Katholieke Universiteit Leuven
Master of Artificial Intelligence

Genetic Algorithms Project: Travelling Salesman Problem

Aymeric Charles de la Brousse	r0735815
Guillaume Lamine	r0737163

Professors : Dirk Roose

2018 - 2019

1 Introduction

For this paper we have been asked to analyze and improve the Matlab code of a Genetic Algorithm to solve the *Travelling Salesman Problem* which is N-P complete. In addition to the general framework asked for this paper, we have decided to choose task (a)*implement two new parental selections* and (b) *implement one survivor selection strategy.*

This paper will be divided into several sections. Section 2 are the early experiments. This part helps us get familiar with the code and how certain parameters affect the algorithm's performance. Section 3 will let us explore the improvements we can make to the existing algorithm and ways to tune the parameters. Finally, in section 4, we benchmark our chosen algorithm to one of the proposed datasets.

Concerning the vocabulary used in this paper, the instance of a genetic algorithm refers to a GA with a particular set of parameters (numeric or symbolic). Also from hereon the first version of the code that was given as basis of this project will be denoted as the *original code.* Finally, we use the objective value to determine which chromosomes are the best. The objective value is the sum of all distances between the cities and the goal of this paper is thus to conceive an algorithm that can find the minimum objective value for a dataset of cities.

2 Early experiments

In this section, our goal is to manipulate the different parameters of the genetic algorithm in order to observe the changes in performance. Because we do not plan to do some exhaustive parameter tuning, we only plan to change one parameter at a time and describe the changes.

Before starting, let's discuss how we are going to evaluate the utility (performance) of our different parameter vectors. One method suggested by Eiben and Smit (Eiben & Smit, 2012) consists of setting a maximum running time and extract the shortest path obtained at termination. After a few experiments we decided to set the maximum time at 20 seconds. Also, because GAs are stochastic, it would best to perform the algorithm at least several time, store the best objective value at each run and calculate the mean of all minimum objective values. This method works the same way as the Mean Best Fitness. We also decide to focus our experiments on only one of the available datasets to be able to compare our experiments, the "rondrit050.tsp".

In order to gain some additional insights from the experiments we decided to measure the diversity of the population at termination. The reason is that diversity has a significant importance in GAs as they show that the algorithm had a broad search over all the search space. Furthermore, If we are not able to keep the diversity high, we risk to fall into a local optimum. This phenomenon is called premature convergence. Diversity will encourage the exploration phase for a global optimum of the search and selection will encourage the exploitation phase for a local optimum. Measuring the diversity of the population is a complex task. Koza (Koza, 1992) described the diversity as the "percentage of individuals for which no exact duplicate exists elsewhere in the population". Eiben and Smith (Eiben & Smith, 2015) described it as the number of different solutions present and insisted that "no single measure for diversity exists". A proxy of diversity is already computed in the original code. A method to implement a stopping criteria is present and which works by calculating the difference of fitness value between the best chromosome and the 5^{th} percentile chromosome of the population. However, in this paper we choose to approximate the diversity by computing the percentage of unique objective values in the current population. This will help us analyze our results.

We run these experiments fully aware that some of these results will not perform well. For instance, it is well known that the probability of crossover works the best in the 0.5 to 1 range in order to keep diversity and avoid premature convergence. Setting the crossover probability to 0, helps us understand the importance of that parameter on the performance of the algorithm.

2.1 Methodology

- The experiments were completed on the "rondrit050.tsp" dataset
- We decided to only change the value of one parameter at a time to keep the experiments non-exhaustive
- We kept the default values of the original code
- We decided to keep the local loop removal activated

- we set a 20 seconds timer at the end of which the algorithm terminated
- each instance of the algorithm was performed 10 times to reduce randomness in the evaluation process
- the maximum number of generation is actually in the original code as a limit that you have to be lower than. This means that for a limit of 100, the instance will converge at 99.

2.2 Population size

We perform the algorithm using a population of 10, 50, 150, 500 and 1000. The parameter vectors can be seen in Table 5. The results are shown in Table 6. After performing the experiments, there are a few observations that we can already make.

We notice that when the population is set at its lowest, the population's objective values doesn't consistently improve over time. It seems that there are high chances that the best fitness value gets replaced after the going through the variation operators.

We notice that the algorithm typically performs better as the population size grows. This is due to the fact that a higher initial number of individuals will help us achieve higher levels of diversity to explore more of the search space. Unfortunately, a higher population size also indicates a higher time to convergence. Using the graph in Figure 3 and the results Table 6 we can see that at around 150 individuals the algorithm manages to converge before the end of the timer, the diversity was kept high and the mean best objective value is satisfying as well.

2.3 Number of generations

We perform the algorithm using a maximum generation number of 10, 50, 150, 500 and 1000. The parameter vectors can be seen in Table 7. The results are shown in Table 8 and the graph is shown in Figure 4. While experimenting on the number of generations, we removed the timer limit in order to let it converge to its maximum but still placed a timer to store the speed of each instance. We can see that the mean best objective value decreases rapidly in the beginning but stabilizes after passing 50 generations. A maximum number of generation between 50 to 150 appears to be an appropriate limit considering the time to converge as well.

2.4 Crossover probability

We perform the algorithm using a crossover probability of 0, 0.2, 0.5, 0.8 and 1. The parameter vectors can be seen in Table 9. The results are shown in table 10 and the graph in Figure 5. We can see in our results that the best objective values were obtained with a crossover probability of 0.5 but the algorithm converged early as hardly any diversity could be retained. In a bigger search space containing more local optima, a crossover of 0.5 might not be enough and it would be best to rely on a P_c of around 0.8 where a certain degree of diversity is kept.

2.5 Mutation probability

We perform the algorithm using a mutation probability of 0, 0.2, 0.5, 0.8 and 1. The parameter vectors can be seen in Table 11. The results are shown in table 12 and the graph in Figure 6. The performance of the algorithm seems to be pretty constant whatever the mutation probability. We only notice a drop in performance at $P_m = 1$ where it also took 25% more time to complete all generations.

2.6 Elitism percentage

We perform the algorithm using an elitism percentage of 0%, 20%, 50%, 80% and 100%. The parameter vectors can be seen in Table 13.The results are shown in table 14 and in Figure 7. We observe that a proper percentage of elitism should certainly not be equal to 0 and should be around 20%. Here we discovered that the 100% elitism actually does not mean that 100% of the initial population was going to be kept as we assumed. Indeed, the original code has implemented a safe stop at 98% in order to allow the population to evolve in any case. We can still notice that as we approach the higher end of the parameter value that the performance drops as the algorithm does not allow for enough diversity. Furthermore, the goal of elitism is to tamper diversity so that we keep the good local optima that we obtain during each run. But elitism also causes the GA to converge on local maxima instead of the global maximum. This is again a balance between exploration of the search space and exploitation of our local optima.

3 Improvements

3.1 Stopping Criterion

The original code already had a stopping criterion. Its goal was to stop the algorithm when the diversity of the chromosomes became too low. Here they approximated the diversity by calculating the difference between the chromosome with the best fitness value and the one sitting at the 5th percentile (meaning only 5% of the population had a worse fitness value). If the difference between them is lower than a certain number, the algorithm stops.

We decided to add another one. The goal of the new stopping criterion is to stop the algorithm if for a certain number of generations we have not seen any improvement in the best tour. After a few tests, we decided to set the number of generations at 50.

3.2 Representation

The representation implemented in the original code was the adjacency representation. As an alternate representation we decided to enforce the path representation. Fortunately for us, the representation was already available in the template, which made the implementation trivial.

1. *Adjacency representation*: Representation implemented in the original code. This representation is essentially used for the travelling salesman problem as it allows to minimize the number of identical tours in the population.

2. *Path representation*: It is the most natural way of representing a tour the travelling salesman problem. In this case the phenotype and genotype are almost equivalent. Hence, there is a chance that multiple representations of the same tour appear in the population. The original code also contained a function to improve our path representation and remove local loops.

3.3 Crossover

The alternating edges crossover was already implemented in the original code and as this operator can only handle adjacency representation, we decided to implement a new crossover.

1. *Alternating Edges Crossover (AEX)*: The offsprings are created by choosing in alternation edges from the first and the second parent. If the selection of an edge results in an early closure of the tour, another edge is chosen at random.

2. *Order Crossover (OX)*: We choose a random segment of cities in the first parent and donate to a child. We mark the donated cities in the second parent. We now fill in the child with the cities that are not marked in the second parent in order of their appearance (left to right). We do the same thing for the second child but by reversing the roles of the first and second parent. The crossover is illustrated in Figure 1.

3.4 Mutation

The mutation operators implemented in the original code can already handle our new path representation. These two operators are:

1. *Reciprocal Exchange*: This mutation simply randomly chooses two cities in the chromosome and swaps their positions.

2. *Simple Inversion*: This mutation randomly chooses two cities in the chromosome and reverses the order between them.

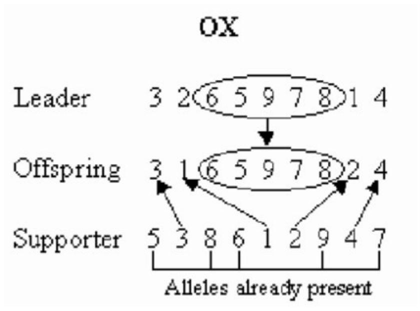


Figure 1: The OX operator. Here the supporter is the second parent and the leader is the first parent. *(Source: Scheduling and Production & Control: MA by Moscato, P., Mendes, A. and Cotta, C.)*

3. *Cut Inversion*: This is an extension of the simple inversion. Here the reversed segment is also replaced randomly in the chromosome. This will increase the degree of diversity in the GA.

In a concern to improve our algorithm we decided to implement and test another mutation algorithm which is an extension of the inversion operator. This operator called *cut-inversion* also chooses two cities randomly and reverses the order of passage. The difference here is that the selected segment will also be moved to another randomly chosen position in the chromosome. This will definitely bring more diversity in the algorithm to explore new undiscovered solutions in the search space.

3.5 Parameter Tuning

In this section we decided to test out different values for our numerical values while we fixed path as the representation, OX as the crossover and cut-inversion as the mutation.

We proceeded the same way as for our early testing phase after running the algorithm 10 times for each meta-parameter vector, we computed the mean values retrieved for:

1. Best Objective Value: We use the mean best objective value of 10 iterations of the algorithm to evaluate the performance of each parameter vector.
2. Diversity: The diversity is computed the same way here than in the earlier experiments. We use Matlab's *unique(Array)* function to determine the proportion of unique fitness values among the population.
3. Number of generations: This provides us with an indication of the moment at which the algorithm was terminated.
4. Timer: We assume that the time that the algorithm takes could be an interesting factor to look at.

The results can be seen in Table 1. Graphs illustrating the effect of changes in population sizes and elitism percentages can be seen in Figure 8.

To limit the number of tests, we analyzed the results of the early experiments in order to agree upon reference values. We then explore around the reference values by only changing one parameter at a time. Finally, we decide to settle on $P_m = 0.2$, $P_c = 0.95$, $Elitism = 0.2$, $PopulationSize = 200$.

3.6 Parental Selection

In the original code, the Stochastic Universal Sampling is already implemented with an rank-based conversion. The code for the Roulette Wheel Selection is also available in the template but not used. We improve these two selection operators with a scaling function and we also introduced a new selection operator:

1. *Roulette Wheel Selection with fitness scaling*: The simplest method to achieve parental selection is called Roulette Wheel Selection. Each individual's probability to be selected is proportional to their fitness value so that the best tours have more chances to be selected for reproduction. It basically assigns a probability

to be selected to each individual based on their share of the total fitness of the population. For RWS scaling is important as we don't want the best fitness values to overtake the rest and to keep diversity in the population. We can decide to scale or fitness functions so that the best individuals don't necessarily overtake the parental pool. Weaker candidates don't necessarily mean that they don't have anything to add to the chromosome pool. Scaling will actually diminish the selective pressure as it gives weaker individuals more chances to get selected.

2. *Stochastic Universal Sampling with fitness scaling*: Stochastic Universal Sampling follows the same principle as roulette wheel selection. If we reuse the roulette metaphor, SUS is like a roulette wheel selection but instead of using one arrow we use one for each individuals that we have to select. The angle between each arrow is identical to all. Same as the RWS, we can also decide to scale the fitness values prior to the selection.

3. *K-Tournament selection*: In a tournament selection there is no need for scaling as we only care about the rank of each individual. The selection proceeds as follow: at each round, we select K individuals and the one with the highest fitness function is selected as a parent. We start over until we reach our maximum number of parents. A K=1 tournament selection is basically a random selection of parents. The convenient part of tournament selection is that we can easily change the selective pressure by adjusting the K. As K increases, weaker individuals have lesser chances to get selected for mating. A K=number of individuals would result to all parents being the same as the fittest individual of the population.

In order to evaluate the performance of each parental selection, we keep the numerical parameters we have chosen in the tuning phase, our new crossover and mutation and run several iterations. The results can be seen in Table 2. As usual, 10 iterations are performed for each set of parameters. We initially decided to test K = 2,4 and 6 but decided to add K = 10 because we were not satisfied with the resulted mean best objective value. Overall, the tournament algorithm performed poorly despite its good diversity. We finally decided to select the original algorithm with is the rank-based stochastic universal sampling.

3.7 Survivor Selection

The original code reinserted the results of the variance operators using the elitism and GGAP percentages. The elitism here referred to the percentage of the population that would be kept in the next generation. The GGAP is equal to 1-Elitism and refers to a percentage of the population size. It will be used as the maximum amount of parents selected for crossover. Because the number of newly created offsprings and the number of kept individuals from last generation would always be equal to the population size, there wouldn't be any competition for reinsertion. In this paper we thus introduce a new survivor selection to add competition between parents and offsprings:

- *Round Robin Tournament*: We decided to implement the Round Robin tournament selection as described in the text book (Eiben & Smith, 2015). The method assembles the parents and the offsprings of the current generation together to be candidates for the next generation's population. Each of the individuals will be selected to fight in a pool of K randomly selected opponents. The fights are pairwise and a win counter increases for each won fight. When we have fought will all the individuals we the population is comprised of the highest win counters. The round robin tournament is more stochastic than the typical elitism, rank-based selection. The perks of this selection operator is that we can adjust the degree of randomness of the operator by changing the K size of the tournament pool. The higher the size, the harder it will be for a weak individual to get selected. The textbook recommends a size of K = 10.

After testing the new round robin survivor selection strategy for different K's and comparing it to the previous survivor selection, we have computed the results in 9.

We observe that the round robin strategy performs worse that our previous survivor operator. The problem might come from the lack of diversity, which shows for even very small K values. Thus, we decide to keep our previous algorithm.

Experiment Number	Min ObjV	Diversity	Number of Generations	Timer	P_m	P_c	Elitism (%)	Population Size
1	6,0921	0,803	247,9	49,77922	0,10	0,95	15	200
2	6,1484	0,805	275,8	55,44863	0,15	0,95	15	200
3	6,0582	0,8145	293	59,43437	0,20	0,95	15	200
4	6,1642	0,816	288,3	48,80887	0,25	0,95	15	200
5	6,0541	0,544	244,8	37,13803	0,15	0,50	15	200
6	6,0537	0,66	241,3	37,91571	0,15	0,65	15	200
7	6,0954	0,752	263,7	42,7766	0,15	0,80	15	200
8	6,1591	0,7855	276,9	46,41837	0,15	0,95	15	200
9	6.1388	-	290.5	83.1745	0,15	0,95	10	200
10	6.1219	-	216.9	60.1247	0,15	0,95	20	200
11	6.2056	-	260.1	63.1938	0,15	0,95	30	200
12	6.1902	-	288.8	26.1969	0,15	0,95	40	200
13	6.5717	-	282.7	18.1112	0,15	0,95	15	50
14	6.2683	-	292.3	22.2466	0,15	0,95	15	100
15	6.1228	-	286.7	28.6935	0,15	0,95	15	200
16	5.9764	-	272.2	50.6968	0,15	0,95	15	400

Table 1: Numerical parameter tuning with default settings: $P_m = 0.15$, $P_c = 0.95$, $Elitism = 15\%$, $Populationsize = 200$. Unfortunately, we lost part of the diversity values of our parameter tuning during the simulations. We still thought the numbers were relevant enough to add the first half of our diversity values to the table.

Experiment Number	Min ObjV	Diversity	Number of Generations	Timer	Mutation Operator
1	6,1063	0,748	298,6	50,0644	Rank-Based SUS
2	6,1647	0,7775	331	65,9508	Scaling SUS
3	6,2380	0,76	308,1	60,4836	Scaling RWS
4	6,6813	0,8005	373,8	67,3612	2-Tournament
5	7,4207	0,826	290,7	50,9787	4-Tournament
6	7,3454	0,8415	328,6	70,0554	6-Tournament
7	8,0317	0,8225	247,2	43,3822	10-Tournament

Table 2: Results for our different parental selections with previously chosen settings: $P_m = 0.2$, $P_c = 0.95$, $Elitism = 20\%$, $Populationsize = 200$

3.8 Local Optimization Heuristic

When creating the initial population, the standard algorithm as well as the original code creates tours of random cities. In order to increase the performance we decided to look into some techniques to improve the initial population. We found the greedy initial population described by (Rana & Srivastava, 2017) to be a good match.

- *Greedy Initial Population*: Here we choose the first city of the chromosome at random and then look for the 1-nearest neighbor i.e. the closest city to the current one. We then move on to the newly selected city and continue the algorithm until we have a complete chromosome. When the chromosome is completed, we move onto the next chromosome by selecting a new random city.

We compare our algorithm with and without the greedy initial population and compute the results in Table 3.

The most obvious observation is the incredible speed at which the algorithm performs. The number of generation it needs before finding a good solution is impressive as well. Because our algorithm also outperforms the previous one we decide to keep the greedy initial population.

Experiment Number	Max ObjV	Diversity	Number of Generations	Timer	Mutation Operator	Survivor Selection	Greedy Initial
1	6,1063	0,748	298,6	50,0644	Rank-Based SUS	Elitism	No
2	5.9333	0,7325	127.8	12.5849	Rank-Based SUS	Elitism	Yes

Table 3: Implementation of the greedy initial population algorithm.

4 Benchmarking

After tuning and testing our algorithm on a medium-sized city dataset, we decide to test its performance on a bigger dataset. In order to do that we used the benchmark datasets available on Toledo. The results for *xqf131*, *bcl380*, *belgiumtour* and *rbx711* can be seen in Figure 2.

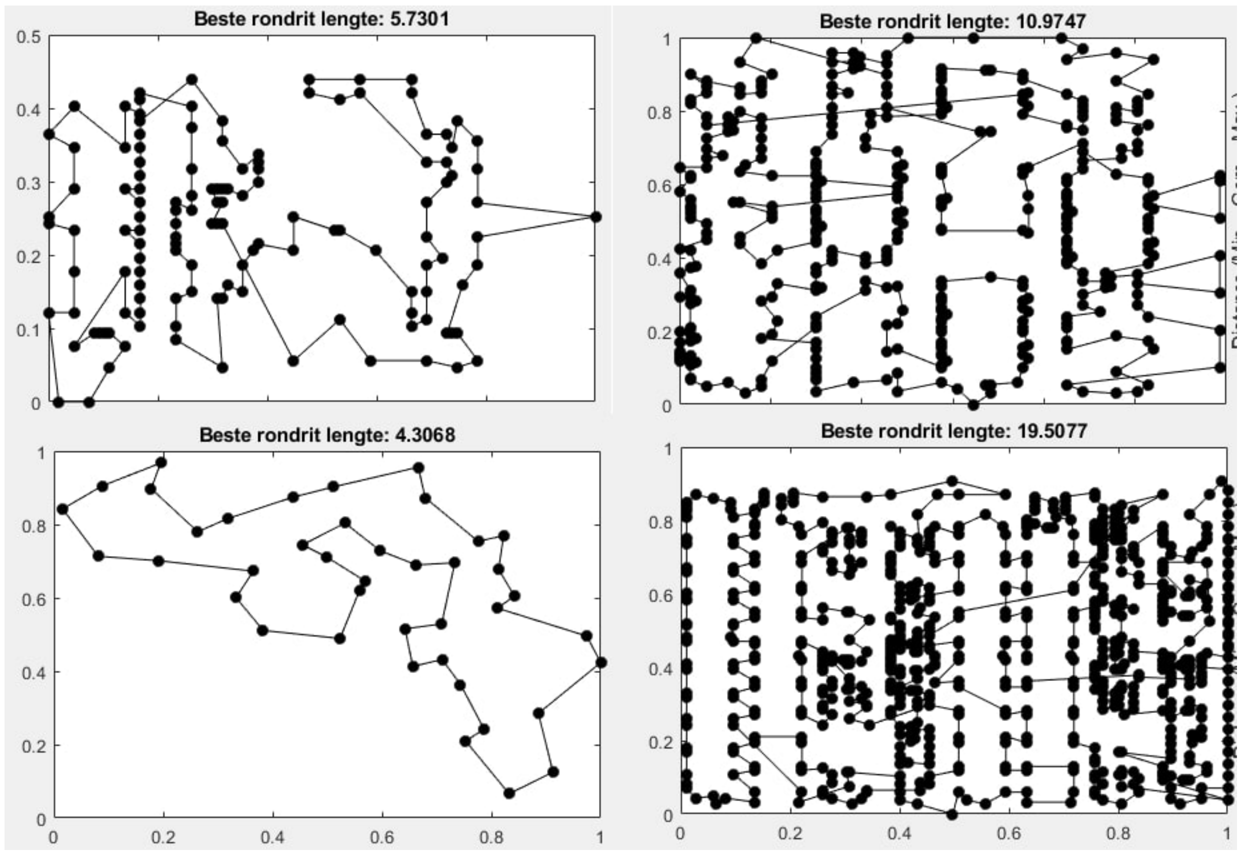


Figure 2: Results of our algorithm. From left to right: *xqf131*, *bcl380*, *belgiumtour* and *rbx711*.

After converting back the path lengths to the real length we obtain the results in Table 4.

Dataset	Our Path	Optimal Path
xqf131	613,12	564
bcl380	1832,78	1621
rbx711	3608,92	3115

Table 4: Our path versus the optimal path

References

Eiben, A. E., & Smit, S. K. (2012). Evolutionary algorithm parameters and methods to tune them. In Y. Hamadi, E. Monfroy, & F. Saubion (Eds.), *Autonomous search* (pp. 15–36). Berlin, Heidelberg: Springer Berlin Heidelberg. Retrieved from `https://doi.org/10.1007/978-3-642-21434-9_2` doi: 10.1007/978-3-642-21434-9_2

Eiben, A. E., & Smith, J. E. (2015). *Introduction to evolutionary computing* (2nd ed.). Springer Publishing Company, Incorporated.

Koza, J. R. (1992). *Genetic programming: On the programming of computers by means of natural selection.* Cambridge, MA, USA: MIT Press.

Rana, S., & Srivastava, S. R. (2017). Solving travelling salesman problem using improved genetic algorithm. *Indian Journal of Science and Technology*, *10*(30). Retrieved from `http://www.indjst.org/index.php/indjst/article/view/115512`

Appendices

A Numerical results : details of the parameters

Numeric Parameters					Symbolic Parameters	
Individuals	Maximum Generations	Probability of Mutation	Probability of Crossover	Elitism (%)	Parent Selection Operator	Crossover Operator
10	100	0.05	0.95	5	Stochastic Universal Sampling	Alternate Edges
50	100	0.05	0.95	5	Stochastic Universal Sampling	Alternate Edges
150	100	0.05	0.95	5	Stochastic Universal Sampling	Alternate Edges
500	100	0.05	0.95	5	Stochastic Universal Sampling	Alternate Edges
1000	100	0.05	0.95	5	Stochastic Universal Sampling	Alternate Edges

Table 5: Parameter vectors for population size

Individuals	Min ObjV	Mean ObjV	Max ObjV	Number of Generations	Diversity
10	12,2135	14,2783	16,1636	90,3	0,71
50	9,6320	14,8246	19,1675	99	0,956
150	8,8812	14,4962	19,2180	99	0,9346
500	8,3636	14,5381	19,9045	91,4	0,9474
1000	8,4506	14,5278	20,2838	80,4	0,9366

Table 6: Performance for different population sizes

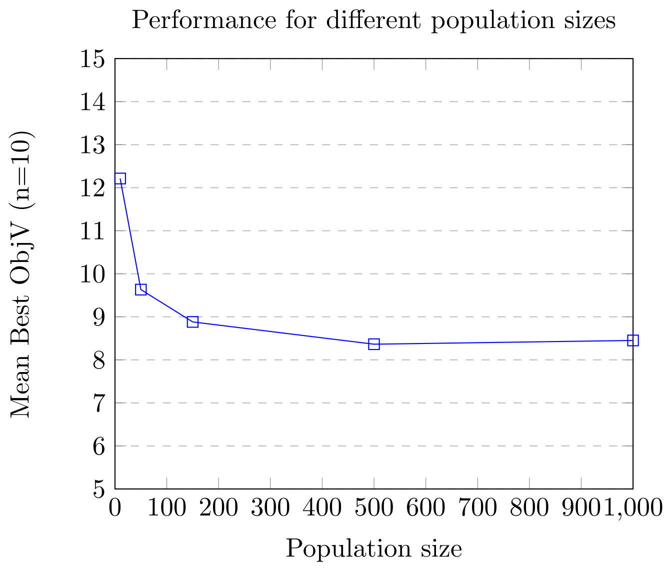


Figure 3: Performance for different population sizes

Numeric Parameters					Symbolic Parameters	
Individuals	Maximum Generations	Probability of Mutation	Probability of Crossover	Elitism (%)	Parent Selection Operator	Crossover Operator
50	10	0.05	0.95	5	Stochastic Universal Sampling	Alternate Edges
50	50	0.05	0.95	5	Stochastic Universal Sampling	Alternate Edges
50	150	0.05	0.95	5	Stochastic Universal Sampling	Alternate Edges
50	500	0.05	0.95	5	Stochastic Universal Sampling	Alternate Edges
50	1000	0.05	0.95	5	Stochastic Universal Sampling	Alternate Edges

Table 7: Parameter vectors for maximum generations

Maximum Generation	Min ObjV	Mean ObjV	Max ObjV	Number of Generations	Diversity	Timer
10	11,4080	15,5098	19,3302	9	0,9940	1,3210
50	9,8992	15,0606	18,6902	49	0,9800	6,5958
150	9,4212	14,8906	18,7133	149	0,9680	24,9586
500	8,7432	14,5877	18,3735	499	0,9540	73,0552
1000	8,7086	14,8070	18,7275	999	0,9660	168,5179

Table 8: Performance for different maximum generations

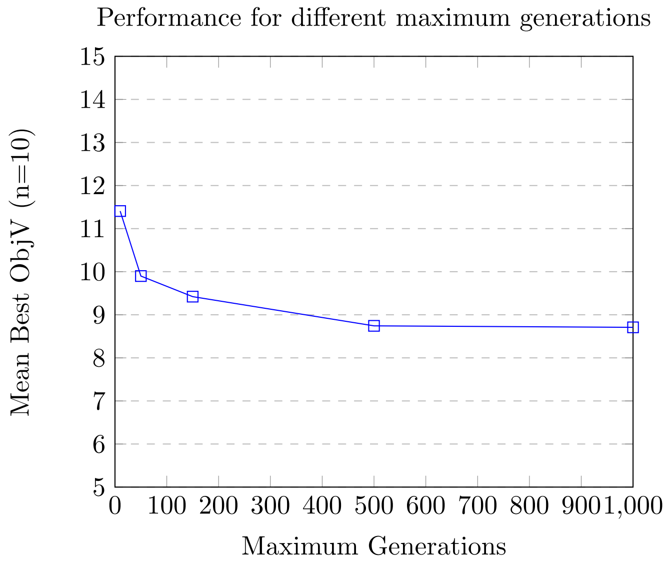


Figure 4

Numeric Parameters					Symbolic Parameters	
Individuals	Maximum Generations	Probability of Mutation	Probability of Crossover	Elitism (%)	Parent Selection Operator	Crossover Operator
50	100	0.05	0.00	5	Stochastic Universal Sampling	Alternate Edges
50	100	0.05	0.20	5	Stochastic Universal Sampling	Alternate Edges
50	100	0.05	0.50	5	Stochastic Universal Sampling	Alternate Edges
50	100	0.05	0.80	5	Stochastic Universal Sampling	Alternate Edges
50	100	0.05	1.00	5	Stochastic Universal Sampling	Alternate Edges

Table 9: Parameter vectors for crossover probability

Probability of Crossover	Min ObjV	Mean ObjV	Max ObjV	Number of Generations	Diversity
0.00	13,4749	13,4862	13,9987	22,8	0,0460
0.20	9,0326	9,0828	10,0035	66,2	0,0760
0.50	7,7246	7,8912	9,1842	82	0,1660
0.80	8,8467	12,9018	17,3944	99	0,8920
1.00	10,1462	15,3415	18,9190	99	0,9740

Table 10: Performance for different crossover probabilities

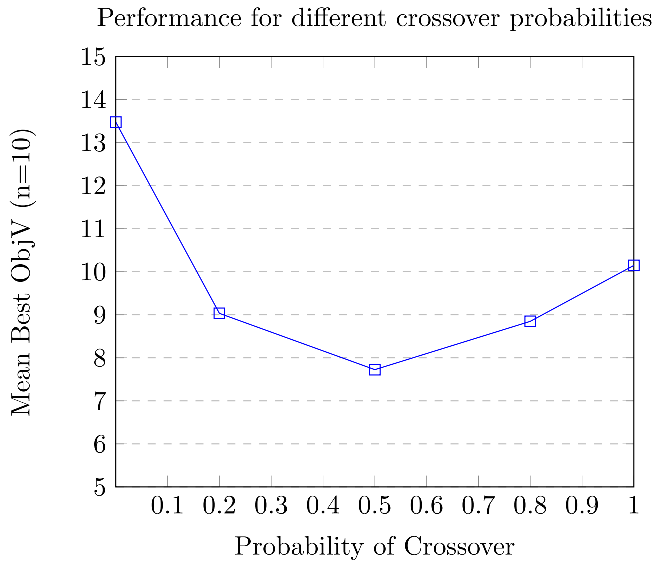


Figure 5

Numeric Parameters					Symbolic Parameters	
Individuals	Maximum Generations	Probability of Mutation	Probability of Crossover	Elitism (%)	Parent Selection Operator	Crossover Operator
50	100	0.00	0.95	5	Stochastic Universal Sampling	Alternate Edges
50	100	0.20	0.95	5	Stochastic Universal Sampling	Alternate Edges
50	100	0.50	0.95	5	Stochastic Universal Sampling	Alternate Edges
50	100	0.80	0.95	5	Stochastic Universal Sampling	Alternate Edges
50	100	1.00	0.95	5	Stochastic Universal Sampling	Alternate Edges

Table 11: Parameter vectors for mutation probability

Probability of Mutation	Min ObjV	Mean ObjV	Max ObjV	Number of Generations	Diversity
0.00	9,6532	14,9829	18,7421	99	0,9660
0.20	9,8030	15,0278	18,7321	99	0,9800
0.50	9,6412	15,3137	18,9749	99	0,9840
0.80	9,7483	15,4778	19,1161	99	0,9860
1.00	10,2592	15,7645	19,2363	99	1

Table 12: Performance for different mutation probabilities

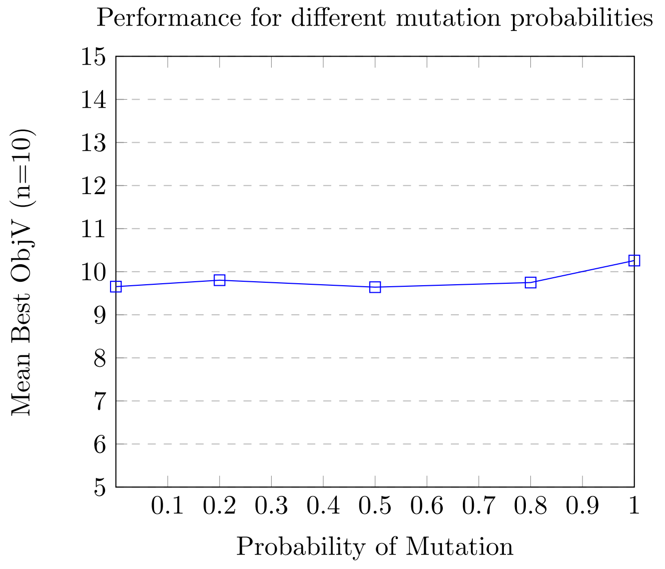


Figure 6

Numeric Parameters					Symbolic Parameters	
Individuals	Maximum Generations	Probability of Mutation	Probability of Crossover	Elitism (%)	Parent Selection Operator	Crossover Operator
50	100	0.05	0.95	0	Stochastic Universal Sampling	Alternate Edges
50	100	0.05	0.95	20	Stochastic Universal Sampling	Alternate Edges
50	100	0.05	0.95	50	Stochastic Universal Sampling	Alternate Edges
50	100	0.05	0.95	80	Stochastic Universal Sampling	Alternate Edges
50	100	0.05	0.95	100	Stochastic Universal Sampling	Alternate Edges

Table 13: Parameter vectors for percentage of elitism

Percentage of Elitism	Min ObjV	Mean ObjV	Max ObjV	Number of Generations	Diversity
0	12,2915	15,8096	18,9999	99	0,9980
20	8,8538	9,6099	11,7767	73	0,3320
50	8,8762	9,3481	11,4149	87	0,2500
80	9,4338	10,0568	12,3614	86	0,2720
100	11,0519	13,2248	15,3421	99	0,9460

Table 14: Performance for different percentages of elitism

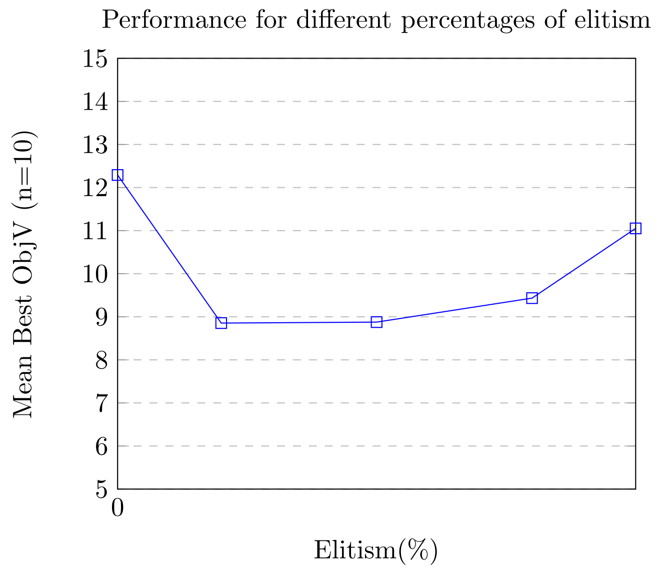
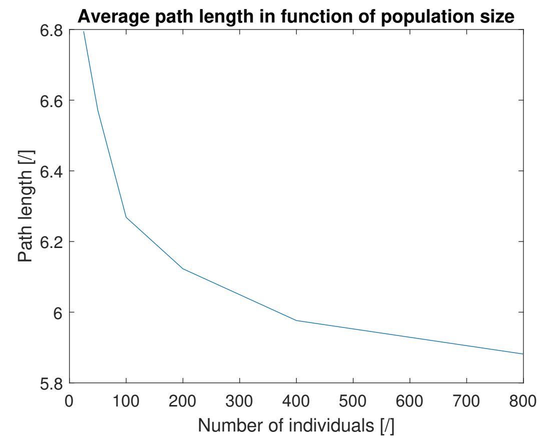
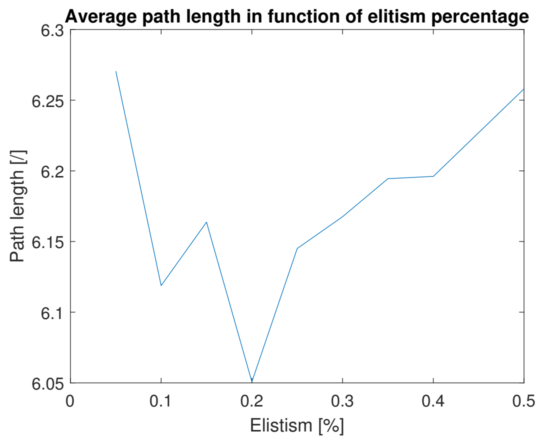


Figure 7

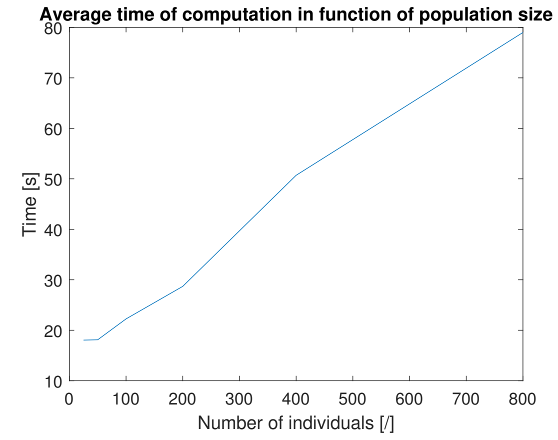
A.1 Graphics for parameter tuning



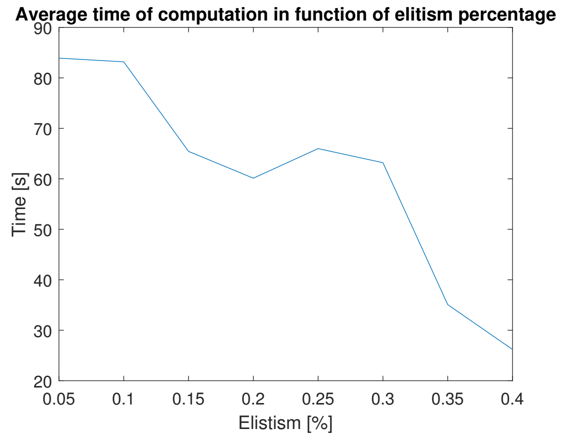
(a)



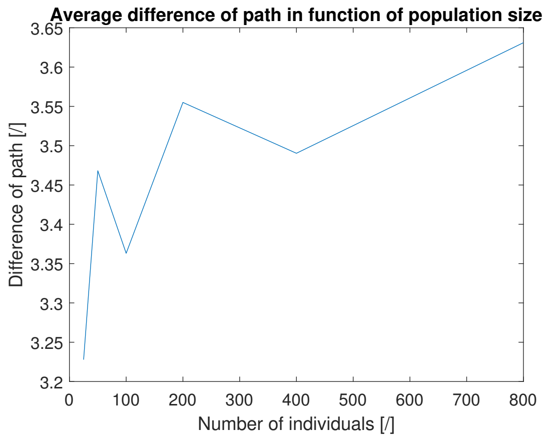
(b)



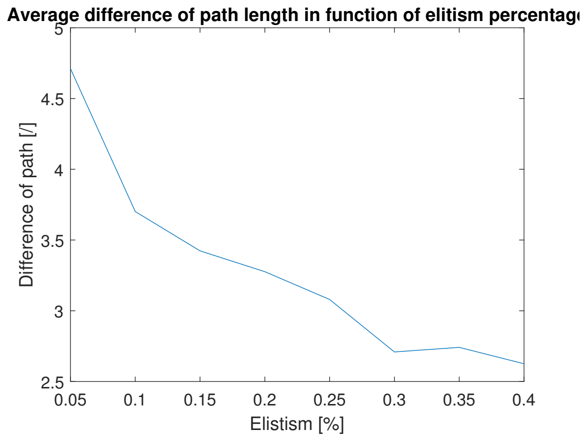
(c)



(d)



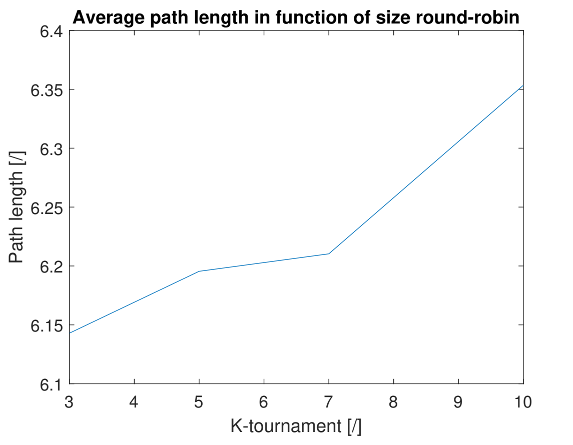
(e)



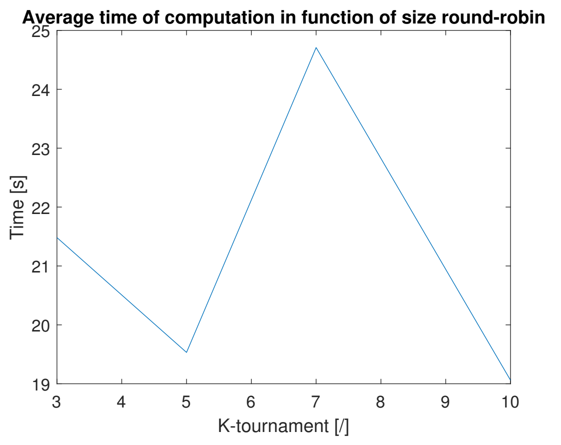
(f)

Figure 8: The graphics show the evolution of path length, computation time, difference of path between the best and the last individual of the 95% population (a measure of diversity) and number of generations before stopping, in function of elitism percentage and population size. We can clearly see that elitism and population size have opposing effects, tuning the selection pressure. The other parameters are left constant. (See reference set of parameters)

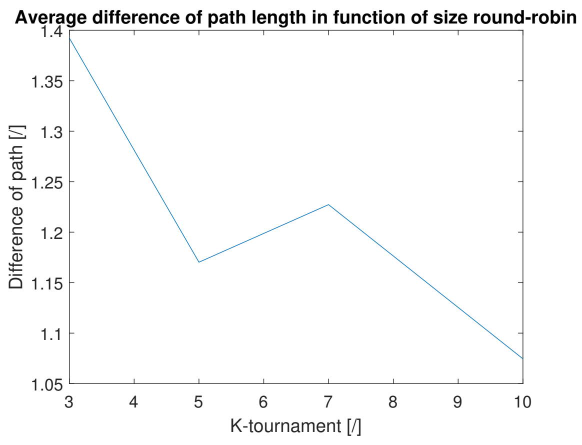
B Graphics of analysis of the round-robin tournament



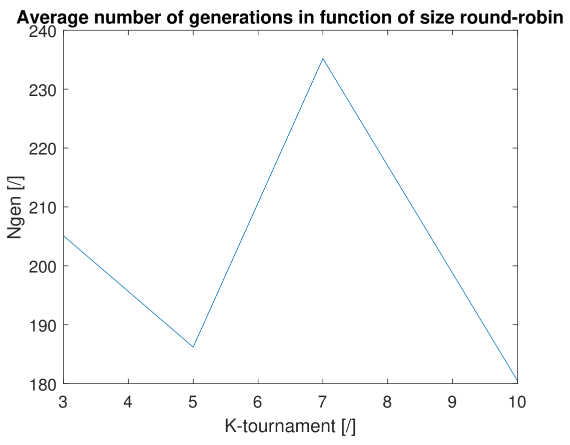
(a)



(b)



(c)



(d)

Figure 9: K is the size of the tournament. Here, it seems that a smaller tournament will fit better our use.