Computational Intelligence for Optimization Project

MASTER DEGREE PROGRAM IN DATA SCIENCE AND ADVANCED ANALYTICS

STIGLER'S DIET PROBLEM

Group 7

Bruna Oliveira F. de Araújo, number: 20220587

Inês Nascimento, number: r20170746

Janaina Santos, number: 20220640

Lucas Ferreira, number: 20220621

May, 2023

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

Access link to the repository: https://github.com/lucasemiliomf/cifo_group7

1. Project's assumptions

The cost optimization calculation was done on an annual basis (365 days) instead of a daily basis, this premise was adopted in order to make the results more consistent. If only daily consumption was considered, it would be possible to have solutions with just one ingredient that already satisfied all nutritional daily requirements. Two other assumptions adopted were that the solution could have a maximum of 3 equal foods and that the number of foods would always be an integer value.

2. Optimization Methodology

In order to achieve good results, tests were carried out with different combinations of selection, crossover and mutation methods. To perform selection, two different methods were used: fitness proportionate selection (FPS) and tournament selection. For crossover, single point crossover, uniform crossover and indexed cycle crossover methods were used. For mutation, the methods used were quaternary, swap and inversion mutation.

3. Code modifications

For this project, some modifications were made in certain functions from the Charles project provided by the professors in order to adapt them to the Stigler's Diet Problem.

3.1 get_values

Concerning the "get_values" function that gives the total price and nutrients values for an individual by adding the requested amount with price and nutrients values, respectively. As a result a list is returned with the total cost, calories, protein, calcium, iron, vitamin A, vitamin B1, vitamin B2, niacin, and vitamin C. Also, it is important to notice that the quantity required for a dish is iterated over a loop that multiplies the amount with the respective price and nutrients values and keeps the accumulated values from that multiplication. Consequently, this function returns the price (fitness) and all macronutrients for each individual.

3.2 get_fitness

Regarding the “get_fitness” function, for each individual, the price (fitness) of the set of values is calculated to achieve the amount of nutrients needed at the lowest possible price. This calculation is done by going through the string of values and multiplying the amount for the price of each individual. The objective is to optimize the total price (fitness), but the solution needs to satisfy the minimum amount of macronutrients. For that, each time one condition is not satisfied, a punishment proportional to the missing amount is added to fitness, since it is a minimization problem.

3.3 execute_ga

In order to get the best methods for selection, crossover and mutation, the execute_ga function was developed. It receives 3 parameters, defining each of the methods, to perform a grid search with those parameters and define the best combination. The other parameters (population size, valid set, mutation probability, crossover probability and elitism), were decided empirically by testing in a different file (defining_parameters.py). The values chosen will be explained on the result part below.

3.4 get_elite

The get_elite is a method implemented inside the class Population on Charles.py. This was implemented to obtain the best individual (or elite) in each population. Since there were multiple instances requiring this individual, it was better to implement this method.

3.5 fps

In the selection.py file the fps function was adapted for a minimization problem. It works similar to the maximization problem, but the fitness is inverted (since smaller fitness will have a bigger inverse).

3.6 indexes_cycle_xo

The original cycle_xo function only accepted strings with different bits. Therefore, the indexes_cycle_xo function was created to get around this issue, indexing the bits and doing the crossover in the indexes and then applying the crossover to the individual itself.

3.7 uniform_xo

The uniform_xo function evaluates the string bit by bit and chooses, with a certain probability, the chance of getting the bit from parent 1 or parent 2. For this project, a probability of 50% was implemented.

3.8 quaternary_mutation

Additionally, this function is an adaptation of binary_mutation, since for this project values between 0 and 3 are accepted. Also, this type of mutation gives diversity to the population so that it enables the genetic algorithm optimization process to explore various solutions.

4. Results

4.1 Best parameters

The parameters were tested independently in the defining_parameters.py file. The values chosen and used to perform the search over the methods in the optimization were the following:

- Population size: 300 (bigger population provided better results in the test, but it was limited to 300 in tests)
- Quantity range of each food: [0, 1, 2, 3] (it was decided as an extension for the binary approach used in class, but limited to quaternary)
- Generation number: 100 (it was limited to 100 to limit time)
- Crossover probability: 90% (best result obtained)
- Mutation probability: 20% (best result obtained)
- Elitism: False (to keep diversity in the population)

The average fitness results after 30 executions can be seen on table 1, for the 18 combinations, from the previously explained methods, to perform selection, mutation and crossover.

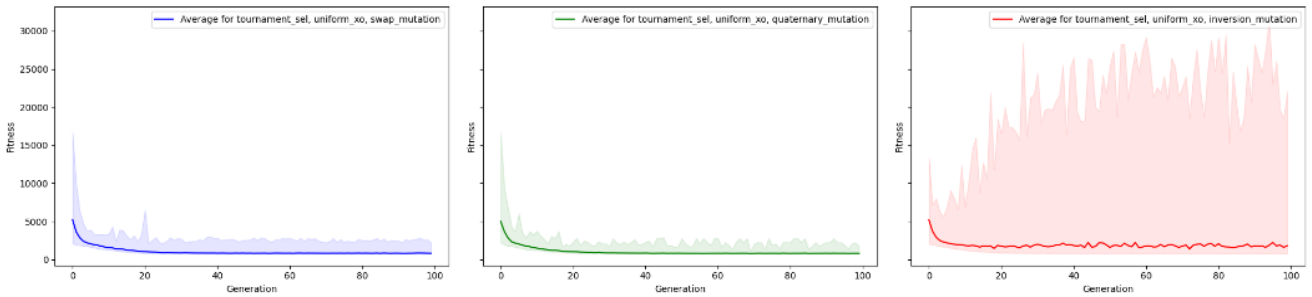
Table 1: Fitness for methods combinations

selection_method	mutation_method	crossover_method	average_fitness
tournament_sel	swap_mutation	uniform_xo	R$ 764,85
tournament_sel	quaternary_mutation	uniform_xo	R$ 769,04
tournament_sel	inversion_mutation	uniform_xo	R$ 772,66
tournament_sel	swap_mutation	single_point_co	R$ 812,99
tournament_sel	quaternary_mutation	single_point_co	R$ 860,09
tournament_sel	inversion_mutation	single_point_co	R$ 921,02
tournament_sel	quaternary_mutation	indexes_cycle_xo	R$ 1.077,87
fps	quaternary_mutation	uniform_xo	R$ 1.252,91
fps	swap_mutation	uniform_xo	R$ 1.272,53
fps	quaternary_mutation	single_point_co	R$ 1.317,79
fps	swap_mutation	single_point_co	R$ 1.378,55
tournament_sel	swap_mutation	indexes_cycle_xo	R$ 1.492,30
tournament_sel	inversion_mutation	indexes_cycle_xo	R$ 1.576,35
fps	inversion_mutation	single_point_co	R$ 1.580,87
fps	quaternary_mutation	indexes_cycle_xo	R$ 1.829,22
fps	inversion_mutation	uniform_xo	R$ 1.837,39
fps	inversion_mutation	indexes_cycle_xo	R$ 1.857,94
fps	swap_mutation	indexes_cycle_xo	R$ 1.877,70

As it can be evaluated, the best result was given by tournament selection, inversion mutation and uniform crossover with the average fitness of R$ 764,85, as it was performed with the goal to minimize the value needed to be spent on the diet.

In order to check the minimum, maximum and average statistics over the 100 generations for all 30 executions, the 3 best combinations (minimum fitness values), ordered from the best to worse, are shown in Figure 1. Thus, it can be observed that in the first combination that the conversion was faster and stable, while in the third combination the variance of results was large and more unstable.

Figure 1: Average Fitness Over Generations for Top 3 Combinations



4.2 Best individual

In the table below, we can find the best individual obtained in an execution of the code.

Table 2: Best Individual

Ingredient	Quantity	price (cents)	calories (kcal)	protein (g)	calcium (g)	iron (mg)	vitaminA (KIU)	vitaminB1 (mg)	vitaminB2 (mg)	niacin (mg)	vitaminC (mg)
Wheat Flour (Enriched)	3	108	134,1	4233	6	1095	0	166,2	99,9	1323	0
Corn Meal	3	13,8	108	2691	5,1	297	92,7	52,2	23,7	318	0
Hominy Grits	3	25,5	85,8	2040	2,4	240	0	31,8	4,8	330	0
Rice	3	22,5	63,6	1380	1,8	123	0	6	14,4	180	0
Rolled Oats	3	21,3	75,9	2721	15,3	1023	0	111,3	26,7	192	0
White Bread (Enriched)	3	23,7	45	1464	7,5	345	0	41,4	25,5	378	0
Whole Wheat Bread	3	27,3	36,6	1452	8,1	375	0	41,7	19,2	480	0
Milk	3	33	18,3	930	31,5	54	50,4	12	48	21	531
Evaporated Milk (can)	3	20,1	25,2	1266	45,3	27	78	9	70,5	33	180
Peanut Butter	3	53,7	47,1	1983	3	144	0	28,8	24,3	1413	0
Lard	1	9,8	41,7	0	0	0	0,2	0	0,5	5	0
Liver (Beef)	3	80,4	6,6	999	0,6	417	507,6	19,2	152,4	948	1575
Apples	3	13,2	17,4	81	1,5	108	21,9	10,8	8,1	15	1632
Bananas	3	18,3	14,7	180	1,2	90	52,2	7,5	10,5	84	1494
Green Beans	3	21,3	7,2	414	11,1	240	207	12,9	17,4	111	2586
Cabbage	3	11,1	7,8	375	12	108	21,6	27	13,5	78	16107
Carrots	3	14,1	8,1	219	8,4	129	565,5	18,3	12,9	267	1824
Onions	3	10,8	17,4	498	11,4	177	49,8	14,1	17,7	63	3552
Spinach	3	24,3	3,3	318	0	414	2755,2	17,1	41,4	99	8265
Sweet Potatoes	3	15,3	28,8	414	8,1	162	872,1	25,2	16,2	249	5736
Pork and Beans (can)	3	21,3	22,5	1092	12	402	10,5	24,9	23,1	168	0
Prunes, Dried	3	27	38,4	297	7,5	462	257,1	11,7	12,9	195	771
Peas, Dried	3	23,7	60	4101	12,6	1035	8,7	86,1	55,2	486	0
Lima Beans, Dried	3	26,7	52,2	3165	11,1	1377	15,3	80,7	114,6	279	0
Navy Beans, Dried	3	17,7	80,7	5073	34,2	2376	0	115,2	73,8	651	0
Cocoa	3	25,8	26,1	711	9	216	0	6	35,7	120	0
Molasses	3	40,8	27	0	30,9	732	0	5,7	22,5	438	0
Total:	**79**	**750,5**	**1099,5**	**38097**	**297,6**	**12168**	**5565,8**	**982,8**	**985,4**	**8924**	**44253**
Required			**1095**	**25550**	**292**	**4380**	**1825**	**657**	**985,5**	**6570**	**27375**
Difference			**4,5**	**12547**	**5,6**	**7788**	**3740,8**	**325,8**	**-0,1**	**2354**	**16878**

The best individual cost was approximately 751 cents. In the total row, it can be observed the total amount of each nutrient for this set of foods in a given year. In the required row, it is shown the total amount of each nutrient required for a person over a year.

It can be observed that only Vitamin B2 from the nutrients requirement was not met. Since the difference was only 0.1 mg over a year, it was decided to keep this solution as the best.

As the code approves, individuals that do not meet the nutrient constraints, this type of solution can be found, although it could be approached over other techniques to avoid this.

For other nutrients, some of the constraints were reached by a large amount. Since there was no maximum amount constraint, the solution can be accepted. However, there can be issues for a person to eat in excess of some nutrients.

5. Conclusion

All group members contributed equally to the project. Firstly, the group met to choose the theme and then began planning the adaptations to the class codes to build the optimization. Finally, the construction of the report and its subsequent revision were made.

For future studies, what can be done differently in the project is, firstly, try another valid set. Since individuals could only have bits of integer values between 0 and 3, the solution is partially restricted by those constraints. Approaching the solution with float numbers could improve the results and also bigger amounts than 3.

Another approach that can be done is define the hyperparameters together in a grid search. To save time and resources, most of the parameters were defined before, doing the search only on the selection, crossover and mutation methods. Since adding parameters in the search the time of execution is multiplied, this approach would be costly to be done.

It could also be implemented using some different methods in the algorithm. There are multiple functions for crossover and mutation not implemented in class and ranking selection was not approached for the code. Also, elitism was only approached in a true or false situation, but it can be implemented with different sizes of individuals to carry on elitism.

Finally, the fitness could be calculated differently. The weights of the punishment for constraints that are not met could be changed to improve the solution. Also, there could be other ways to define fitness in the problem.

In conclusion, the problem was well solved in the terms defined. Over the generations, fitness is constantly improving. Adapting the code to have the changes above could achieve even better results, especially considering that some nutrients are way above the requirements, meaning there could be some savings on those parameters.