NOVA IMS Information Management School

MSc in Data Science and Advanced Analytics 2021/2022

Computational Intelligence for Optimization

Genetic Algorithm for Travelling Salesperson Problem (TSP)

GITHUB LINK:

https://github.com/veracanhoto/CIFO_project21-22_Group_L.git

Ramzi Ayass, m20210705

Lorena Hahn, m20211302

Vera Canhoto, m20210659

Problem

In this assignment, we'll be using a variety of Genetic Algorithms exploring different configurations to find the best possible solution for the Travelling Salesman Problem (TSP). Given a list of 17 cities and the distances between each pair, our challenge is to find the best possible solution to find the shortest possible route in order to visit all cities and return to the original city.

Representation

For our representation, we considered that each sequence of locations represents one individual.

Considering our problem description, we had in mind the following rules to create our representation:

- Each location can only be visited once.
- In the end of our journey, we must return to the first location. Therefore, the distance needs to be calculated accordingly.
- Each route that respects the rules above can be considered as a possible 'Parent'.
- Each 'Parent' when mutated or combined (crossover) with another 'parent' can originate a new route 'child'.
- Several 'Parents' can be used or combined to create the next generation (new routes).

Fitness Function

Genetic algorithms are inspired by the idea of "Survival of the fittest". Therefore, it was essential to define a fitness function for each individual in order to assess how close a given solution is to the optimum solution, in other words how good is each route. In this case we considered the fitness of an individual to be the sum of the distance for each pair of consecutive cities, including the sum of the last city in the sequence and the first.

Fitness=[∑N−1i=1Distance(Cityi,Cityi+1)]+Distance(CityN,City1)

Genetic Algorithms - Mix of Configurations

To solve our problem, we tried a total of 12 different configurations with a mix of different *Selection, Mutation and Crossover methods.*

Selection

We considered two selection methods to select individuals from our population and create the next generations: Fitness proportionate selection (FPS), Tournament and Ranking.

In FPS, the fitness of each individual relative to the population is used to assign a probability of selection. Whilst in Tournament selection a group of individuals is randomly selected. The individual with the highest probability is then selected as the first parent. Finally, the process is repeated to select the second parent. Lastly, in Ranking selection the individuals are ordered in an ascending manner by their fitness values with the lowest fitness being assigned number one in the ranking. The individual, which has the best value of fitness is ranked as the last one. After this ranking process is done, FPS is applied based on the ranking previously established.

Crossover

After creating our mating pool, we can create the next generation through crossover.
Our TSP problem has a particularity that must be considered before we apply any crossover techniques. The cities in our population can only be included exactly one time. As per the way we created our representation, the cities never repeat because we assumed, while calculating the fitness, that we know that we go back to city number one. Hence, we selected two viable options regarding crossover. Cycle and partially mapped crossover (Pmx) crossover can be applied in our TSP as they don't repeat any cities. This would not be possible for example with binary mutation or single point crossover where cities can end up repeated.

For the Cycle crossover technique, we performed several cycles between two randomly selected parents. The Cycle crossover attempts to create an offspring from the distances where every position is occupied by a corresponding element from one of the distances.

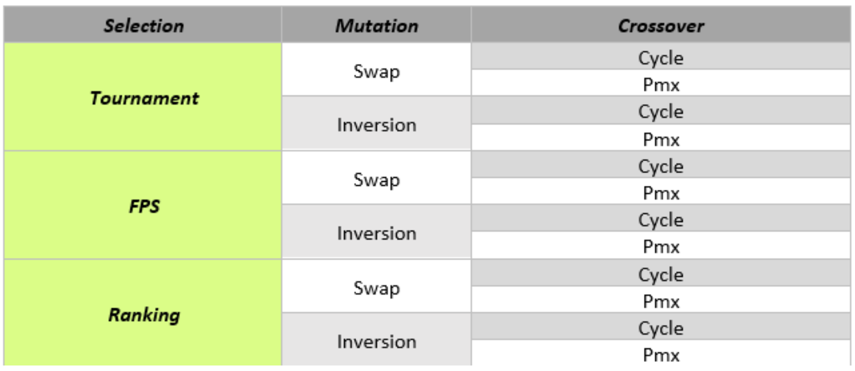
Furthermore, we used Pmx that generates offspring solutions by taking a subsequence from one of the distances and putting it into the other distance while retaining the original order of as many points as reasonable.

Mutation

By mutating individual genes, we gain the capability of gaining diversity and avoiding local convergence, as we introduce novelty and increase the solution space. When applying mutation, we need to consider the same limitation as previously considered for crossover, since each gene can only appear exactly once. Following this logic, we opted for two mutation solutions: Swap and Inversion.

In the swap mutation we select two positions at random and interchange the values. However, in the inversion we select two positions within a distance and then inverts the substring between these two positions.

Based on the above, we tried the following 12 configurations:



Selection	***Mutation***	***Crossover***
Tournament	Swap	Cycle
		Pmx
	Inversion	Cycle
		Pmx
FPS	Swap	Cycle
		Pmx
	Inversion	Cycle
		Pmx
Ranking	Swap	Cycle
		Pmx
	Inversion	Cycle
		Pmx

In addition to these configurations, we experimented changing the values of the parameters to find our best solution:

- Mutation rate
- Probability of the crossover
- Population size
- Number of generations

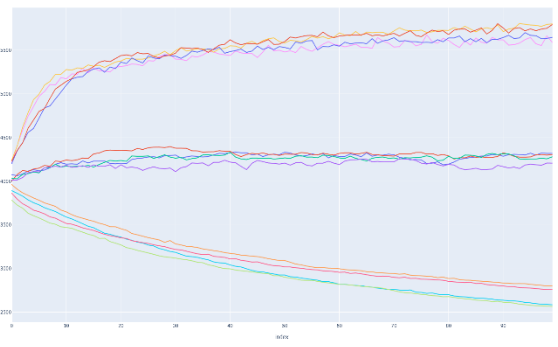
Each parameter was run 20 times and the best results were chosen. Firstly, we just adjusted the mutation rate and the probability of the crossover. However, we couldn't get good results out of it. After, we adjusted the number of generations to a higher number, and we started to obtain better results. We got even better results as we also increased the population size.
In general, while experimenting, we concluded that when choosing a higher number of generations and a population size, this positively impacted our results.

Elitism implementation

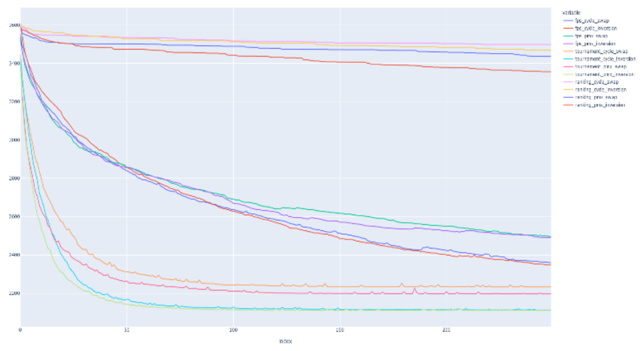
We attempted two approaches when implementing our algorithms. Firstly, without Elitism and then applying Elitism. Elitism is a technique that assures that the best performing individuals from a population will be present in the next generation, replacing the weakest individuals. In our case, as our population is not too large, we opted for replacing only one single weakest individual. When we apply elitism, we expect our results to be the same or better as we are replacing the weakest option for the strongest in the previous population.

Our results confirm this expectation as seen in the graphics below:

Without Elitism



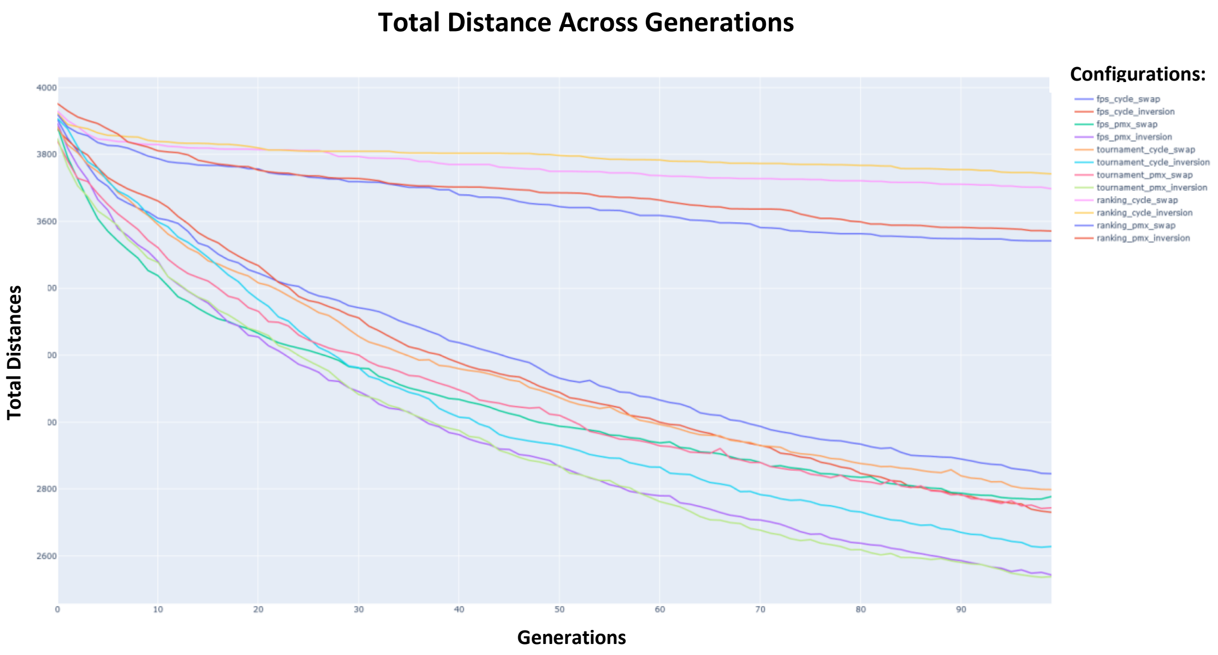
With Elitism



As shown in the graphics above, without elitism we obtained worse results for all configurations, but most of all the results obtained were comparatively much lower when using Ranking and then FPS as our selection methods. Therefore, we decided to apply elitism in our final solution to optimize our results.

Results

In genetic algorithms we never know where the global optimum is, so we just need to find an acceptable solution and then we stop. In our first attempt we tried all our configurations with a Mutation Rate of 0.1, a Crossover Probability of 0.8, a Sample Population of 10 and a total of 100 Generations. For this attempt a combination of Tournament-Pmx- Inversion has yield the best result as shown by the graphic below.

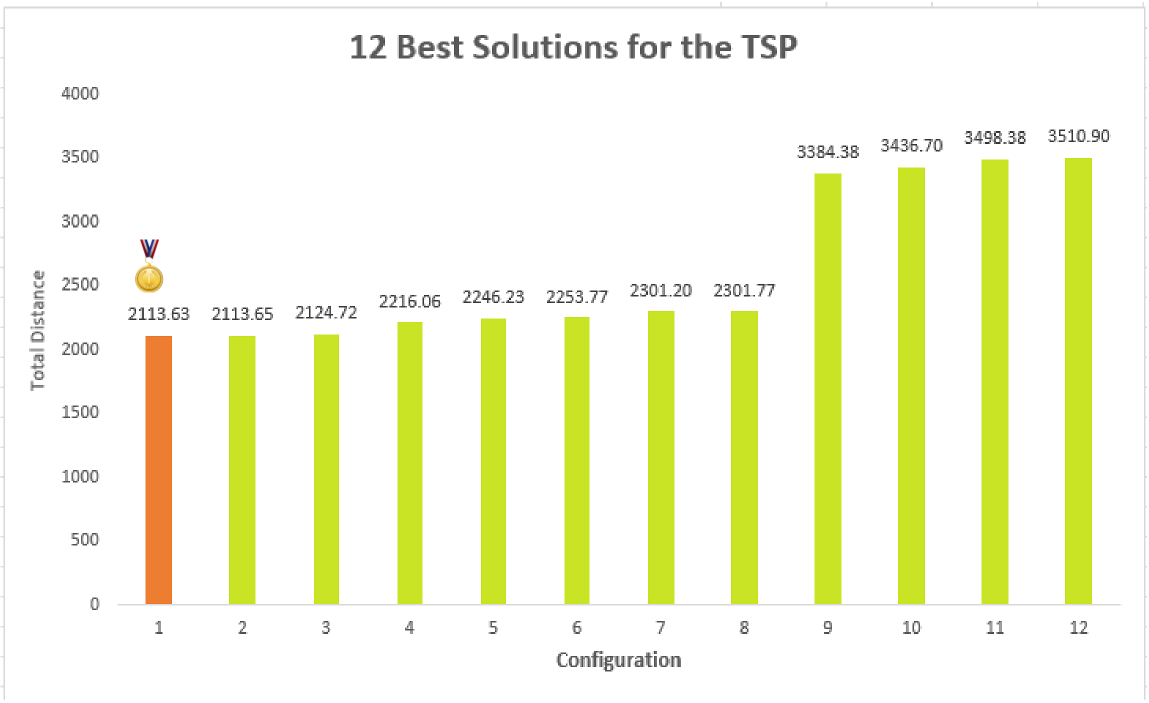


Looking at the graphic above we could clearly see that improving the number of generations was positively contributing for our solution. So, we decided to experiment and adjust different configurations as follows:

Configurations			*Parameters*																			
Selection	Crossover	***Mutation***	**Mutation rate**					**Probability of the cross over**					**Population size**					**Number of generations**				
			0.1	0.1	0.25	0.2	0.2	0.8	0.8	0.95	0.9	0.8	50	25	10	50	20	250	250	100	250	200
Tournament	Cycle	Swap	2216.06	2272.63	2490.8	2226.01	2253.77	2216.06	2272.63	2490.8	2226.01	2253.77	2216.06	2272.63	2490.8	2226.01	2253.77	2216.06	2272.63	2490.8	2226.01	2253.77
		Inversion	2113.63	2129.82	2327.75	2111.05	2126.48	2113.63	2129.82	2327.75	2111.05	2126.48	2113.63	2129.82	2327.75	2111.05	2126.48	2113.63	2129.82	2327.75	2111.05	2126.48
	PMX	Swap	2225.72	2256.56	2518.3	2204.1	2297.17	2225.72	2256.56	2518.3	2297.173	2204.1	2225.72	2256.56	2518.3	2297.173	2204.1	2225.72	2256.56	2518.3	2297.173	2204.1
		Inversion	2113.65	2124.72	2307.36	2113.69	2121.3	2113.65	2124.72	2307.36	2113.69	2121.3	2113.65	2124.72	2307.36	2113.69	2121.3	2113.65	2124.72	2567.54	2113.69	2121.3
FPS	Cycle	Swap	2344.12	2124.72	2718.76	2361.1	2408.03	2344.12	2124.72	2718.76	2361.1	2408.03	2344.12	2124.72	2718.76	2361.1	2408.03	2344.12	2124.72	2718.76	2361.1	2408.03
		Inversion	2325.01	2433.13	2567.54	2311.1	2301.77	2325.01	2433.13	2567.54	2311.1	2301.77	2325.01	2433.13	2567.54	2311.1	2301.77	2325.01	2433.13	2668.45	2311.1	2301.77
	PMX	Swap	2340.2	2246.23	2668.45	2464.2	2344.24	2340.2	2246.23	2668.45	2464.2	2344.24	2340.2	2246.23	2668.45	2464.2	2344.24	2340.2	2246.23	2668.45	2464.2	2344.24
		Inversion	2485.2	2301.2	2444.27	2497.2	2387.06	2485.2	2301.2	2444.27	2497.2	2387.06	2485.2	2301.2	2444.27	2497.2	2387.06	2485.2	2301.2	2444.27	2497.2	2387.06
Ranking	Cycle	Swap	3523.53	3634.17	3540.19	3498.38	3577.17	3523.53	3634.17	3540.19	3498.38	3577.17	3523.53	3634.17	3540.19	3498.38	3577.17	3523.53	3634.17	3540.19	3498.38	3577.17
		Inversion	3510.9	3566.48	3538.86	3468.14	3537.82	3510.9	3566.48	3538.86	3468.14	3537.82	3510.9	3566.48	3538.86	3468.14	3537.82	3510.9	3566.48	3538.86	3468.14	3537.82
	PMX	Swap	3531.44	3518.4	3480.88	3436.6	3444.14	3531.44	3518.4	3480.88	3436.6	3531.44	3531.44	3518.4	3480.88	3436.6	3531.44	3531.44	3518.4	3480.88	3436.6	3531.44
		Inversion	3501.15	3487.33	3384.38	43356.29	3401.41	3501.15	3487.33	3384.38	43356.29	3401.41	3501.15	3487.33	3384.38	43356.29	3401.41	3501.15	3487.33	3384.38	43356.29	3401.41

Note: Each colour indicates a combination of parameters for each configuration.

In the end we selected and graphed the 12 best configuration that obtained the least total distances, obtaining a best solution of 2116,63 as a result of Tournament – Cycle - Inversion- Mutation rate (0,1)- Probability of the cross over (0.9) – Population (50) - Generation (250).



Configurations:
1 Tournament – Cycle - Inversion- Mutation rate (0,1)- Probability of the cross over (0.9) – Population(50)- Generation (250)
2 Tournament – PMX - Inversion- Mutation rate (0,1)- Probability of the cross over (0.9) – Population(50)- Generation (250)
3 FSP - Cycle - Swap- Mutation rate (0,1)- Probability of the cross over (0.8) – Population (25)- Generation (250)
4 Tournament – Cycle- Swap- Mutation rate (0,1)- Probability of the cross over (0.8) – Population(50)- Generation (250)
5 FSP - PMX - Swap- Mutation rate (0,1)- Probability of the cross over (0.8) – Population (25)- Generation (250)
6 Tournament – PMX- Swap- Mutation rate (0,1)- Probability of the cross over (0.9) – Population(50)- Generation (250)
7 FSP - PMX - Inversion- Mutation rate (0,1)- Probability of the cross over (0.8) – Population(25)- Generation (250)
8 FSP - Cycle - Inversion- Mutation rate (0,2)- Probability of the cross over (0.8) – Population(20)- Generation (200)
9 Ranking – PMX -inversion -mutation rate (0,25)- Probability of the cross over (0.95) – Population(10)- Generation (100)
10 Ranking – PMX - swap -mutation rate (0,2)- Probability of the cross over (0.9) – Population(50)- Generation (250)
11 Ranking – cycle - swap -mutation rate (0,2)- Probability of the cross over (0.9) – Population (50)- Generation (250)
12 Ranking – cycle -inversion -mutation rate (0,25)- Probability of the cross over (0.95) – Population (50)- Generation (250)

What could be improved?

Next time we would like to add the name of the cities associated with the distances so that we can build a route graphic for the best solution, and also add some restrictions to our routes to make the problem more challenging.

Division of work

Ramzi – Implemented the code for Tournament and FPS selection methods, mutation, and crossover.
Vera and Lorena – Code for the Ranking algorithm, experimented with different parameters across all configurations and wrote the whole report.
Sami Mefteh, m20211308 – was also initially in the group, but he did not contribute with any work for the project.

Sources

- https://ayhandemiriz.com/SakaryaWebSite/papers/benelearn09.pdf
- https://www.realcode4you.com/post/genetic-algorithm-for-traveling-salesman-problem-tsp-in-python-ml-sample-text-paper
- https://towardsdatascience.com/how-to-define-a-fitness-function-in-a-genetic-algorithm-be572b9ea3b4
- https://aip.scitation.org/doi/pdf/10.1063/1.5039131
- https://aip.scitation.org/doi/pdf/10.1063/1.4993023
- https://www.researchgate.net/publication/335094833_Creating_a_fitness_function_that_is_the_right_fit_for_the_problem_at_hand