COMPUTATIONAL INTELLIGENCE FOR OPTIMIZATION

MASTER DEGREE PROGRAM IN DATA SCIENCE AND ADVANCED ANALYTICS

Group K:
Danilo Arfeli, number: 20211296
Gabriel Avezum, number: 20210663

Github:
https://github.com/gavezum/Genetic-Algorithm

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

1. Introduction

Genetic algorithms (GA´s) can be implemented to find a solution to the optimization problems of various types they are a technique for searching optimal solutions in a space of possible alternative solutions, do not require an exhaustive analysis of the search space and typically begin their search with randomly generated solutions, try to improve their quality in a stepwise refinement fashion, by means of an iterative algorithm, the potential solutions (individuals) are represented as strings of characters of a prefixed length, the solutions that are considered at each iteration (generation) are more than one, and the principle used as an inspiration is the theory of evolution of Darwin. [1]

In this project we worked with genetic algorithms to try to find the optimum solution to the traveler salesman person problem, where we have a number of cities and we need to visit each one only one time and return to the starting city, in 2 datasets.

2. Datasets

The first data set called rd100.tsp was extracted from the website presented in the project guidelines, representing 100 cities, and the best solution has a fitness of 7910. The second data set is called dj38.tsp [2] represents 38 Cities from Djibouti and the best solution has a fitness of 6656.

3. Distance Matrix, Representation and Fitness function

The first step we take to to build the solutions is to create a distance matrix where each element will be a distance between the cities on the indexes, with this representation the path cannot pass twice on the same city and we can use the negative index to return to the first city and give us an easy understanding of the path if necessary.

In this way it's possible to create a representation that will have a length equal to the number of cities in our problem and on each element will be the index of the distance matrix.

Since the goal is to minimize the path, the fitness function needs to be the sum of all the distances, so combining this idea with the representation the function was defined to be the sum of distances between the elements of the representation. Therefore our problem is TSP we didn't try different fitness functions.

Dataset	Representation	Fitness functions
rd100.tsp	[0,1,2,3,4,...,Len -1] with length 100	Sum(Distance of elements in distribution)
dj38.tsp	[0,1,2,3,4,...,Len -1] with length 38	Sum(Distance of elements in distribution)

4. GA´s Application

For both datasets and during all the experiments it was decided to run the genetic algorithm fifty times and then take the average of the best fitness of each generation.

As the first approach we will look to test the number of generations and population, and the base parameter in this exercise will be set with: tournament selection, cycle crossover, inverse mutation, probability of mutation equals to 0.1, probability to cross over equal to 0.9 and elitism equals True.

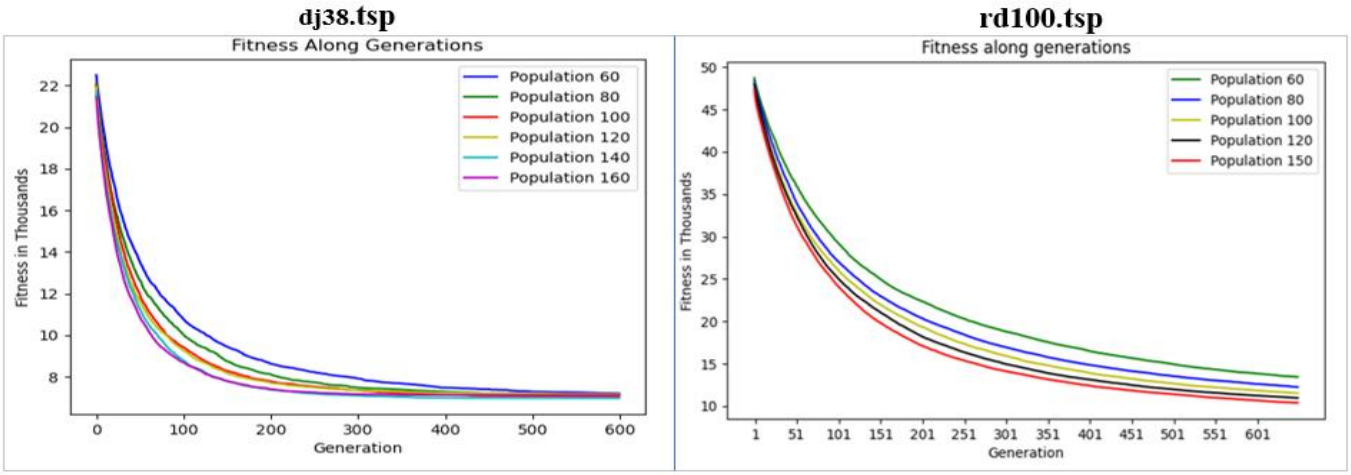

Figure 1: x-axis represents the number of generations, the y- axis is the fitness and each line is a different population size

In the figure above we can see for the rd.100 tsp data set that we have the number of generations on the x-axis and each line is a different population size. Looking at the lines it is possible to notice that after the generation 350 the fitness has a slow decay and the fitness doesn't change much, so it was decided to keep the generations to 350. The population size has a good effect on fitness until a size of 120, after this the improvement is marginal and the processing time is longer, so we decided to use the population equal to 120.

For the dj38.tsp dataset, we can see an exponential drop of all the lines along the first generation but the lines representing the population of 140 and 160 have a bigger decrease in fitness. Around the 200 generation the fitness of population of 140 and 160 almost stabilizes and the rest stabilizes around 350 - 400. So to optimize the processing time it was decided to use a population equal to 140 with 200 generations.

Dataset	Population	Generations
rd100.tsp	120	350
dj38.tsp	140	200

5. Selection algorithm

At this point the population and generations are decided and settled. Now we are going to test the others parameters while the others will remain the same, the first one to be changed is the selection algorithm.

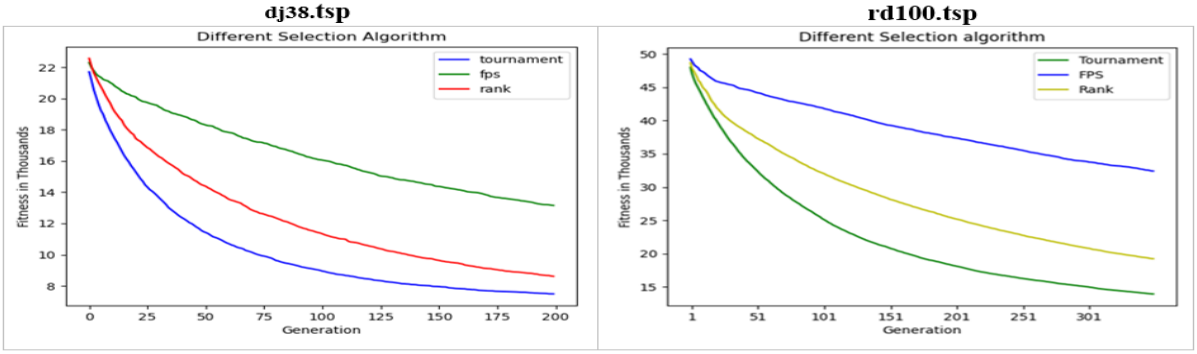

Figure 2: x-axis represents the number of generations, the y- axis is the fitness and each line is a different Selection algorithm

In the figure above we can see for the rd.100 tsp and dj38.tsp data that the **tournament** has the lowest fitness values in all generations so it will be our best selection algorithm. We also tested different parameters of the tournament selection, but they gave the same result, so it was decided to not present a graph.

6. Mutation

Continuing to this analysis we are going to test 3 mutation algorithm: swap, inverse and center_inverse

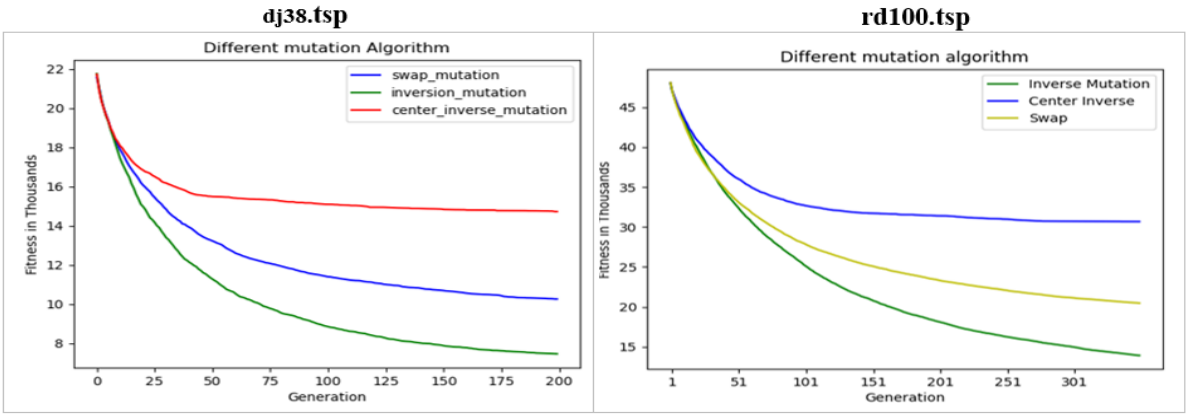

Figure 3: x-axis represents the number of generations, the y- axis is the fitness and each line is a different mutation algorithm

In the figure 3 we can see for the rd.100 tsp and dj38.tsp data that as the selection algorithm the mutation presents a great difference in fitness in all of the algorithms and having the inversion_mutation with lowestfitness between the rest, so we are going to move forward with it.

Here we can see a huge impact in the convergence of our algorithm in both datasets when we use only center inverse mutation, another conclusion that we can take from the graph is the impact of mutation in our problem, so changing the mutation probability should be tested.

7. Crossover

Going forward to the last algorithm possible, that is the crossover algorithm., testing 3 diferent types: PMX, 01, Cycle.

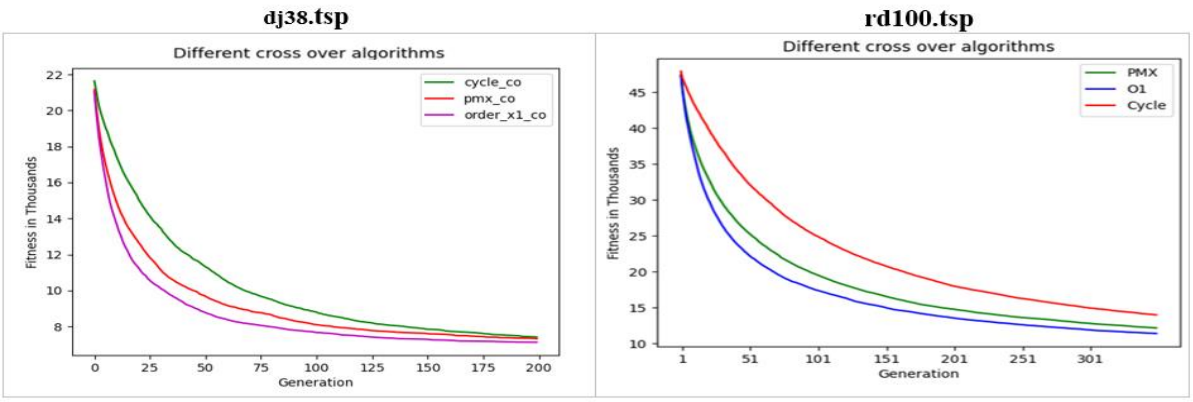

Figure 4: x-axis represents the number of generations, y- axis is the fitness and each line is a different Cross over algorithm

In the figure 4 we can see for the rd.100 tsp and dj38.tsp data that the **o1(or order_x1)** has the lowest fitness values in all generations so it will be our best selection algorithm. An interesting fact happened here, for both problems we changed our defaut crossover algorithms from cycle to O1.

8. Other tests

For both datasets we can also try different values for Elitism, mutation probability and crossover probability. For both of them the different values of Elitism and crossover probability didn't make a difference in fitness, for the mutation we tested several of them and obtained 0.2 as the best value for mutation (change from 0.1 to 0.2).

So we have reached the final test and reached the final parameters as:

Dataset	Representation	Fitness functions	Population	Generations	Selection	Mutation - Prob.	Crossover - Prob.	Elitism
rd100.tsp	[0,1,2,3,4,....,Len -1] with length 100	Sum(Distance of elements in distribution)	120	350	Tournament	Inverse - 0.2	O1 - 0.9	TRUE
dj38.tsp	[0,1,2,3,4,....,Len -1] with length 38	Sum(Distance of elements in distribution)	140	200	Tournament	Inverse - 0.2	O1 - 0.9	TRUE

9. Results and Conclusions.

In the website we can check that the best solution for both datasets, the rd.100 tsp has a fitness of 7910 and in our GA we found a solution with the best fitness equal to 9676, so we have a space to improve. Maybe one way will be to try other mutations and cross over algorithms or we could implement the algorithm for premature convergence. The dj38.tsp has a fitness of 6656 and in our GA we found a solution with the best fitness equal to 6656, so we reached the optimum solution.

10. References

[1]
https://elearning.novaims.unl.pt/pluginfile.php/110499/mod_resource/content/1/BOOKLET_CIFO_Leonardo_Vanneschi.pdf

[2]
http://www.math.uwaterloo.ca/tsp/world/countries.html#WI