of the representation and fitness function. Contribution to the revision and report content, implementation of the Entry-50:50 crossover, and correcting and revising the code. Design and implementation of the statistical test.

Miguelangel Mayuare: Implementation of the fitness function, crossover operators, grid search for models, fine-tuning of parameters, tweaking the Charles file to fit operators and selection methods for extra parameters.

1. Overview of the Project

The objective is to build the best Genetic algorithm for the Stigler's Diet Problem. To achieve it, several methods of selection, crossover, and mutation were implemented and compared. Thereafter, we tuned the parameters to obtain an efficient model capable of bringing good solutions to this optimization problem.

For the problem, each solution is represented as a vector (in Python, a list) in which each entry is a continuous value that indicates the amount of money that will have to be paid for each product each day. To do so, each entry is associated with an element of a products list (this list is normalized with respect to the cost of each product), maintaining the order, which contains the names and nutritional information of each food item. To retrieve the diet, each entry of the vector has to be multiplied with the value of its corresponding element of the list that indicates the units of the product (since each element of the list is normalized by its price, then the value of the units of the product indicates the quantity of the product that can be bought with one dollar). This representation was chosen because it facilitated the development of the code.

2. Fitness Function of the model

The fitness function was designed to calculate the total cost of the diet by summing the entries of the representation of the solution. The proposed solutions that do not meet the nutritional requirements are penalized by increasing their fitness' values. By doing this, the infeasible solutions are less likely to have descendants. This was the only fitness function that was implemented since it was very natural to define.

To evaluate the nutritional requirements, the total amount of each nutrient is calculated by multiplying the amount of each nutrient in each food item by the number of food items in the diet. Thus, we then compare the total amount of each nutrient in the diet to the minimum intake requirement for each nutrient. If the total amount of a nutrient in the diet is less than the minimum intake requirement, the function adds a penalty to the fitness value by multiplying the difference between the total amount of the nutrient in the diet and the minimum intake requirement by 100.

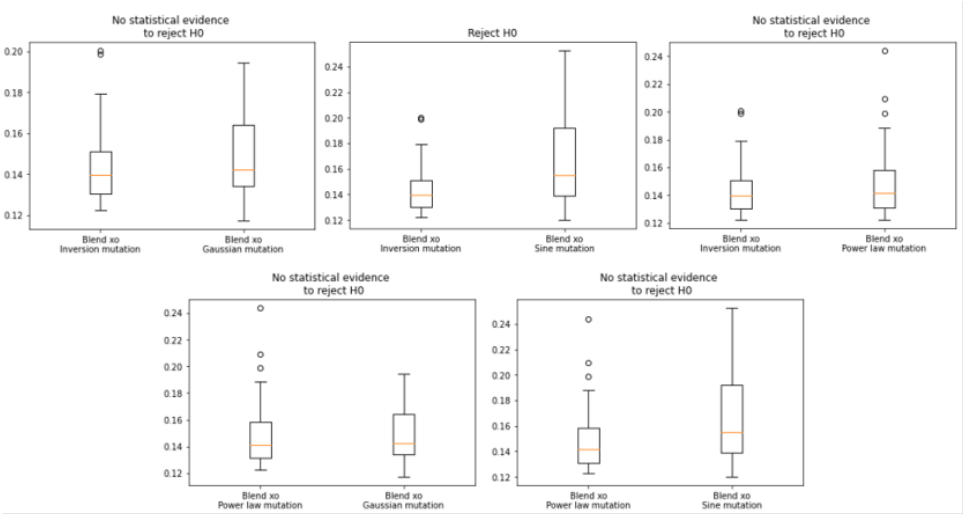
3. Selection Methods

The aim was to choose individuals that are likely to produce the fittest offspring. Considering the nature of our optimization problem and population, which contains genes represented by continuous values of floats, three selection algorithms were used in this study: Tournament selection, Ranking selection, and Fitness proportion selection (roulette wheel). It was decided to

keep Tournament selection because with this method we obtained the best solutions, while when using the others we obtained very bad results.

4. Determination of the configurations

As stated before, first we tried different selection methods, and the tournament method consistently yielded the best results. Then, to decide between different crossover and mutation operators, a grid was designed. This grid consisted on the combination of 7 crossover methods with 4 mutation methods, thus bringing 28 combinations. For each combination of crossover-mutation model, genetic algorithms were run 25 times, thus obtaining the fitness of the best individual in each run, and then applying the Mann-Whitney's test for each pair of models to determine if there exists statistical evidence that the two samples of solutions were from different populations or if there is no statistical evidence of this statement. After comparing the results, three models produced the best fitness: Blend xo as crossover and those that use Inversion mutation, Gaussian mutation or Power law mutation as mutation operator. Because of this, we decided to keep Blend xo crossover and Power law mutation, and afterwards tune its parameters.



5. Are the implementations abstract and work with both minimization and maximization?

No. To work with a maximization problem, some changes must be made to the code beforehand. For instance, the penalty to the infeasible solutions would have to be changed from summing a quantity to subtracting it.

6. Implementation of Elitism

We tested our algorithm with and without elitism and we got better results with elitism. The inclusion of elitism was a positive factor in our genetic algorithm because it helped to obtain the best solutions so far.

7. Parameter Tuning

Fig 1: Tuning of the tournament size: We ran the GA with different tournament sizes from 1 - 10 and found that the best tournament size was 4. The GA with tournament size 4 provided the lowest fitness value when compared to other selection parameters as shown in the image below.

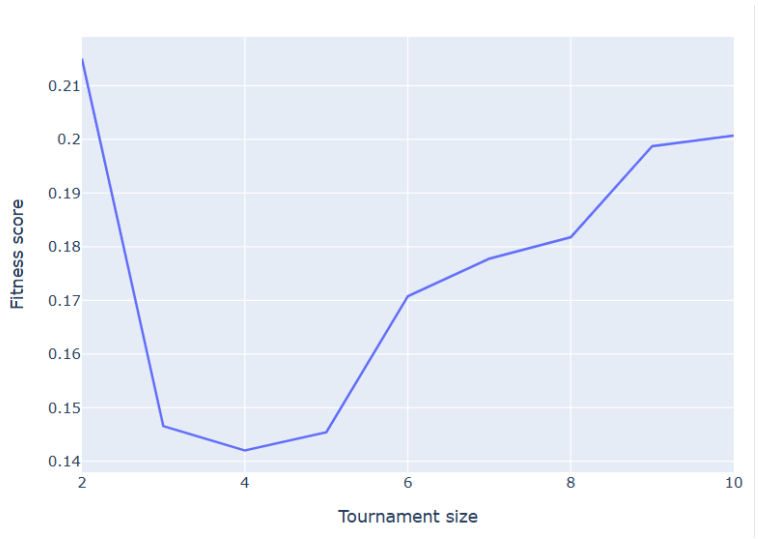


Fig 2: Similar fine-tuning was made for the blend crossover and power law mutation operators.

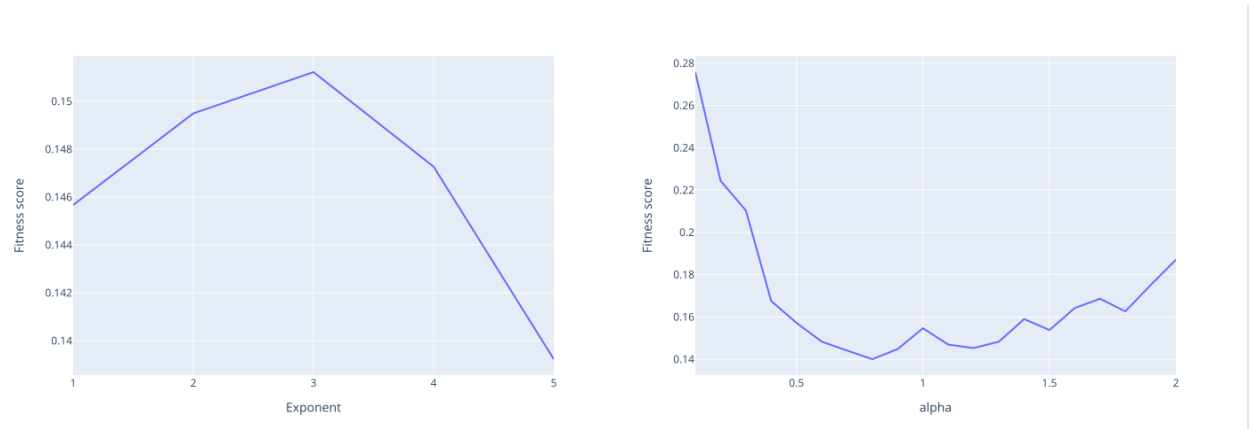
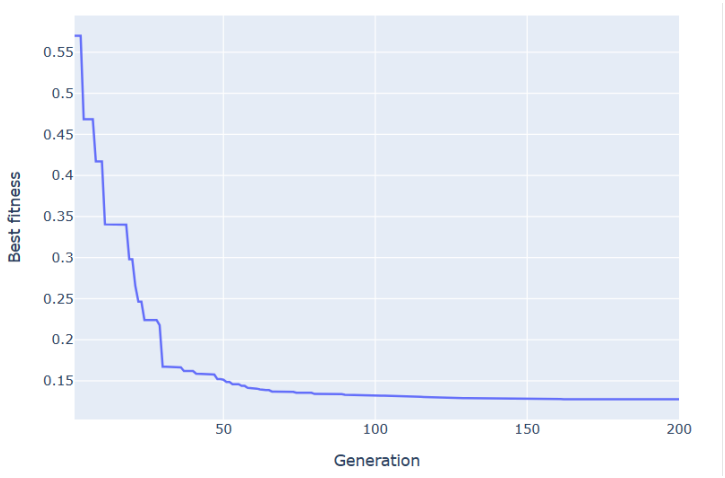


Fig 3: Evolution Performance – 200 Generations



8. Results and Discussions

We were pleased with the results. The chosen genetic algorithm model demonstrated a smooth convergence during the optimization process. As the algorithm iteratively evolved and refined its solutions, it consistently approached an optimal or near-optimal solution.

Nevertheless, if we wanted to improve it, it is possible that this could be achieved by implementing an algorithm that decides when to diminish (or increment) some parameters' values to refine the search for better solutions. Furthermore, in the end, it could be possible to apply a Local Search algorithm to the best solution found by Genetic Algorithms since the latter assures a very good solution, while the former assures obtaining a local optimum. Nonetheless, since this problem is treated with continuous solutions, the quality of the local optimum depends a lot on how the neighborhoods are defined. Therefore, the implementation of a Local Search does not completely guarantee a better solution.

It is important to mention that it is possible to add more, take out some, or change products of the list of food items, with their corresponding nutritional information, or to change only the values of the nutritional requirements, because the model is capable enough to adapt to all these changes and find a proper solution to the problem.

The best model that was built of Genetic Algorithms, which has the operators Selection: Tournament; Crossover: Blend; Mutation: Power law, obtained the lowest fitness value of 0.134

Moreover, for the data to which the model was applied, it is known that the global optimum has a value of $39.6617 (annual cost), while the best fitness value that we obtained is $46.82 (annual cost). Therefore, we must note that our model obtains very good results.

A typical output of the problem is:

```
Expenditure per day: 0.13$
Yearly expenditure: 46.82$


Calories (kcal):__3.38__
Protein (g):__146.45__
Calcium (g):__0.8__
Iron (mg):__58.06__
Vitamin A (KIU):__11.4__
Thiamine (mg):__4.33__
Riboflavin (mg):__2.7__
Niacin (mg):__30.24__
Ascorbic Acid (mg):__75.0__
Wheat Flour (Enriched):__0.04$__
Sweet Potatoes:__0.04$__
Navy Beans, Dried:__0.05$__
```

9. Conclusions

In this project, the objective was to develop a genetic algorithm for solving Stigler's Diet Problem. Various methods of selection, crossover, and mutation were implemented and compared. The chosen model used Tournament selection, Blend crossover and Power law mutation, achieving a fitness value of 0.134 (best solution between its corresponding 25 runs), thus indicating its effectiveness. Besides, elitism was included in the algorithm and proved to be beneficial. Parameter tuning was performed, and the optimal tournament size was found to be 4. The algorithm demonstrated smooth convergence towards an optimal or near-optimal solution. While further improvements are possible, such as dynamic parameter adjustment and applying Local Search, the model is adaptable to changes in food items and nutritional requirements. Overall, the project successfully developed an efficient genetic algorithm for the Stigler's Diet Problem.