The McDonald's Diet Problem

An adaptation of the Stigler Diet Problem

David Halder (20220632), Felix Gayer (20220320), Lukas Stark (20220626) & Jan-Paul Briem (20222214)

Group: Group 2

GitHub Repository

Table of Contents:

Contribution Statement

Initially, each team member independently developed their own solution, and their contributions were later combined to create the final outcome of the project.

David Halder conducted extensive research and development on mutation functions, designed the report structure, and created result visualizations.

Lukas Stark played a pivotal role by developing visualization functions, structuring and developing experiments, and contributing to algorithm development.

Jan-Paul Briem contributed by developing the class architecture of the genetic algorithm and hyperparameter tuning as well as assisting with the report.

Felix Gayer focused on the research and development of crossover functions as well as the selection algorithms.

I. Introduction

The Stigler Diet Problem describes a mathematical way to solve basic nutritional needs in an inexpensive way. Consequently, it is about minimizing the total price under certain conditions. While this topic is still very relevant, the problem framework is slightly outdated. This project aims to provide a new and modern approach on the problem using openly available data from the fast-food chain "McDonalds" to find the cheapest constellation of items on the menu, which can fulfill daily nutritional needs.

The data set consists of 91 observations, which are the different items on the menu and 8 variables, including price, calories, total fat, cholesterol, sodium, carbohydrates and protein. The items are mixed, which means there is no differentiation between breakfast or other items. Drinks are excluded for this project. The sources for the nutritional needs and the McDonalds data can be found in the bibliography.

The minimum daily nutrients which need to be met are:

Calories (kcal):	*1.670*
Total Fat (g):	*60*
Sodium (mg):	*3.820*
Carbohydrates (g):	*220*
Protein (g):	*70*

It is important to mention that this project by no means aims to advise on a healthy way of nutrition. Most of the items on the menu are very calorie dense and high in fat and sodium, which can also be seen in the results later on. Thus, this project solely focuses on the optimization exercise not on the nutritional value of the final list of items.

This report is structured as follows. First, the produced algorithm and its operators are stated and explained. Next the experiments conducted are presented, explained and compared. In particular it is reviewed how the different selection, crossover and mutation methods affect the performance of the genetic algorithm. Lastly, the final algorithm and results are presented and critically examined.

II. Algorithm

For this problem a binary representation of the products was chosen, i.e., every solution is represented by a vector with 91 elements and if an item is part of the solution its corresponding value is "1" and otherwise "0". It was decided to work with this binary representation to ensure a very high diversity of products in the solutions by avoiding item duplicates. Besides, this representation allows the usage of matrix calculation methods to get an individual's fitness, which brings performance benefits. The definition of the representation can be found in the file "*individual.py*".

For the fitness function of the algorithm different versions were tried. The main variable for calculating the fitness is the cumulated cost of the solution, which is aimed to be minimized in this project. The second part of the fitness function is concerned about the constraints imposed by the nutritional values that need to be met. The first version of

the fitness function imposed a very high additional price on the Individual, exacerbating its fitness even if only one constraint is broken. This had the consequence that solutions, which were only slightly above a single constraint, had only a very small chance to get selected, even though crossover and mutation could have evolved these individuals to fit the restrictions to represent a viable solution. Thus, it was decided to change the penalizing logic. The final version of the fitness function puts a penalty of ten extra euros on the fitness of an individual for every constraint broken. That way, it is ensured that individuals which break a lot of constraints get a lower chance of being selected than individuals which break only a few. Consequently, the possible penalty range is between ten to fifty USD, which was selected taking into consideration the minimum, maximum and the average item price of 0.88 USD, 13.35 USD and 4.51 USD, as well as experiments in the beginning of the algorithm development. Apart from the fitness value, the implemented function returns the total costs as well as total nutritional values of the respective individual to store them for evaluation purposes and avoid repeating calculations. The code for the final fitness function can be found in the file "*fitness.py*".

Initially, a population of random individuals is generated, and corresponding fitness values are calculated. This population is then iteratively evolved over the generations, while the implemented function "*evolve()*" of the "Population" class allows to specify whether a minimization or maximization is aimed. To get the new population of the next generation, an iterative process is executed, where in every iteration two individuals get selected for crossover and mutation methods to add the resulting offspring to the new population, until the new population is as large as the current one. The following three different selection methods were implemented: Fitness proportionate, single tournament and ranking selection. To not break the scope of this report the methods discussed in class will not be further explained. The different selection methods can be found in the file "*selection.py*".

The chosen parent individuals have a probability to be combined by one of four crossover methods: Uniform, one-point, five-point and ten-point crossover. The offspring then have the chance to be mutated. The mutation methods implemented include single bit flip, complete bit flip, swap, multiple bit flip and scramble mutation. The complete bit flip method switches every single bit in the individual, which means every "1" becomes a "0" and every "0" becomes a "1". The multiple bit flip works in a similar way, but this method only flips a preassigned number of randomly chosen bits. Finally, a scramble mutation was implemented. For this mutation an arbitrarily sized part of the individual is chosen and the bits within it are shuffled. The crossover as well as the mutation methods can be found in the file "*variation.py*". The mutation is the last step before adding the offspring to the new population of the next generation.

To ensure that the best fitting individual of the population of the previous generation is also present in the next one, the implemented genetic algorithm has the option to enable elitism, which is enabled by default to not lose the best individual. In detail, the best fitting individual from the old generation is compared to the worst fitting individual of the new generation and the better one stays in the new population.

III. Experiments

Next, we present the experimental evaluation and parameter tuning process for optimizing the genetic algorithm used to solve the McDonald's diet problem. The aim was to understand the impact of different approaches and to find the

best configuration of the genetic algorithm, including the previously presented methods for selection, crossover and mutation as well as corresponding hyperparameters, that would result in improved performance and convergence towards the optimal solution.

A Jupyter Notebook was utilized for conducting the experiments and comparing the results. To ensure statistical significance and reliability, we performed 30 independent iterations for each genetic algorithm variant. The evaluation and comparison were based on the average fitness and standard deviation metrics. Additionally, various visualizations were employed, including plots of all (30) iterations, the most commonly selected products, a table displaying the best solution/individual, and nutrition curves.

Throughout the experiments, certain parameters remained fixed to maintain consistency. The fixed parameters included "Generations per Run/Iteration=30" and "Population Size = 500". Additionally, due to the probabilistic characteristic of genetic algorithms it is reasonable to set the parameter "Elitism" to be true to consistently obtain the best solution by not losing the best solution ever found across the generations.

The evaluation process focused on analyzing one parameter at a time while keeping other factors constant (ceteris paribus). First, different selection algorithms were tested, comparing their performance. The initial algorithm utilized fitness proportion selection (roulette), single-bit flip mutation, and one-point crossover. However, we observed that this algorithm did not converge within the predefined 30 generations. The average costs/fitness for this algorithm were 104.054 USD, suggesting that more generations might be needed. Furthermore, there was a high standard deviation of 8.7883 USD among the runs, indicating instability.

To address the convergence issue, we investigated alternative selection algorithms. By switching from fitness proportion selection (roulette) to ranking selection, we observed a significant improvement in results. The average fitness/costs decreased to 14.8613 USD, and the standard deviation of fitness also decreased significantly to 1.3221 USD. The fitness curves demonstrated a faster convergence rate with ranking selection.

We further enhanced the convergence speed by adopting the tournament selection algorithm with a tournament size of 5. The average fitness decreased again to 13.0637 USD as well as the standard deviation to 0.9211 USD. Notably, this combination of parameters led to the first instance of reaching the so-believed global optima, with the best possible solution identified in all the experiments as 11.25 USD. Based on these findings, we selected the tournament selection algorithm for the remaining experiments.

Next, we focused on modifying the crossover method. Transitioning from one-point crossover to five-point crossover improved the average fitness/costs to 11.6437 USD and reduced the standard deviation of fitness to 0.3537 USD. Although we observed further improvements with ten-point crossover, it was the final method "uniform crossover" that yielded the best results keeping all other parameters fixed (avg. fitness 11.4747 USD, std. ± 0.2186 USD).

Finally, we evaluated different mutation algorithms, but no major differences were observed. It is important to state, that the Single Swap Mutation is by no means optimal for the chosen representation, since the swap has a fairly high probability of not changing the Individual (swaps 0 with 0 or 1 with 1). All configurations reached the so-believed global optima (11.25 USD), although the Multiple-Bit-Flip mutation algorithm, in run 9, achieved the minimum average cost/fitness of 11.4473 USD and a standard deviation of 0.1836 USD. A summary of the experiments and their results can be found in Table 1.

Until now, we used for several hyperparameters guessed values based on our problem domain, the respective properties of each hyperparameter and corresponding best practices. Therefore, we applied a simple Grid-Search to further optimize the genetic algorithm for our problem. Due to hardware limitations we had to decrease the iterations per model configuration to 20. The examined hyperparameters encompassed the population size, the tournament size of the selection method, the crossover probability, the mutation probability, and the number of bits flipped within the multiple bit flip mutation method. After the tuning process, we were able to improve the performance of our best genetic algorithm to an average fitness/cost of 11.3055 USD with a standard deviation of 0.0963 USD. One major driver for the improvement is the increased population size, which is expectable, but it is related with longer processing times. Moreover, the mutation probability was decreased from *0.2 to 0.1* throughout the tuning, which is also reasonable because the representation of a good solution for our optimization problem has many "0" values, i.e. there is a high chance of adding a new product to the solution leading to a worse fitness. Finally, the crossover probability was increased by 5% from *0.9 to 0.95* increasing exploration and diversity, supporting the improved result.

Results of the experiments		Generations per Iteration	Population Size	Selection	Crossover	Mutation	Elitism	Average Fitness (USD)	Standard Deviation (USD)
Selection Experiments	Trial 1	30	500	**Roulette**	One-point	Single-Bit-Flip	TRUE	104.054	8.7883
	Trial 2	30	500	**Ranked**	One-point	Single-Bit-Flip	TRUE	14.861	1.3221
	Trial 3	30	500	**Tournament (k=5)**	**One-point**	Single-Bit-Flip	TRUE	13.064	0.9211
Crossover Experiments	Trial 4	30	500	Tournament (k=5)	**Five-point**	Single-Bit-Flip	TRUE	11.644	0.3537
	Trial 5	30	500	Tournament (k=5)	**Ten Point**	Single-Bit-Flip	TRUE	11.518	0.2141
	Trial 6	30	500	Tournament (k=5)	**Uniform**	**Single-Bit-Flip**	TRUE	11.475	0.2186
Mutation Experiments	Trial 7	30	500	Tournament (k=5)	Uniform	**Complete-Bit-Flip**	TRUE	11.578	0.2547
	Trial 8	30	500	Tournament (k=5)	Uniform	**Single-Swap-Mutation**	TRUE	11.470	0.2152
	Trial 9	30	500	Tournament (k=5)	Uniform	**Multiple_Bit_Flip (flips = 5)**	TRUE	11.447	0.1836
	Trial 10	30	500	Tournament (k=5)	Uniform	**Scramble Mutation**	TRUE	11.455	0.1973
Hyperparameter Tuning	Final Run	30	**750**	**Tournament (k=4)**	Uniform	Multiple-Bit-Flip (flips = 5)	TRUE	11.348	0.1414

Table 1: Experiments with the respective results.

It is important to note that due to the inherent randomness involved in the genetic algorithm, the exact scores obtained from the conducted runs cannot be reproduced precisely. However, the chosen parameters and their corresponding performance improvements provide valuable insights for solving real-world diet optimization problems.

Figure 2 shows the results of the first and final run. For the final run we used the parameters of the best configuration of the hyperparameter tuning across 30 iterations, which this time led to a slightly worse average fitness (11.3477 USD and standard deviation (0.1414 USD). The fitness curves clearly show the small standard deviation and the fast convergence of the runs. However, what can be seen from the final list of items and the nutritional curves is that this solution is by no means healthy. Especially the calories and the total fat fail to approach the minimum value given.

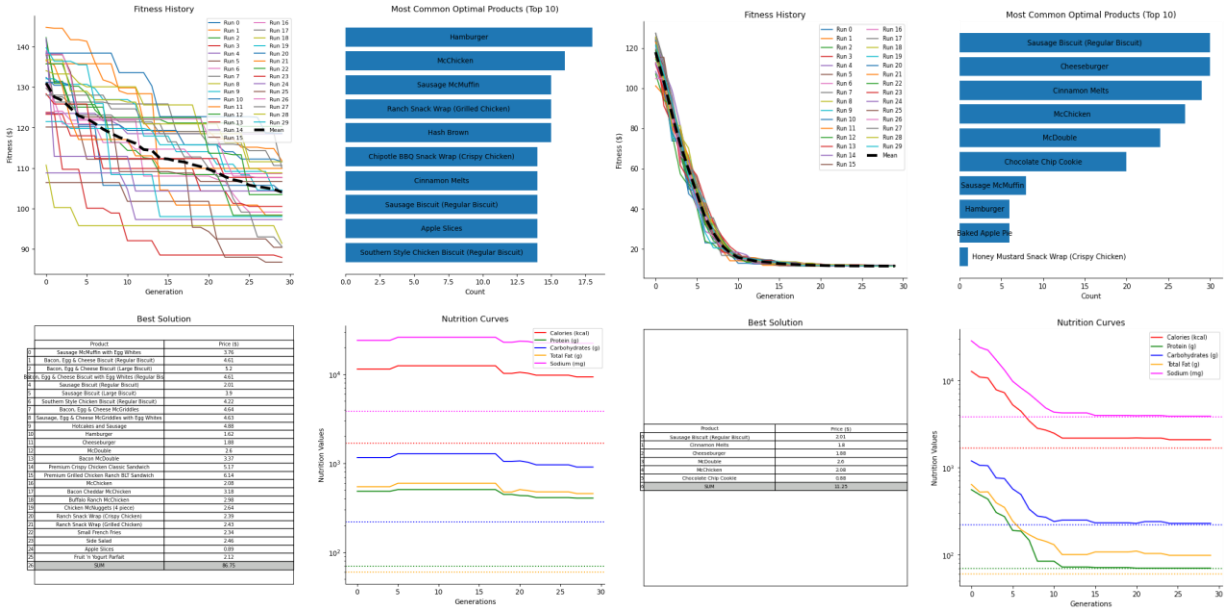

*Figure 1 (left): Fitness History, Most Common Items, Best Solution and Nutritional Curves from the **first run.***

*Figure 2 (right): Fitness History, Most Common Items, Best Solution and Nutritional Curves from the **final run.***

IV. Conclusion

This project aimed to find the most affordable way to meet one's daily nutritional needs, consisting only of items from the fast-food restaurant McDonald's. To do so a genetic algorithm which entailed the development of an appropriate representation and a fitness function as well as selection, crossover and mutation algorithms was developed.

To find the optimal combination of genetic operators and to assess their impact, ten different experiments were conducted changing only one operator at a time. These experiments concluded that the combination of the Tournament Selection, the Uniform Crossover and the Multiple Bit Flip Mutation performs best. The results were further optimized through hyperparameter tuning, reaching an average fitness of 11.35 USD with a standard deviation of 0.14 USD. This algorithm led to a solution containing six different items, which together fulfill the set nutritional requirements.

Future projects could involve adding more items or additional constraints to the dataset. It would also make sense to apply the developed algorithm to a healthier and more diverse set of foods. Also, the algorithm could be further improved. For example, an even more sophisticated penalty could be implemented for breaking constraints, incorporating the distance to the minimum value. This could have a positive impact on the efficiency of the convergence and lead to a more precise selection. To promote diversity fitness sharing could additionally be implemented, preventing dominance of a particular solution and thus promoting the exploration of the search space.

In conclusion, the usage of genetic algorithms led to the solution of the McDonalds Diet problem proposing a menu of six items to meet daily nutritional requirements for a price of 11.25 USD. Once again, the team does not advise following this diet since it clearly is not a healthy nor a balanced diet.

V. References

Data and Nutritional Values:

Askew, Eldon Wayne (2023): Mean Daily Nutritional Intake for Group 3. . Available online at https://www.researchgate.net/figure/Mean-Daily-Nutritional-Intake-for-Group-3_tbl7_235045008, updated on 5/24/2023, checked on 5/24/2023.

Enjia (2023): Nutrition-Facts-for-McDonald-s-Menu/menu.csv at master · Enjia/Nutrition-Facts-for-McDonald-s-Menu. Available online at https://github.com/Enjia/Nutrition-Facts-for-McDonald-s-Menu/blob/master/menu.csv, updated on 5/24/2023, checked on 5/24/2023.

Fast Food Menu Prices (2022): McDonald's Menu Prices - Fast Food Menu Prices. Available online at https://www.fastfoodmenuprices.com/mcdonalds-prices/, updated on 3/3/2022, checked on 5/24/2023.

Sources for Hyperparameter Tuning

Datta, Subham (2023): Genetic Algorithms: Crossover Probability and Mutation Probability. In Baeldung on Computer Science, 8/1/2023. Available online at https://www.baeldung.com/cs/genetic-algorithms-crossover-probability-and-mutation-probability, checked on 5/27/2023.

Datta, Subham (2023): Genetic Algorithms: Crossover Probability and Mutation Probability. In Baeldung on Computer Science, 8/1/2023. Available online at https://www.baeldung.com/cs/genetic-algorithms-crossover-probability-and-mutation-probability, checked on 5/27/2023.

IEEE (Ed.) (2018): 2018 IEEE International Conference on Systems, Man, and Cybernetics (SMC). 2018 IEEE International Conference on Systems, Man, and Cybernetics (SMC).

Mathworks (2023): Effects of Genetic Algorithm Options - MATLAB & Simulink. Available online at https://www.mathworks.com/help/gads/options-in-genetic-algorithm.html, updated on 5/27/2023, checked on 5/27/2023.

Miller, Brad L.: Genetic Algorithms, Tournament Selection, and the Effects of Noise. Available online at https://wpmedia.wolfram.com/uploads/sites/13/2018/02/09-3-2.pdf, checked on 5/27/2023.

Tutorialspoint (2023): Genetic Algorithms - Crossover. Available online at https://www.tutorialspoint.com/genetic_algorithms/genetic_algorithms_crossover.htm, updated on 5/1/2023, checked on 5/27/2023.

Tutorialspoint (2023): Genetic Algorithms - Mutation. Available online at https://www.tutorialspoint.com/genetic_algorithms/genetic_algorithms_mutation.htm, updated on 5/1/2023, checked on 5/27/2023.

Tutorialspoint (2023): Genetic Algorithms - Population. Available online at https://www.tutorialspoint.com/genetic_algorithms/genetic_algorithms_population.htm, updated on 5/1/2023, checked on 5/27/2023.

Y. Lavinas; C. Aranha; T. Sakurai; M. Ladeira (2018): Experimental Analysis of the Tournament Size on Genetic Algorithms. In IEEE (Ed.): 2018 IEEE International Conference on Systems, Man, and Cybernetics (SMC). 2018 IEEE International Conference on Systems, Man, and Cybernetics (SMC), pp. 3647–3653.