A CIFO PROJECT

ON THE

TRAVELING SALESPERSON PROBLEM

Team members:

Eliane Gotuzzo m20210996,
Marcos Oliveira m20210593,
Tiago Seca m20210564.

NOVA Information Management School
Master's Degree in Data Science and Advanced Analytics
2021/2022

GitHub link: https://github.com/Elijacobsen/CIFO_PROJECT_GROUP_AB

About the Project:

The scope of this project is to develop a genetic algorithm 'GA' to be able to solve a traveling salesperson problem, for that, the GA will receive many different combinations of parameters along with different methods for selection, crossover, and mutation.

In order to test the efficacy of our algorithm, we have chosen 3 datasets - berlin52, xqf131, pcb442 the first one can be considered small with only 52 locations, the next one is of medium size and has 131 locations, and the third and last consist of a large dataset with 442 locations, respectively.

The measurement of best or worse is directly related to the fitness of the individuals (in this case, the route itself). In the genetic algorithm we created many generations and out of these generations we detected the best individuals with the fitness function.

Methodology:

Our methodology consists of applying selection, crossover, and mutation methods. After a given configuration has been defined, we run it 10 times to analyze how that configuration performed to bring us a possible (individual) route with the lowest fitness in its generation.For selection, we coded 3 methods (random, fps and tournament), 3 mutation methods (swap, inversion and exponential) and 3 crossover methods (cycle, switch and pmx).

We have come up with the following operators that were not given in class, which are: random_selector which randomly picks an individual out of the population. Switch_crossover that switches between 25% to 75% of the total elements between the parents. And the last created was the Exponential_mutation which is similar to swap mutation, but it is done for a random number of individuals.

Why did you choose the representation you did?

Our individuals are a list, and this list will contain their values, each value represents a city, and while on the route, you cannot pass by the same city again. This list size will be the same number of cities, and that is because the route must go throughout all the cities.

How did you design the fitness function? Did you try using different fitness functions to see the impact on your GA?

As already mentioned, the solution for our problem involves trying to find the shortest route. The fitness of the route can be assessed with the total distance traveled based on the distance matrix. Due to the nature of our problem, no other types of fitness functions were tried, as the one implemented perfectly satisfied our need with this sort of problem.

Which configurations worked best together? How many did you try, and how did you determine the "best" one?

We tried 27 (3 x 3 x 3) configurations for each dataset. We ran each configuration for 500 generations. Although this might not be enough for some configurations to converge, after debating between ourselves, we considered that this was still a viable alternative, because the "best" configurations tend to converge rapidly, so 500 generations is enough to evaluate the different combinations.

According to our findings, we have gotten our best results using the methods tournament for selection. Now talking about the crossover methods, the pmx and cycle has shown us some impressive results. And the inversion mutation has fared better in comparison to other mutation methods. The random selector and the switch crossover performed constantly the worst. Finally, whenever we ran our trials, having elitism on has given us better performances in contrast to when it was off. Below we can see the tables with different combinations and their respective results:

Selector	Crossover	Mutation	Time	Best_Fitness	Avg_Fitness	Std_Fitness
tournament	cycle_co	inversion_mutation	0.27	8088	8461	320
tournament	switch_crossover	inversion_mutation	0.44	8101	8509	327
tournament	pmx_co	inversion_mutation	0.36	7991	8576	263
tournament	switch_crossover	swap_mutation	0.43	10155	10683	490
tournament	pmx_co	swap_mutation	0.35	10246	10991	707
tournament	cycle_co	swap_mutation	0.28	10234	11261	665
tournament	switch_crossover	exponencial_mutation	0.44	10854	11655	571
tournament	pmx_co	exponencial_mutation	0.36	11599	12319	643
fps	cycle_co	inversion_mutation	0.44	9795	12506	1006
tournament	cycle_co	exponencial_mutation	0.28	11924	12847	542
fps	cycle_co	swap_mutation	0.42	12905	14005	1122
fps	cycle_co	exponencial_mutation	0.43	13240	14991	842
random_sel	cycle_co	swap_mutation	0.25	14786	15771	743
random_sel	cycle_co	inversion_mutation	0.25	14496	16173	910
fps	pmx_co	swap_mutation	0.55	14329	16222	1239
fps	pmx_co	inversion_mutation	0.56	15931	17248	1055
random_sel	cycle_co	exponencial_mutation	0.25	15932	17518	686
fps	pmx_co	exponencial_mutation	0.56	15775	17534	1057
random_sel	pmx_co	inversion_mutation	0.37	16648	18510	865
random_sel	pmx_co	swap_mutation	0.37	18606	19387	769
random_sel	pmx_co	exponencial_mutation	0.37	17691	19387	1321
fps	switch_crossover	swap_mutation	0.59	17850	19502	924
fps	switch_crossover	exponencial_mutation	0.60	16665	19672	1384
fps	switch_crossover	inversion_mutation	0.61	19176	21015	797
random_sel	switch_crossover	swap_mutation	0.41	19431	21066	923
random_sel	switch_crossover	exponencial_mutation	0.41	20412	21415	513
random_sel	switch_crossover	inversion_mutation	0.41	20620	21468	518

(Table 1. Different combinations for the berlin52 dataset)

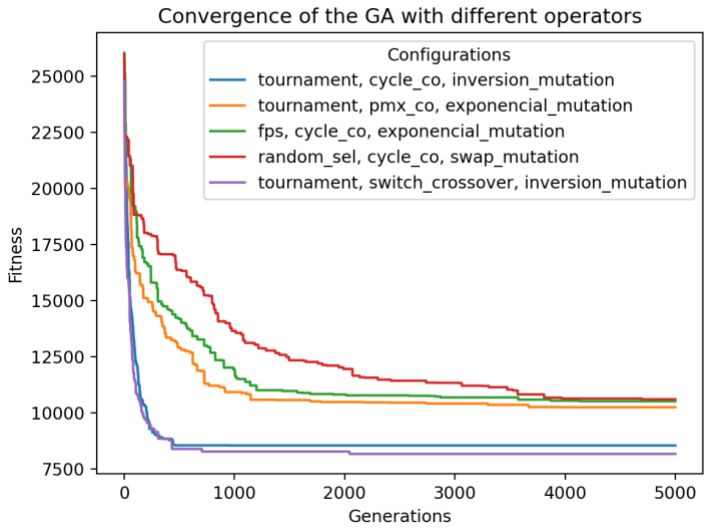
Selector	Crossover	Mutation	Time	Best_Fitness	Avg_Fitness	Std_Fitness
tournament	pmx_co	inversion_mutation	10.78	900	951	27
tournament	cycle_co	inversion_mutation	0.62	1074	1146	38
tournament	switch_crossover	inversion_mutation	10.73	1071	1148	45
tournament	pmx_co	swap_mutation	10.71	1471	1605	83
tournament	switch_crossover	swap_mutation	10.63	1538	1633	55
tournament	cycle_co	swap_mutation	0.62	1432	1659	107
tournament	pmx_co	exponencial_mutation	10.84	1913	1992	58
tournament	switch_crossover	exponencial_mutation	10.68	1729	2017	129
tournament	cycle_co	exponencial_mutation	0.62	2020	2141	76
fps	cycle_co	inversion_mutation	0.71	2300	2514	129
fps	cycle_co	swap_mutation	0.71	2462	2595	123
fps	cycle_co	exponencial_mutation	0.71	2439	2638	125
random_sel	cycle_co	swap_mutation	0.59	2669	2762	81
random_sel	cycle_co	inversion_mutation	0.59	2588	2768	166
random_sel	cycle_co	exponencial_mutation	0.60	2836	2961	85
fps	pmx_co	swap_mutation	12.45	2963	3116	97
fps	pmx_co	exponencial_mutation	12.57	2996	3215	123
fps	pmx_co	inversion_mutation	1.25	3052	3260	122
random_sel	pmx_co	inversion_mutation	1.14	3162	3313	139
random_sel	pmx_co	swap_mutation	11.37	2967	3319	156
random_sel	pmx_co	exponencial_mutation	11.43	3131	3453	176
fps	switch_crossover	swap_mutation	11.57	3309	3596	124
random_sel	switch_crossover	swap_mutation	10.45	3584	3643	60
fps	switch_crossover	exponencial_mutation	1.17	3510	3661	96
fps	switch_crossover	inversion_mutation	11.61	3506	3680	81
random_sel	switch_crossover	inversion_mutation	10.66	3605	3701	62
random_sel	switch_crossover	exponencial_mutation	10.51	3559	3702	71

(Table 2. Different combinations for the xqf131 dataset)

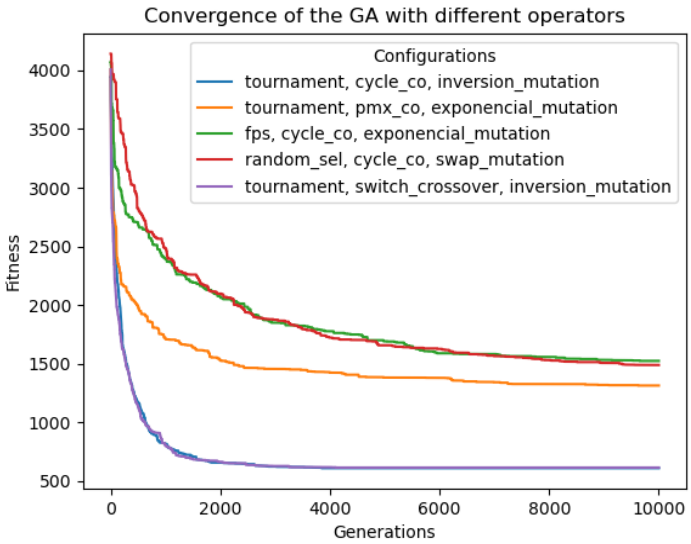
Selector	Crossover	Mutation	Time	Best_Fitness	Avg_Fitness	Std_Fitness
tournament	pmx_co	inversion_mutation	3.79	376970	387268	6146
tournament	pmx_co	swap_mutation	3.78	448291	463099	11081
tournament	switch_crossover	swap_mutation	33.35	474002	487091	8945
tournament	pmx_co	exponencial_mutation	38.38	492565	511233	8393
tournament	switch_crossover	inversion_mutation	3.35	506110	522935	12833
tournament	cycle_co	swap_mutation	17.10	500177	525380	12757
tournament	cycle_co	inversion_mutation	17.21	508372	525844	8896
tournament	switch_crossover	exponencial_mutation	3.35	511743	530576	11616
tournament	cycle_co	exponencial_mutation	17.28	546022	563039	9868
fps	cycle_co	swap_mutation	17.40	631221	648045	11382
fps	cycle_co	exponencial_mutation	17.27	634389	648750	9020
random_sel	cycle_co	swap_mutation	16.95	634962	663419	16189
fps	cycle_co	inversion_mutation	17.16	649054	669729	11904
random_sel	cycle_co	exponencial_mutation	16.79	661487	671566	8017
random_sel	cycle_co	inversion_mutation	16.77	661267	678020	9449
fps	pmx_co	exponencial_mutation	40.62	671441	680207	5463
random_sel	pmx_co	swap_mutation	4.01	670850	682946	8640
fps	pmx_co	swap_mutation	4.07	669643	683651	7087
random_sel	pmx_co	exponencial_mutation	39.94	677163	686664	6697
fps	pmx_co	inversion_mutation	4.05	664079	686733	9408
random_sel	pmx_co	inversion_mutation	40.02	679256	689175	5577
fps	switch_crossover	exponencial_mutation	3.40	694008	701410	5656
random_sel	switch_crossover	swap_mutation	33.35	697019	703612	4697
fps	switch_crossover	swap_mutation	3.37	697145	703669	3581
random_sel	switch_crossover	inversion_mutation	3.34	692143	703844	6136
random_sel	switch_crossover	exponencial_mutation	3.35	693670	704962	6873
fps	switch_crossover	inversion_mutation	33.66	702330	707610	4084

(Table 3. Different combinations for the pcb442 dataset)

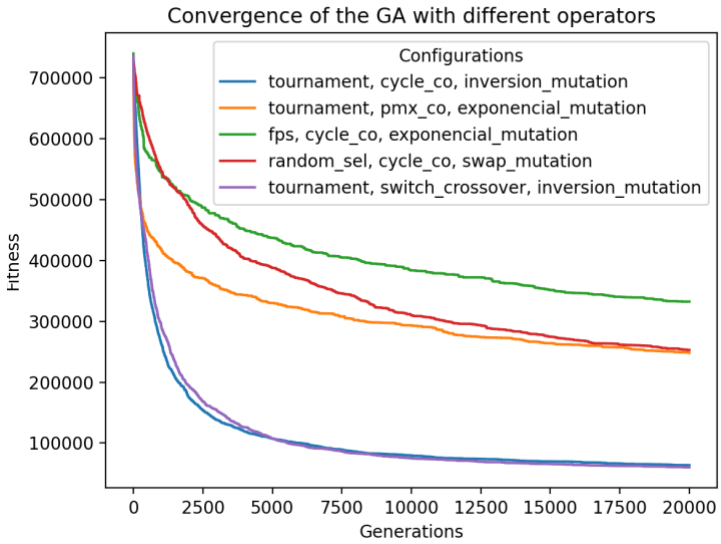
Do different operators affect the convergence of your GA?



(Figure 1. Different operators for the berlin52 dataset)



(Figure 2. Different operators for the xqf131 dataset)



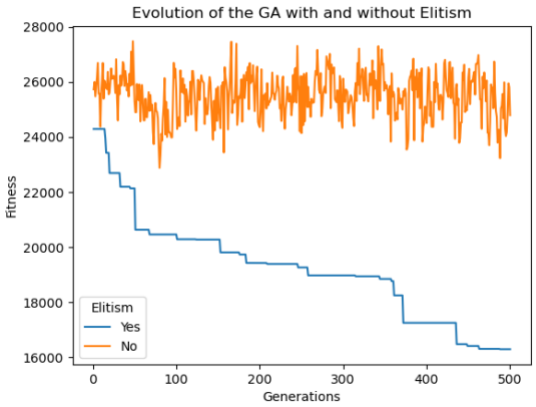
(Figure 3. Different operators for the pcb442 dataset)

As we can see based on the graphics above (Figure 1, 2 and 3) which were run on different datasets, the information they bring us is that different operators affect how early the algorithm reaches convergence. It is also clear in the graphics that the bigger the dataset, the longer it will take to converge. During this test, it was noticeable that regardless of the dataset used, the blue and purple configurations converged in a different place compared to the other 3 configurations. So, ultimately the answer is that yes, the use of different operators might affect the convergence of the GA.

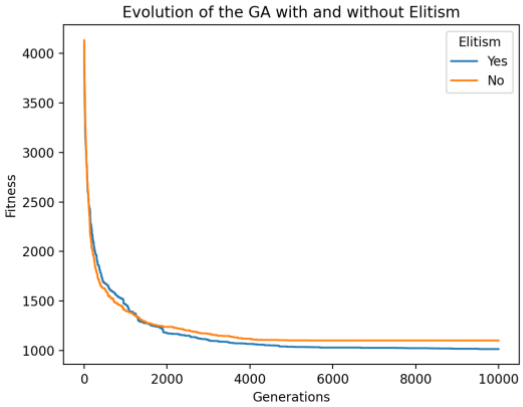
Have you implemented elitism? Does the inclusion or exclusion of elitism impact your GA?

Yes, we did. The elitism we implemented consisted of substituting the worst individual with the best one, the 'elite'. Implementing elitism, by far, has shown us better results than when not applying it. This was particularly evident by how fast the algorithm converged to optimum when

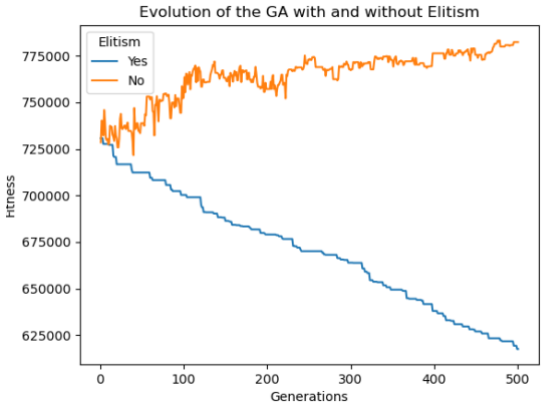
the elitism was being implemented. To better illustrate our findings, we have plotted some graphs below to show the difference of with and without elitism.



(Figure 4. Elitism on berlin52 dataset)



(Figure 5. Elitism on xqf131 dataset)



(Figure 6. Elitism on the pcb442 dataset)

The figures above show 3 different configurations, one in each dataset, and we can confidently say that including elitism impacts the algorithm in a positive way. Even on figure 5 that appeared that without Elitism the GA performed better, the convergence was still worse than with Elitism. We tried many different configurations on different datasets, and the results are clear, we have gotten better outcomes in the scenarios where elitism was included.

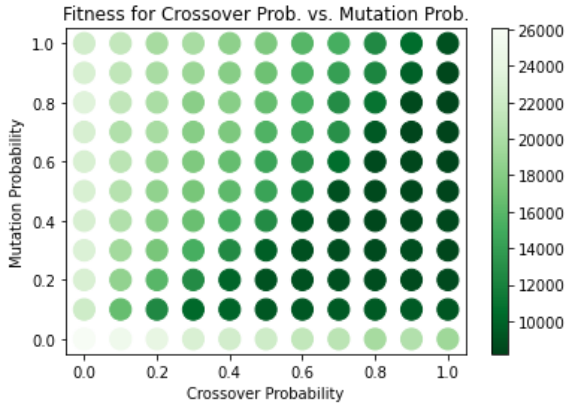
Are your implementations abstract and work with both minimization and maximization?

Yes. During the coding phase we were cautious about our code, and it worked for both cases.

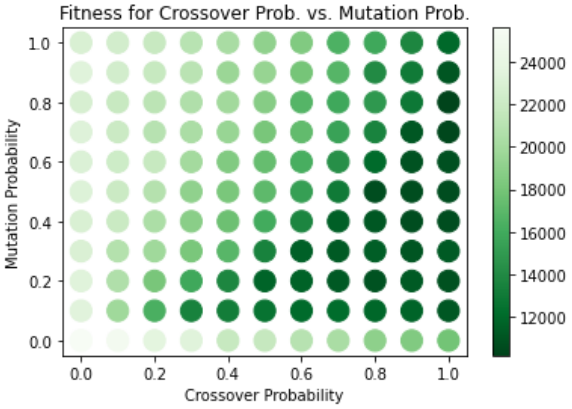
Do you get good results for the project you chose? What could be improved?

Although we obtained good results, we could've gathered better results if we had increased the number of generations for the bigger datasets, that most of the times the GA didn't converge in a reasonable time. Considering this, naturally, with more powerful equipment this situation would've been overcome, and we could increase the number of generations that would lead to a result, perhaps a bit closer to a global optimum. Even with that limitation, we think that our project has achieved satisfactory results. And concerning elitism, we might've had better results if it was implemented for more than one individual.

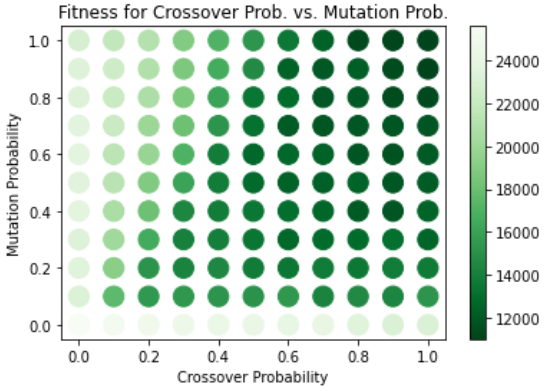
How high (or low) should the crossover and mutation probability be?



(Figure 7. tournament, pmx crossover, inversion mutation)



(Figure 8. tournament, switch crossover and swap mutation)



(Figure 9. tournament, cycle crossover and, exponencial mutation)

Here we have checked how our algorithm behaves, and the fitness is affected with different crossover and mutation probabilities applied to different configurations, also it is worth saying that we have tried this only on the berlin52 dataset, and that was because of the time it took to run these three tests on it, it the same thing was to be applied on bigger datasets, it would take an eternity to run it. Nonetheless we are content with our findings.

Generally, from these graphics we can see that the more crossover we do, the better fitness we get. Now, concerning mutation, it seems it doesn't affect our results when crossover is high. Basically, with a high crossover, the mutation rate does not affect our results, whether it is set high or low, but if the crossover probability is low, the results are more sensitive to how high the mutation probability is. Except when using Exponential Mutation, it seems that higher values for the mutation probability (when the crossover probability is also high) tends to perform better (Figure 9).