Computational Intelligence for Optimization

2021/2022

Travelling Salesman Problem using Genetic Algorithm

Luís Santos	**20210694**
Gaurav Luitel	**20210979**

Repository: https://github.com/luisfilipe-santos/CIFO_group_R

Remark: Although this report exceeds the limited number of pages, we find that, due to its fair number of figures and plots, the average amount of words per page doesn't drift from what's commonly used in a 5 pages report.

1. Objective

The ambition of this project is to apply and explore the usage of Genetic Algorithm (GA) and its methodologies, in order to reach an optimized solution for the Travelling Salesman Problems (TSP).

The goal of this challenge was to contrast the set of different solutions for the TSP, obtained through multiple conjugations of the different methodologies taught during this course and used throughout this project.

Furthermore, this report will cover some of the theoretical concepts associated with Genetic Algorithms and will focus on the application and demonstration of these.

For this end, we used and expanded the libraries created and developed throughout this course.

2. Travelling Salesman Problem

The TSP is an algorithmic problem tasked with finding the shortest route between a set of points and locations that must be visited.

The premise of the TSP is trivial enough, although, when scaled up, any manual attempt at solving the problem becomes futile. A problem with 10 locations already puts us in the millions of different possible tours, so we'll resort to more efficient ways such as genetic algorithms.

3. Genetic Algorithm

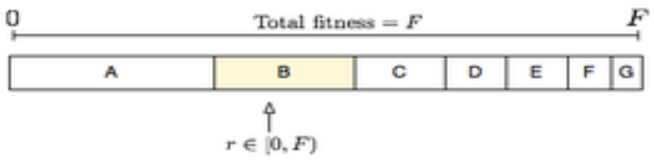
3.1. Introduction

A genetic algorithm is a heuristic search inspired by Charles Darwin's theory of natural evolution. It's also commonly used in optimization problems wherein we have to maximize or minimize a given objective function.

3.2. Methodology

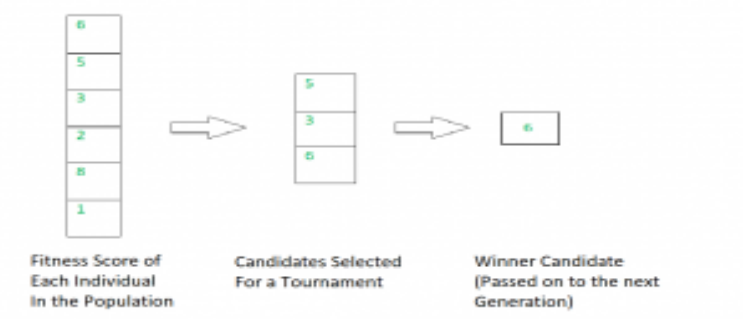
This process can be summarized in 5 different phases: initial population; evaluation (fitness function); selection; crossover; mutation.
It begins with a set of individuals known as Population. Each individual is a different solution to the problem at hand.
In the evaluation phase, the fitness function is responsible for determining how fit each individual is. Given a fitness score the algorithm move on to a Selection phase. In this phase, two pairs of individuals (parents) are selected based on their fitness scores and according a selection method.
The following, are the selection methods used in this project:

1. Fitness Proportionate Selection



An individual is selected based on a probability which is proportional to its fitness. Since the goal of our problem is to minimize the fitness of our solutions, our implementation of FPS for minimization problems replaced the fitness of every individual for (total fitness – individual's fitness).

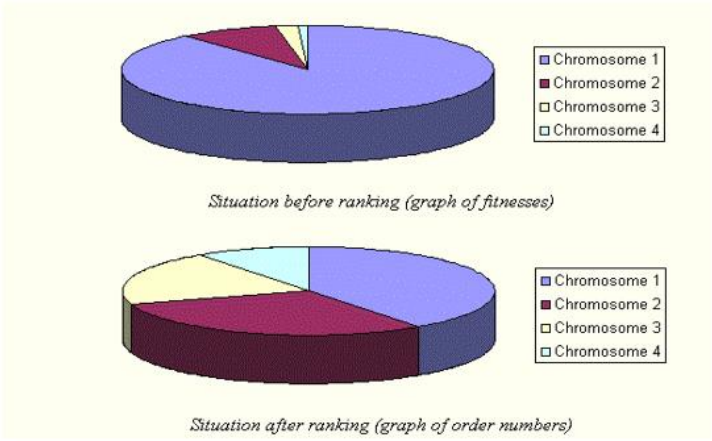
2. Tournament Selection



Tournament selection involves running several "tournaments" among multiple individuals (tournament size) with each winner(s) being selected.

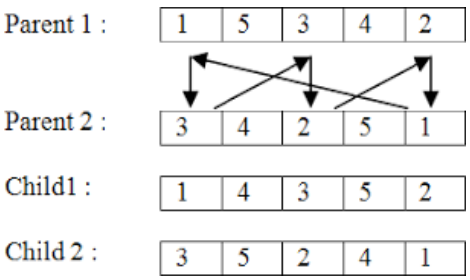
3. Ranking Selection

Ranking selection sorts the population first according to fitness value and ranks them accordingly. This can prevent situations like the one shown below.



In the Crossover phase, the genetic information of two selected parents is used to generate a new offspring given a certain probability. For this effect we used the following three methods, the first two already created in libraries provided:

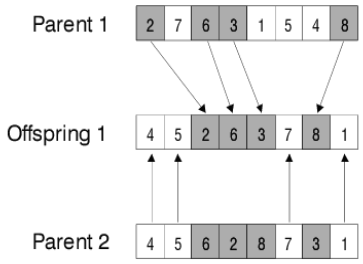
1. Cycle Crossover



2. Partially Mapped Crossover

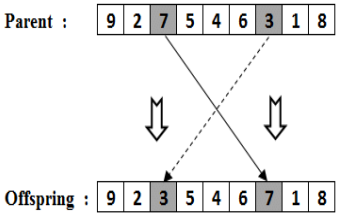
3. Order Based Crossover

We also added on more crossover method, called order based, it firstly selects a subset of cities in the first parent, next, in the offspring, these cities appear in the same order as in the first parent, but at positions taken from the second parent. Then, the remaining positions are filled with the cities of the second parent.

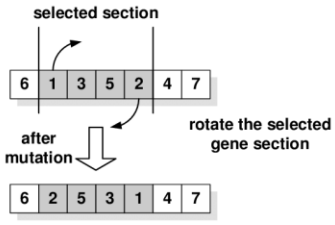


Finally, in the Mutation phase each individual may be subject to a mutation, given a small probably, in order to maintain genetic diversity from one generation to another. We used three different methods for this purpose:

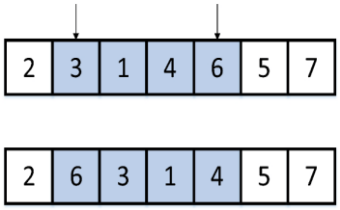
1. Swap Mutation



2. Inversion Mutation



3. Scramble Mutation



4. Application

4.1 Our Data and Fitness function

We collected two more sets of data adding to the already existing one, in the Charles library, each with the same representation but different sizes (9, 13 and 27 locations). For the implementation of the fitness function, we reused the one provided in the Charles library and commonly used in the TSP, which is essentially, the total length of the route taken.

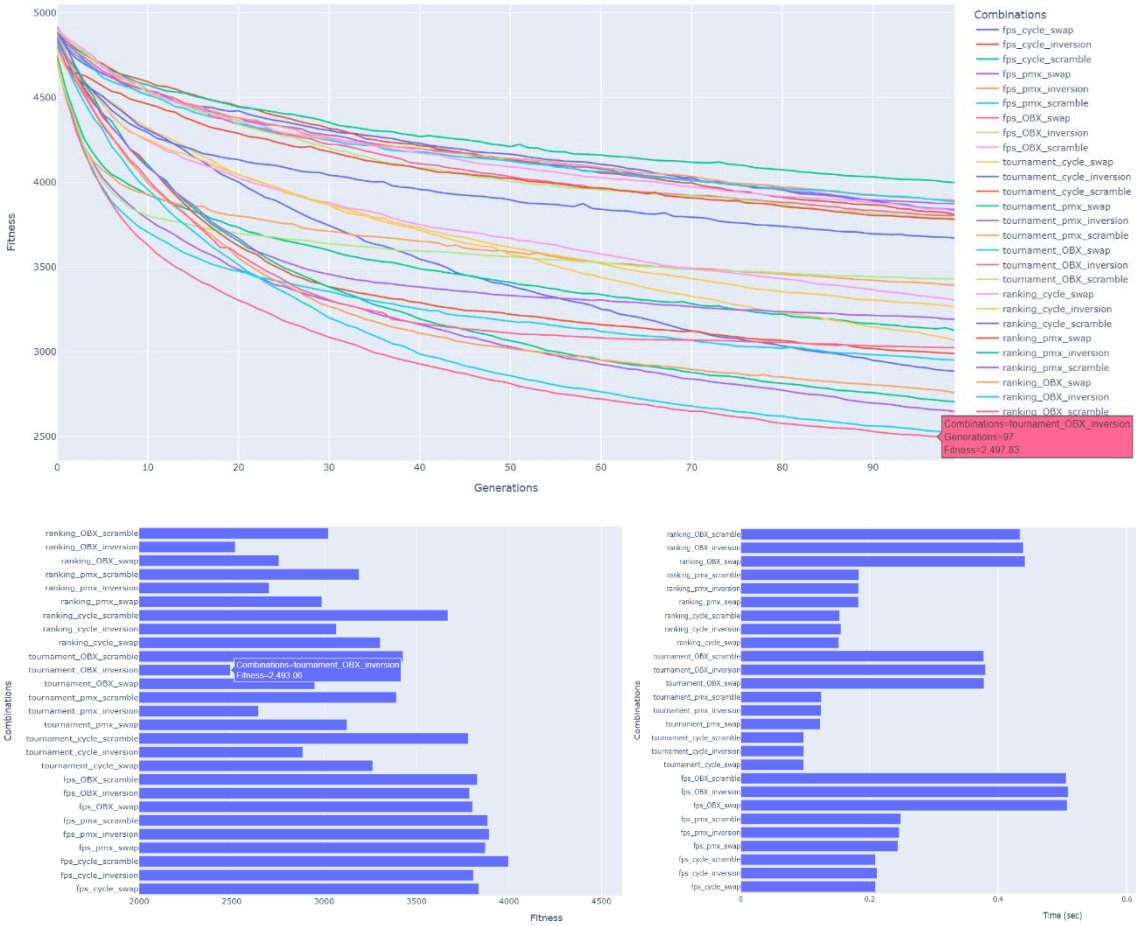
4.2 Initial Parameters

In the benchmarking phase of this project we made some initial runs of our algorithm in order to apprehend which combination of selection methods and operators would better fit each provided set of data. We settle on a crossover and mutation probability of 0.9 and 0.05, respectively. These, are values that fall in the 'commonly used' window for genetic algorithms, and also worked the best for the data we provided. We also set our population size to 50 and the number of generations in our evolution process to 100. Each combination ran 100 times, with the results being averaged out in the end. Elitism and Replacement were set True and False, respectively.

4.3. Phase 1

Given the initial parameters, these were the results of the 3 datasets:

1. Bavarian Cities (27 locations)



We remind that in the first visualization, every value in each line is an average of the fitness obtained at each generation of every run.
It's apparent that, for this dataset, combinations with tournament or ranking selection methods not only reach early better fitness scores but also mature into better solutions than the FPS method, this seemed to be the case for the larger datasets. Our initial benchmark of this dataset, gave the combinations of tournament selection, order based crossover and inversion mutation the best fitness score (2493) out every other combinations for this data set. For a second phase, where we'll be testing what other parameters better influence our algorithm, we'll be using not only this combination but 'ranking_OBX_inversion' and 'tournament_pmx_inversion' as well, since they are, respectively, a close second best in terms of fitness and the best combination in terms of fitness per time.
We had similar approaches to the remaining datasets and picked the combinations bellow, following the same criteria.

2. Bangladesh cities (8 locations)

- fps_OBX_swap (best fitness)
- fps_OBX_scramble (2nd best)
- tournament_cycle_swap (fitness per time)

3. Dataset from Charles (13 locations)

- ranking_OBX_inversion (best fitness)
- tournament_OBX_inversion (2nd best)
- tournament_cycle_inversion (fitness per time)

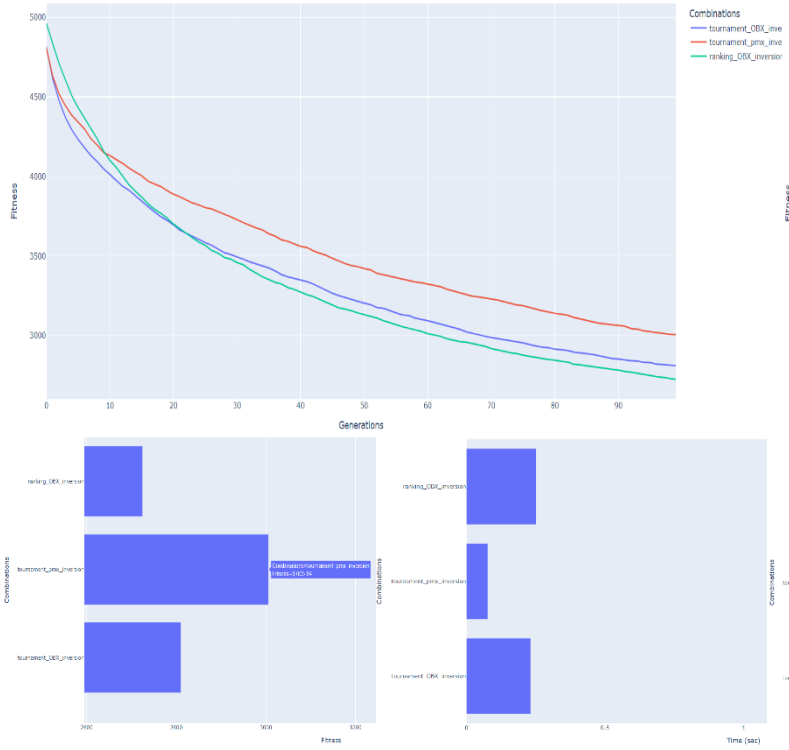
4.4. Phase 2

In this phase we'll be only focusing in the combinations chosen for the Bavarian dataset since it's the most challenging one and the procedures would be same for the other two datasets. We'll try to observed how our algorithm changes with different sizes of population, number of generations, the usage of elitism.

4.4.1. Different Population sizes and number of Generation

Bavarian Cities (27 locations)

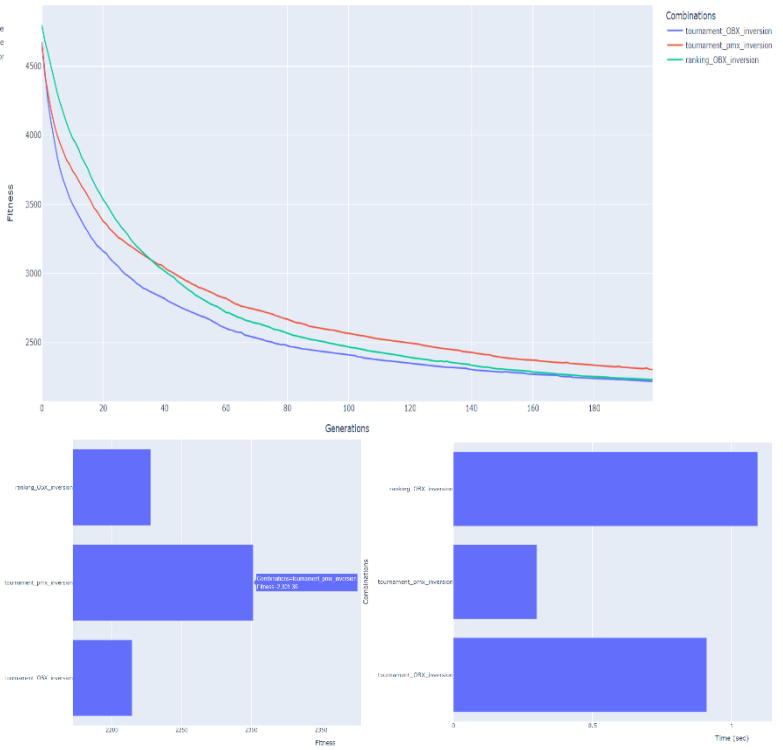
Generations number: 100
Population Size: 30



Best Fitness: 2724.84

Best Time: 0.076 (sec)

Generations number: 200
Population Size: 60



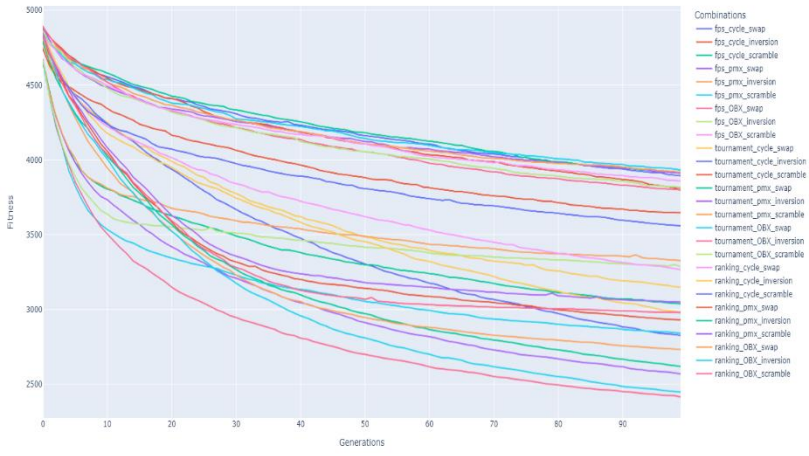
Best Fitness: 2214.42

Best Time: 0.30 (sec)

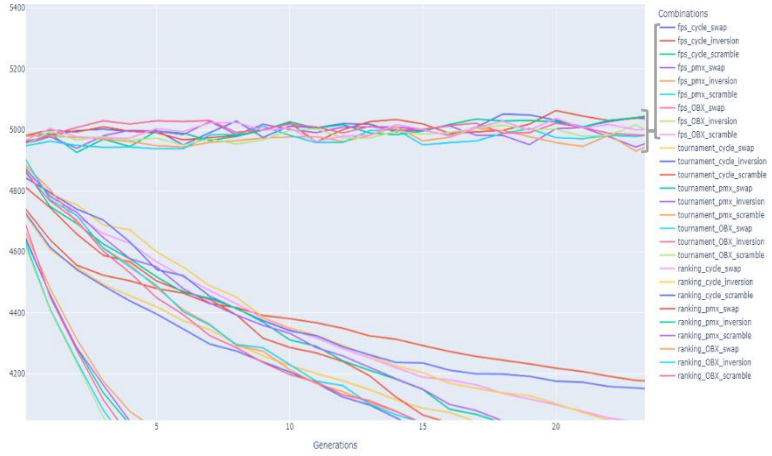
It's evident by the plots above that to have better fitness you have to compromise resources so there's no exact answer to 'What should my population size be?' or 'How many generations should I run?', it depends on what the goal of solving the problem is.

4.4.2. Elitism

With Elitism:



Without Elitism:



Despite not existing a huge amount of different between the final fitness values for the best preforming combinations, most of the combinations preformed worse and our implementation of the fitness proportion selection method is not compatible with elitism turned off.

5. Conclusion

Like previously said, the best way to achieve the best results in a TSP using Genetic algorithms is to exhaustively keep tweaking the parameters of the algorithm. This process should be done with the specific goal of the problem in mind, be it a maximization, minimization or any other type of objective. In our case, and most commonly in the TSP, our target was minimization, even despite our algorithm being susceptible to a maximization, nevertheless, our goal was to lower our fitness values as much as computationally viable. To that effect, our best results were the following:

For the Bavarian Dataset (27 locations) our best results were obtained with a Tournament selection method with an Order Based crossover and an Inversion Mutation operator.

For the dataset provided in the Charles library (13 locations), our best results were obtained with a Ranking selection method with an Order Based crossover and an Inversion mutation operator.

Finally for our smallest dataset (8 locations), containing the cities of Bangladesh, our best results were obtained with Fitness Proportionate selection method with and Order Based crossover and Swap mutation operator.

6. References

- Wikipedia, Genetic algorithm, https://en.wikipedia.org/wiki/Genetic_algorithm

- Pedram Ataee, (2020), How to solve the Travelling Salesman Problem, https://towardsdatascience.com/how-to-solve-the-traveling-salesman-problem-a-comparative-analysis-39056a916c9f

- Jason Brownlee, (2021), Simple Genetic Algorithm From Scratch in Python, https://machinelearningmastery.com/simple-genetic-algorithm-from-scratch-in-python/

- Stack Overflow, https://stackoverflow.com/

- ResearchGate, https://www.researchgate.net/