Traveling Salesperson Study

Camila Sarmiento m20201019, Laura Ramos m20200541
Omar Jarir m20201378, Natalia Castaneda m20200575

I. INTRODUCTION

This project aim is to solve the traveling salesperson problem using genetic algorithms, with different combinations of selection algorithms, crossover, and mutation operators. A comparative study analysis is used to identify the parameters that achieve the best results for the problem in question.

II. BACKGROUND

The traveling salesperson problem is an NP-hard problem of combinatorial optimization, it intends to respond to the following question: "Given a list of cities and the distances between each pair of cities, what is the shortest possible route that visits each city exactly once and returns to the origin city?"

To apply a genetic algorithm, we need to create a population composed of a predefined number of individuals which can be represented as an array composed of the possible combination of cities indexes, as it is a TSP problem each city must be visited only once, at the end the traveling salesperson return to the first city, this is not represented at the end of the array to ease the representation.

Example: if the salesperson must visit 5 cities with indexes 1,2,3,4,5 a representation of an individual (possible route) is [2,3,1,5,4].

The fitness function is used to detect the best individuals in a population. In the context of our problem, considering the distance matrix of all cities, the fitness function corresponds to the cumulative distance between the cities in each representation given its order.

We also considered creating a fitness function based on traveling time but since the speed it not constant we discarded the option.

Elitism was implemented. The user can choose the number of best individuals to keep from the previous population.

The code implementation is abstract and work with both minimization, when the fitness function is the sum of the distances, and maximization, when the fitness function is one over the sum of the distances.

III. METHODOLOGY

- **Selection methods:** The three classic selection method are in use in this project, the fitness proportionate selection, the rank selection, and the tournament selection. All of them require a population as input, and the tournament selection require a tournament size argument.

- **Mutation methods:** The three mutation methods used in the scope of this project respect the uniqueness of cities in the representation. They are swap mutation, inversion mutation, and shuffle mutation.

- **Crossover Methods:** Three crossover operators are used; they require two individuals as parameters. They are cycle crossover, partially mapped crossover (pmx), and the order 1 crossover.

Order 1 crossover works as follow:

Given the following two parents, parent1 = [1, 2, 3, 4, 5, 6, 7, 8, 9], and parent2 = [3, 4, 7, 2, 8, 9, 1, 6, 5].

We randomly select a set from first parent, and copy it into the first offspring, then we copy the rest from second parent in order 1,2,3,8,9, the offspring is given by offspring1 = [3, 2, 8, 4, 5, 6, 7, 9, 1].

IV. RESULTS

In this section, we present some of the results obtained, combining different selection algorithms and operators, and using three different distance matrix which are swiss42, brazil58, and bayg29.

Two experiments were done, each one will be detailed below.

a. Statistical analysis experiment 1:

We conducted statistical analysis for different selection algorithms, mutation, and crossover operators' combinations.

Considering a population size of 50 individuals, taking the total number of generations equal to 750. A crossover rate of 0.8 and a mutation rate of 0.1.

The Figure 1. shows the average of the best fitness for each generation, considering 30 runs for each combination, without the use of elitism, taking a tournament size equal to 10.

Average fitness per generation

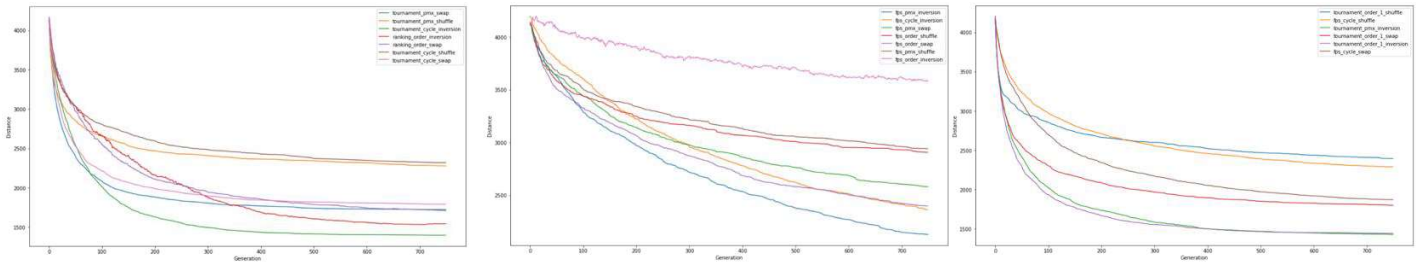
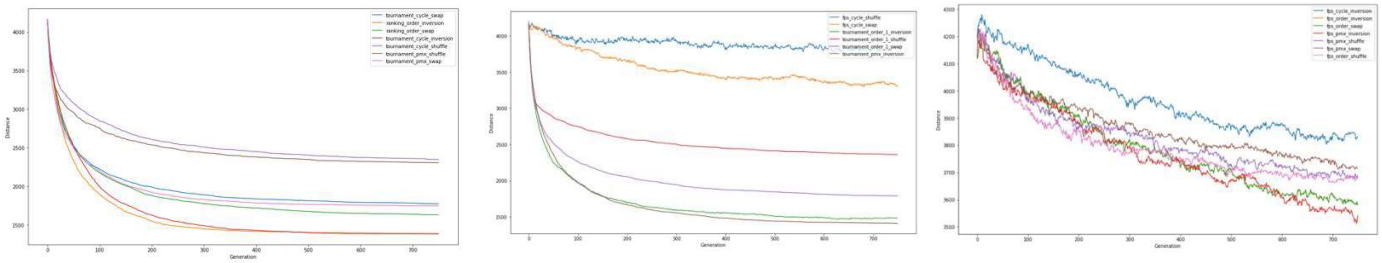


Figure 1. Average fitness per generation without elitism.

When using elitism, we kept the top 6 individuals from the initial population, and we get the results shown in the Figure 2.



Average fitness per generation

Figure 2. Average fitness per generation with elitism.

The Figures show us that while using elitism the algorithm converges faster, and the fitness function is strictly decreasing.

We can see that the combination of tournament selection, pmx crossover, and inversion mutation performs best either using elitism or not, achieving similar performance.

A performance that is closely followed by the combination of tournament selection, order 1 crossover, and inversion mutation which perform good in both cases.

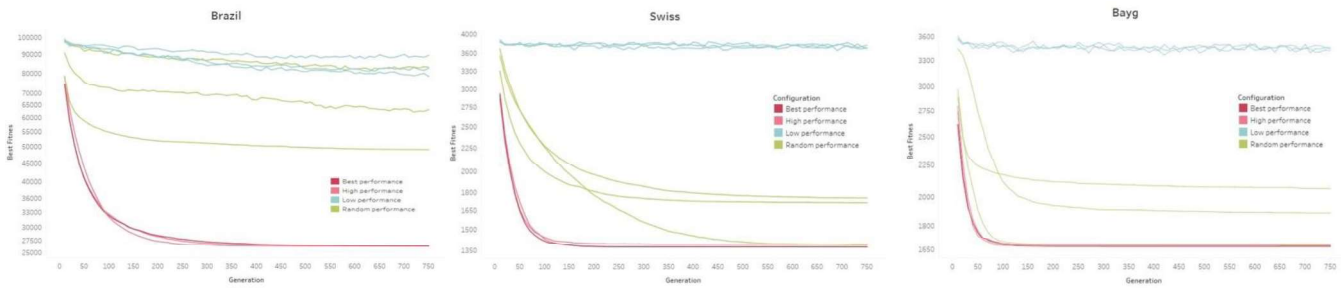
The tournament selection method, the inversion mutation, and the pmx crossover seem to be the bests in each category.

The fitness proportionate selection operator performs poorly because the individuals in the population have close fitness value and consequently similar probability of being selected.

b. Statistical analysis experiment 2:

576 scenarios were made for each data set where all possible combinations were treated. They were used. Crossover rate (0.8 and 0.9), Mutation rate (0.2 and 0.5), The algorithm running with and without elitism with Size of the elite (10 and 20), also was implemented fitness sharing the mutation, selection and crossover were the same listed before.
Each combination was executed only once, then three groups were selected, the three-best fitness, the three worst and additional 3 groups were randomly taken to see how after 40 executed each combination would improve.

Average fitness per generation by dataset



The graphs above show how bad groups in the first run stayed bad during the rest of the runs. While the random and the best selections increased their performance over time. This gives an indication of how the first execution can give information about what will happen in the future with the combinations if they are executed 40 times.

Bayg29

Segment	Elitism and fitness sharing	Elite size	Crossover rate	Mutation rate	Select	Crossover	Mutate	Best Fitness
Best	None		0.9	0.2	tournament_selection	order_1_crossover	inversion_mutation	1610
Best	Elitism	10	0.9	0.5	tournament_selection	pmx_crossover	inversion_mutation	1615
Random	Elitism	20	0.9	0.5	tournament_selection	cycle_co	inversion_mutation	1618
Best	Elitism and sharing	20	0.8	0.5	tournament_selection	pmx_crossover	inversion_mutation	1620
Random	None		0.9	0.2	ranking_selection	pmx_crossover	shuffle_mutation	1661
Random	Elitism and sharing	10	0.8	0.5	tournament_selection	pmx_crossover	shuffle_mutation	1694
Bad	Sharing		0.9	0.5	ranking_selection	pmx_crossover	shuffle_mutation	2468
Bad	Sharing		0.9	0.5	ranking_selection	pmx_crossover	shuffle_mutation	2560
Bad	None		0.9	0.5	ranking_selection	pmx_crossover	shuffle_mutation	2582

brasil

Segment	Elitism and fitness sharing	Elite size	Crossover rate	Mutation rate	Select	Crossover	Mutate	Best Fitness
Best	None		0.8	0.2	tournament_selection	order_1_crossover	inversion_mutation	25400
Best	Sharing		0.8	0.2	tournament_selection	order_1_crossover	inversion_mutation	25475
Best	Elitism	20	0.8	0.5	tournament_selection	pmx_crossover	inversion_mutation	25562
Random	Sharing		0.9	0.2	ranking_selection	order_1_crossover	inversion_mutation	28349
Random	Elitism and sharing	10	0.9	0.5	tournament_selection	order_1_crossover	shuffle_mutation	39057
Random	Sharing		0.8	0.5	ranking_selection	pmx_crossover	shuffle_mutation	49967
Bad	None		0.8	0.5	ranking_selection	pmx_crossover	shuffle_mutation	50268
Bad	Sharing		0.8	0.5	ranking_selection	pmx_crossover	shuffle_mutation	51798
Bad	None		0.9	0.5	ranking_selection	pmx_crossover	shuffle_mutation	60833

Swiss

Segment	Elitism and fitness sharing	Elite size	Crossover rate	Mutation rate	Select	Crossover	Mutate	Best Fitness
Best	Elitism	10	0.9	0.5	tournament_selection	order_1_crossover	inversion_mutation	1257
Best	Elitism	20	0.9	0.5	tournament_selection	order_1_crossover	inversion_mutation	1266
Random	Elitism	10	0.8	0.5	ranking_selection	cycle_co	inversion_mutation	1273
Best	Elitism and sharing	10	0.8	0.5	tournament_selection	order_1_crossover	inversion_mutation	1295
Random	Elitism and sharing	10	0.8	0.2	tournament_selection	cycle_co	swap_mutation	1421
Random	None		0.8	0.2	ranking_selection	cycle_co	swap_mutation	1483
Bad	None		0.9	0.5	ranking_selection	pmx_crossover	shuffle_mutation	2606
Bad	Sharing		0.9	0.5	ranking_selection	pmx_crossover	shuffle_mutation	2819
Bad	Sharing		0.9	0.5	ranking_selection	pmx_crossover	shuffle_mutation	2949

c. Succes rate:

After many tries the global minimum found for the distance matrix swiss42 was 1257.

Only the combination of tournament selection pmx crossover inversion mutation without the use of elitism, could found a local minimum bellow 1300, only 3 times in 30 runs, yielding a success rate of 1/10, cases with zero success rate are not considered.

	No Elitism.	Elitism
Tournament pmx inversion	1/10	0
Ranking order inversion	5/30	4/30

Considering local minimums bellow 1400, we can see that only the combinations of tournament selection pmx crossover and inversion mutation, the combination of tournament order 1 and inversion mutation, the combination of tournament selection cycle crossover and inversion mutation, and the combination of ranking selection order 1 mutation and inversion mutation,

found local minimum below that threshold, the table below summarizes the results.

	No Elitism.	Elitism
Tournament pmx inversion	12/30	11
Tournament order 1 inversion	1/2	16/30
Ranking order inversion	1/30	0
Tournament cycle inversion	21/30	21/30

These results are illustrated by the fact that toward the end generations the algorithms get stuck in a local minimum, this might be due to the lack of diversity among individuals and the population size.

V. CONCLUSIONS

The results obtained in this work are satisfactory since optimal values were obtained. We also find the different parameters that managed to optimize the fitness function.

Implementing others crossover operator might improve the converge of the genetic algorithm, Edge recombination crossover among others could be considered.

Some limitation for this exercise are the execution time and the cpu capability for running more executions. With better machines we could be able to find and optimal number of executions for best performance.

I. REFERENCES

Genetic Algorithms. Genetic Algorithm Tutorial. (n.d.). https://www.rubicite.com/Tutorials/GeneticAlgorithms.aspx.

Goldberg, D. E. (1989). *Genetic algorithms in search, optimization, and machine learning*.

Github repo: https://github.com/omarja12/Cifo_Project