Genetic Algorithms and Evolutionary Computing Project

Danai Maria Triantafyllidou (r0735135)
Roshni .R. Kamath (r0727961)

Table of Contents

Task 2 - Perform a limited set of experiments by varying the parameters of the existing genetic algorithm and evaluate the performance

1. Introduction

The given implementation uses the adjacency representation, alternating edges crossover and swap mutation operators. The algorithm performs selection with stochastic universal sampling. We evaluate the performance of the give Matlab code by using the *average best-of-fitness generation (BOG)* which is often used in dynamic environments [1]. This measure shows the best fitness of each generation over all generations *G* and over the total amount of runs *n*. For evaluating the provided implementation, we consider ten simulation runs for each parameter setting and report the best-of-fitness generation evolution. In the base case experiments of task 2 the probability of mutation was set to 0.05, probability of crossover 0.95, population size 200, elitism 0.05 and number of generations 500. We test the given matlab code in a map of 100 cities.

1.1 Number of generations

The first experiment involved varying the Maximum number of generations performed by the genetic algorithm. The maximum number of generations is a termination criterion that can be used to control the maximum number of chromosomes that will be generated before the solution is returned. In the bottom left image of Figure 1, we report the average best fitness of each generation over 10 runs and for 10 different stall generation values. Increasing the maximum number of generations has a positive effect on the performance of the genetic algorithm, as it ensures that the algorithm has enough iterations to properly converge.

1.2 Elitism

Elitism refers to the process of preserving the fittest individuals, unchanged, into the next generation so that part of their genome is spread to the subsequent generations. This ensures that the evolutionary algorithm will not have to rediscover the fittest individuals and therefore improves the speed of convergence. The effect of elitism is analyzed by varying elitism from 0-50% for a fixed number of population size and generations and the results are shown in the upper left image of Figure 1. The use of elitism has a significant effect on the outcome of the simulations and the average best-of-fitness is produced by preserving 30% of the high fitness individuals. Our results indicate that when elitism is above 30% premature convergence arises, as high fitness chromosomes tend to dominate the population and the diversity within the population is lost.

In order to test the latter hypothesis, we calculate the fitness standard deviation in each iteration of the algorithm and we plot the mean standard deviation of each iteration. The standard deviation of the population is used as a measure of the population diversity. Over generations, offspring of elite individuals become more common than offspring of non-elite individuals, leading to a decrease in the population diversity. Having an elite speeds the convergence of the algorithm but also can increase the risk of converging to a local optimum.

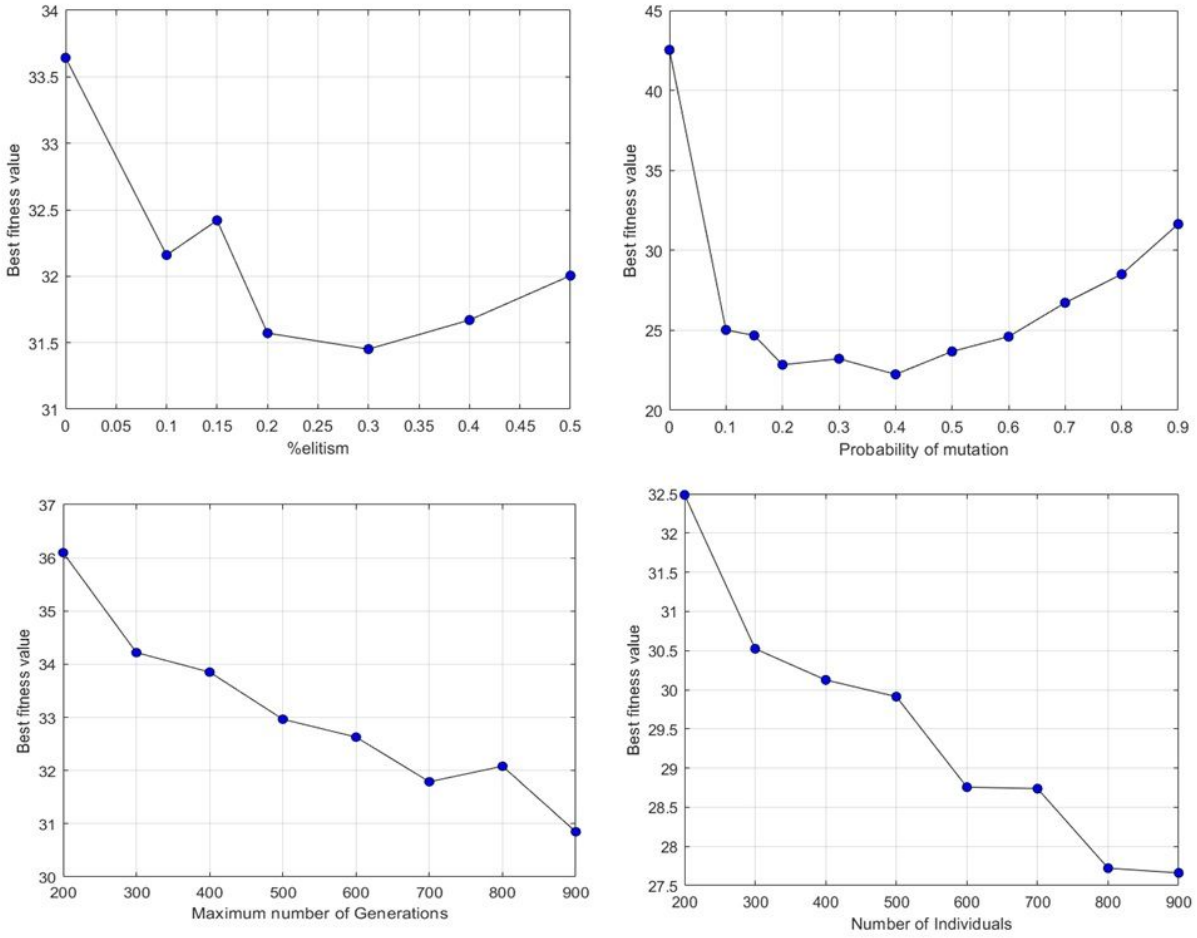


Figure 1: (Upper Left) Varying % of elitism, (Upper Right) Varying probability of mutation, Varying % of elitism, (Bottom Left) Varying Maximum number of generations, (Bottom Right) Varying number of individuals

1.3 Probability of Mutation

The mutation operator is performed after the crossover operation and ensures that the algorithm will not fall into a local optimum of the solved problem. In the upper right image of Figure 1, the results for varying the probability of mutation for a fixed number of generations- 500 and fixed population size-200 are shown. Increasing the probability of mutation from 0 to 0.4 seems to have a positive effect on the convergence of the algorithm, possibly due to additional genetic diversity. Accordingly, it is good practice to increase the mutation probability as the search progresses, in order to maintain an acceptable level of diversity in the population.

1.4 Probability of Crossover

Finding a solution to the traveling salesman problem requires we set up a genetic algorithm in a specialized way. For instance, a valid solution would need to represent a route where every location is included at least once and only once. If a route contains a single location more than once or missed a location out completely it wouldn't be valid and we would be valuable computation time calculating it's distance. If the algorithm converges too fast increasing the mutation rate will allow the algorithm to

explore a wider space. If the algorithm misses to reach local optima increasing the crossover rate helps to reach global optima.
Figure 2 shows the results for varying the probability of crossover for a fixed number of generations- 500 and fixed population size -200. As seen in the graph, increasing crossover rates creates more new offspring, at the risk of losing many good chromosomes in the current population. Conversely, low crossover rates tend to preserve the good chromosomes from one generation to the next, via a more conservative exploration of the search space. It is suggested that good performance requires the choice of a fairly high crossover rate such as 0.70 so that about 70% of the selected parents will undergo crossover.

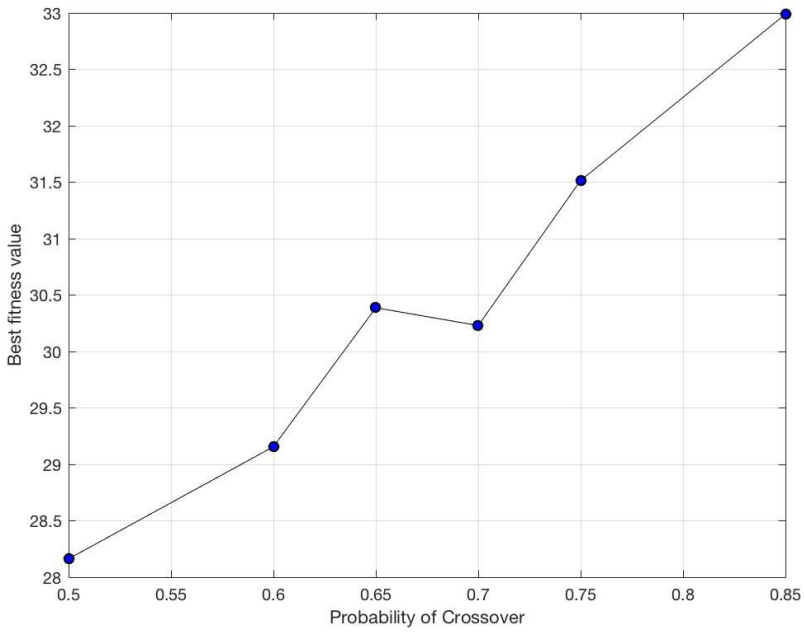


Figure 2: Varying probability of Crossover

1.5 Number of Individuals

A conclusion that may reasonably be drawn from this study is that the algorithm converged more rapidly with smaller populations, but with larger populations it eventually identified shorter tours. The algorithm performs best -- it discovers near-optimal tours most quickly -- with population sizes near those the estimate recommends with p = 900, but as population sizes move away from these values, no significant boost in performance occurs. As is shown in the bottom right image of Figure 1, small populations converge quickly to inferior results; they cannot maintain the variety that drives the genetic algorithm progress. As population sizes increase, the GA discovers shorter tours more slowly; it becomes more difficult for the GA to propagate good combinations of edges through the population and join them together. A limited set of experiments were also performed on problems of 25, 50 and 100 cities with population sizes 75, 100 and 300 sizes. As a genetic algorithm replaces current population elements with new offspring, it may delete some good edges from the population. This suggests that populations should be larger, so that multiple copies of good edges can provide insurance against their loss.

Task 3 - Implement a stopping criterion that avoids that rather useless iterations are computed.

The stopping criteria is used in the evaluation process to determine whether or not the current generation and the best solution found so far are close to the global optimum. Various stopping criteria can be used, and usually, more than one is employed to account for different possibilities during the running of the program: the optimal solution is found, the optimal solution is not found, a local optimum is found, etc. The standard stopping criterion that is used stops the procedure after a given number of iterations. Another stopping criterion is to stop after the "best" solution has not changed over a specified number of iterations. This will usually happen when we have found an optimum - either local or global - or a point near the optimum. Finally, another possible stopping criterion is when the average fitness of the generation is the same or close to the fitness of the 'best' solution.

The efficiency of a search algorithm can be evaluated as a ratio of the solution quality to the computational effort. Consider the following equations:

$$E_1(T) = \max_{t=0,\ldots,T} (F_t) \qquad (1)$$

$$E_2(T) = \sum F_t \qquad (2)$$

$$E_3(T) = (1/T) \max_{t=0,\ldots,T} (F_t) \qquad (3)$$

where T is a number of generations processed by algorithm, Ft is a maximal value of fitness observed in generation t. We measure the difference of E3(T) between two consecutive generations and we stop whenever this quantity is below a fixed threshold. This experiment is performed over 10 runs on a map with 25 cities with a fixed number of individuals=200, a maximum number of generations 6000, probability of crossover 0.95, mutation rate 0.2 and elitism 5%. In table 1, we present the the results of the implemented stopping criterion and we report the execution time in sec for each of value of the threshold. It is shown that when no extra stopping criterion is performed the algorithm needs 1363.26 seconds. About the same fitnes value can be reached with a threshold of 0.0001 by reducing the time to 15.65 seconds.

Average best-of-fitness of 10 runs	Execution Time in Sec	Threshold
4.0106	**1363.260252**	**No stopping criterion**
8.0138	0.231481	1
6.3524	0.593676	0.1
4.7266	1.649402	0.01
4.4942	5.278753	0.001

4.1001	15.658078	0.0001

Task 4 - Implement and use another representation and appropriate crossover and mutation operators. Perform some parameter tuning to identify proper combinations of the parameters.

4.1 Representation and operators

The most commonly used representations for the TSP problem are: 1) path representation, 2) ordinal representation and 3) adjacency representation. For this task, the default adjacency representation is changed to path representation. The adjacency representation does not support the classical crossover operator, as it may result in illegitimate tours [2].

We have decided to alter the existing algorithm in the following way:

1. Representation: path representation
2. Mutation: inversion type
3. Recombination: Partially Mapped Crossover (PMX), Edge Recombination Crossover(ERX).

There are four popular recombination operators for permutation representation:

a. Partially mapped crossover (PMX), which works fine for adjacency problems like TSP.
b. Edge crossover, which also well suited for adjacency problems. This operator is the default recombination operator in the given implementation of the algorithm.
c. Edge Recombination crossover: The edge recombination operator reduces the myopic behavior of the alternate edge approach with a special data structure called the "edge map".
d. Cycle crossover: it focuses on preserving information about the absolute position of the elements. As this is not desired or needed in TSP, we have decided to discard this recombination operator from further analysis.

Out of the aforementioned recombination operators, only two seem suitable for TSP: Alternate Edge Crossover, Edge Recombination crossover, and PMX. As edge crossover has already been implemented, we focused on PMX and Edge Recombination crossover(ERX).

4.2 Simulation and results

There are several widely accepted ways to measure performance:

1. Keep the maximum running time (approximated by computational effort) constant and define performance as the best fitness value at the end of the run. This is usually done by fixing the number of individuals in the population and the number of iterations. This metric is called mean best fitness (MBF).
2. Assume the minimal fitness level that has to be reached, and use computation effort expended as a performance metric. This metric is called the average number of evaluations to a solution.
3. Constrain both running time and minimum fitness level and use the ratio of successful runs to unsuccessful runs as a performance metric. A successful run is defined here as a run in which minimum fitness level has been achieved within the time limit. This metric is called success rate.

As minimal fitness level is not trivial to determine for different ‘city maps’ and there is no objective way to do it, the second and the third option would be rather challenging to define in a way that is not biased.

Therefore, we have decided to use the first performance metric. The performance of the developed implementation is measured in the following way:

1. We perform parameter tuning for the crossover and the mutation operators on a map with 100 cities.
2. The length or each run is kept constant and equal to 10.
3. We use a random seed to initialize a pseudorandom number generator for each experiment. This way the initialization of the algorithm is the same and meaningful comparisons can be made.
4. Performance is measured by taking the average best fitness of all the runs for a fixed parameter setting.
5. Instead of performing a grid search over all the parameters of the algorithm, we vary one parameter and we fix the rest.

4.2.1 Evaluating the performance of the Insertion mutation operator

We test the performance of the genetic algorithm by examining the effect of varying population sizes and mutation rates for the insertion mutation operator. Our results show that a higher mutation rate leads to deteriorated performance for the TSP problem. We analyze the performance of different operators by varying their parameters for different values for the number of generations and population size. First, we set the population size to 400 and we measure the fitness by varying the maximum number of generations. The algorithm did perform better with mutation than without when populations were small. For each of these cases, we compute results for probability of mutation 10%, 15% 20% and 25%. The same experiment is performed for a population size of 600 and 1000.
The results of this experiment are summarized in Figure 3. In each figure, the y-axis is the value of the best tour length or the *average best-of-fitness generation* and the x-axis is the maximal number of generations, which serves as the only termination criterion of the genetic algorithm during this experiment. As previously stated, our results indicate that changing the population size results in slight or insignificant difference in the value of the best tour length. Additionally, increasing the mutation rate does not contribute to better performance, it severely increases the execution time of the algorithm without leading to better solutions.

4.2.3. Evaluating the Crossover Operators

Firstly, we have set the value of the maximal number of generations to 900, in case of edges recombination crossover(ERX), the maximal generation is 500 because of computational limitations and we calculated the fitness for the cases where the size of the initial population is: 300, 500 and 700 and for each of these cases, we calculated the results for the crossover probability of: 0.5, 0.6, 0.65, 0.75 and 0.85 (as in Figures: 4 and 5).
As it can be observed from Figure 4, for ERX the algorithm performs better when the probability of crossover is 0.85.

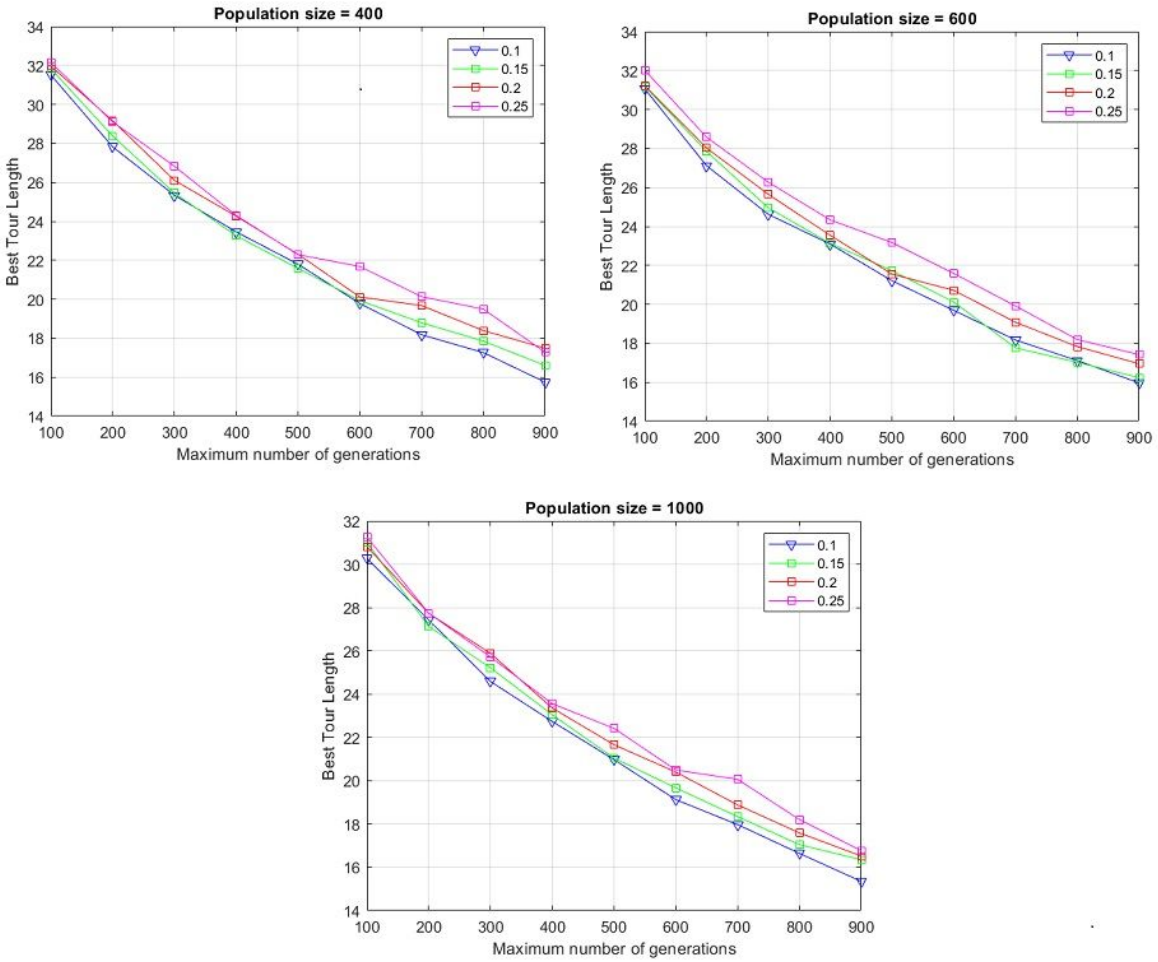


Figure 3: Varying mutation rates for different population sizes

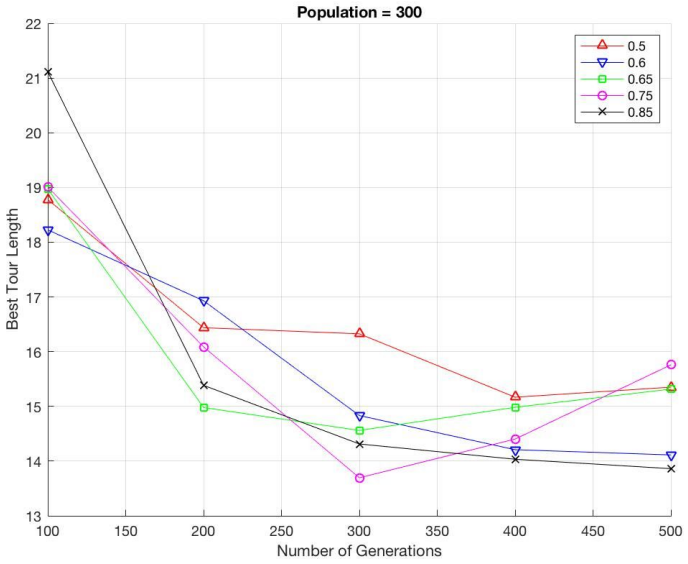


Figure 4: Performance of Edges Recombination Crossover

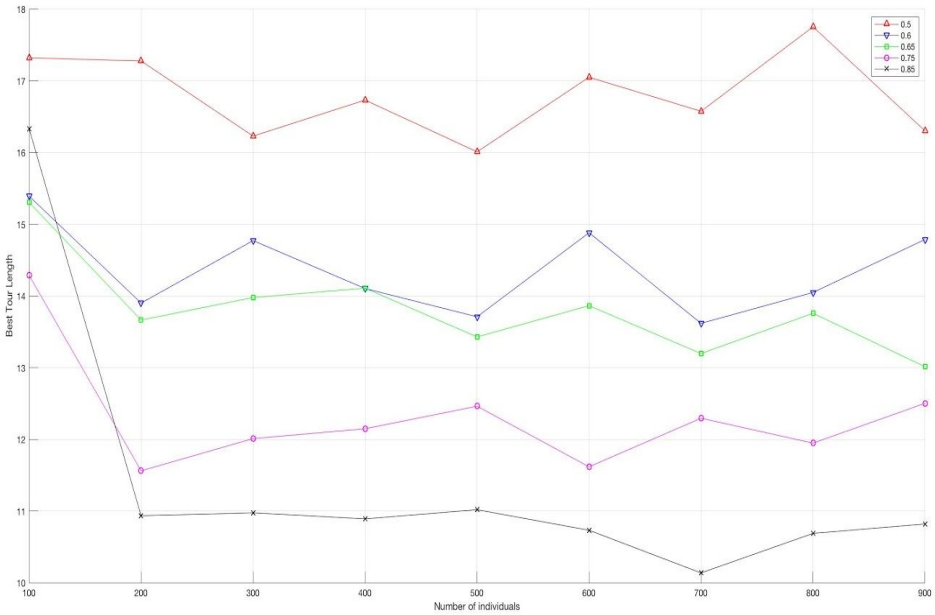


Figure 5: Performance of Partially Mapped Crossover

4.3 Parameter Tuning

Having decided how to measure performance, we had to design our parameter tuning process. Firstly, we have determined the limit values for all the parameters. Limit values determine the range of values for given parameter for which we still obtain reasonable solutions. It is worth mentioning that the parameters are not independent from each other, and changing one of them might change the reasonable range for the others. Because of that, we kept the widest range obtained as a reasonable range. While this part of the tuning process was not strictly rigorous, due to limited computational power, we have taken possible coupling of parameters into account. While the parameter tuning is obviously separate for the default algorithm and our modified algorithm, it turned out, that reasonable limits are actually the same for both algorithms.

If elitism exceeds 20% the quality of solutions drops considerably. Additionally, setting value of elitism to 0 drastically worsens the solutions. Hence the elitism value is kept at 10%. We fixed the number of individuals/chromosomes to 300. The probability crossover to 85% and the probability of mutation to 10% (using bigger step size would probably lead to missing the optimal values, as we could ‘jump over’ the optimal parameters).

Based on tests we have come to the following conclusions:

1. Increasing probability of crossover to 1 leads to worse solutions on average.
2. Varying crossover probability between 0.95-0.85 and mutation probability between 0-0.15 does not affect the performance of the algorithm in a significant way. Between Edges Recombination(ERX) and Partially mapped crossover(PMX) operators, we observed that ERX converges to better value and at the same time exploring the search space but at the cost of computation time.

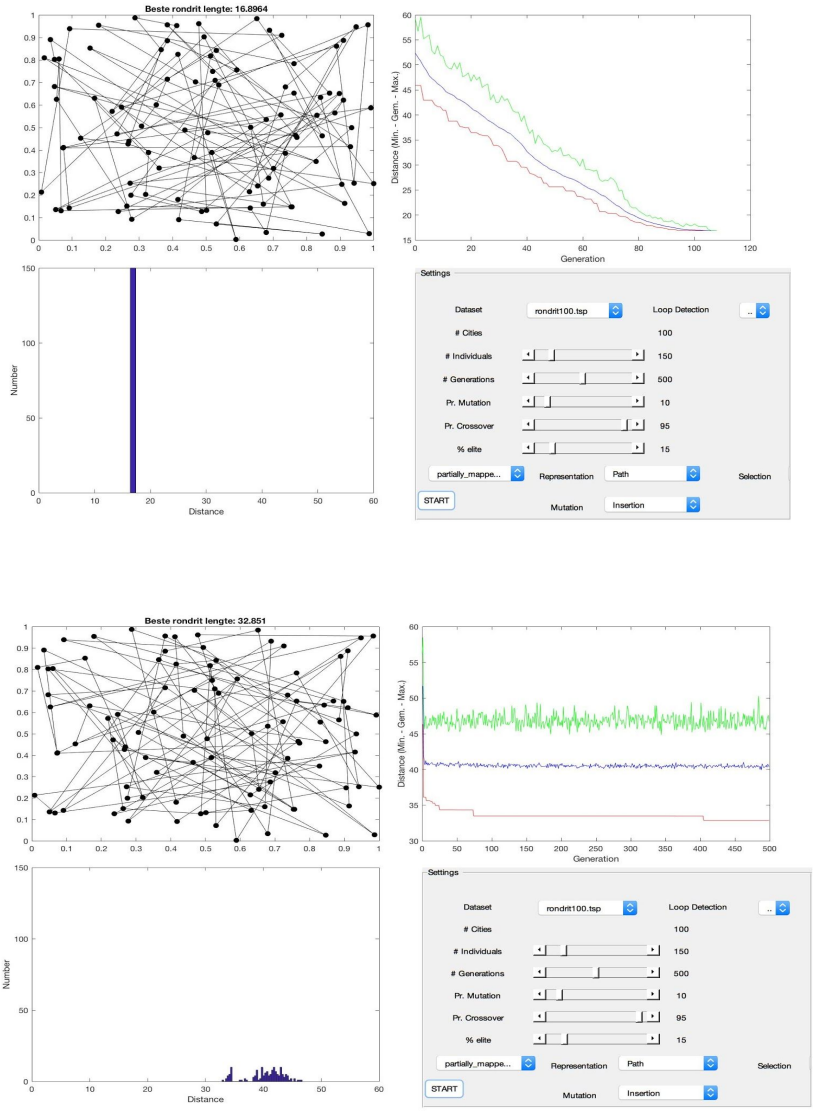
3. Generally, algorithm performs better with elitism at 0.10 than at higher values.
4. Increasing elitism to 0.10 along with mutation rate to 0.15 leads to better performance increase. As elitism reduces the diversity and focuses more on exploitation than exploration, while mutation rate focuses on exploration mostly and increases diversity, these two balance out and increase the overall performance. More specifically the improved performance might be explained by the fact that increased mutation rate allows to find more promising solutions, while high elitism rate ensures frequent recombination with good solutions increasing the probability of small local improvements. This combination seems to have advantages of both high elitism and high mutation without their disadvantages, as it improves both exploration and exploitation.
5. If we vary the crossover rate we find that the graphs in Figures 4 and 5, in contrast to the mutation rate, favour a high crossover probability. However, setting it too high will lead to too much 'thrashing' since good tours are not preserved enough. It turned out, that as long as this rate is kept within 0.85-0.95 there is no significant difference in performance. The optimal crossover rate seems to be between 0.75- 0.85.

Population,	**Generation ,**	**Cities ,**	**Elitism ,**	**Xover,**	**Mutation**	**Best tour length**
300	800	100	0.10	0.95	0.10	13.976
400	800	100	0.20	0.85	0.20	12.24
500	800	100	0.20	0.65	0.30	15.21
500	800	100	0.15	0.75	0.10	12.3456
500	800	100	0.25	0.95	0.10	13.1188

Table 2: Parameter Tuning Results

Task 5- Test to which extent a local optimization heuristic can improve the result.

Using loop detections sharply improves the performance at the initial stage which and converges to the optimum value faster. In general, it seems to improve overall performance by saving time in the initial stage by quickly finding more optimal solutions but the performance does not improve as the generation increases. The effect is less noticeable in the later stages of the algorithm run. Loop detection can hence save the on the number of generation as it quickly converges to best tour length.



Task 6 - Test the performance of your algorithm using some benchmark problems and critically evaluate the achieved performance.

We tested our algorithm, including the parent selection methods on five complex benchmark problems which were provided, for which the perfect solutions are known. Modified algorithm with Edges recombination crossover(ERX) operator and Tournament selection outperforms the already given algorithm. We also used Partially Mapped Crossover for Path representation which also outperformed the given algorithm though the performance was not as good as ERX as seen in the previous experiments too.

We noted the best fitness length for all the 5 benchmark problems using the same parameters that were used previously. The algorithm was performed over 100 individuals and 300 generations for 5 iterations. Additional results are presented in the Apendix.

	BCL380	Belgium Tour	RBX711	XQF131	XQL662
Default Algorithm	124.1944	8.921	296.35	27.906	199.4916
Tournament + ERX	89.2754	5.614	166.492	19.398	144.7019
RWS + PMX	100.9786	7.413	210.8447	24.7049	174.4203
No. of cities	380	41	711	131	662

Table: Average best tour length

Task 7 - Implement and use two other parent selection methods

Exploring the ways to improve our algorithm further, we have implemented alternative ways of parent selection, namely tournament selection and Roulette wheel selection (fitness proportionate selection). Firstly we concisely describe implemented selection algorithms along with ranking selection, then we discuss the results of using these algorithms on the performance and the quality of solutions of our GA applied to TSP.

7.1 Roulette wheel selection

The parent chromosomes are selected for mating via proportional selection, also known as "roulette wheel selection". It is defined as follows:

1. Sum up the fitness values of all chromosomes in the population.
2. Generate a random number between 1 and the sum of the fitness values.
3. Select the first chromosome whose fitness value added to the sum of the fitness values of the previous chromosomes is greater than or equal to the random number.

7.2 Ranking selection

Proportional selection(RWS in this case) has some drawbacks. In particular, a "super-chromosome" with a very high fitness value will be selected at almost every trial and will quickly dominate the population. When this situation occurs, the population does not evolve further, because all its members are similar (a phenomenon referred to as "premature convergence"). To alleviate this problem, the rank of the fitness values can be used as an alternative to the usual scheme

The method is implemented as follows: individuals are ranked based on their fitness. The probability to be selected is assigned based on the rank, which might be done in a number of ways. Assigning the probability proportionally to the ranking is widely used. However, due to the linear relationship between the ranking and the probability, only limited selection pressure can be applied. If one wants to apply stronger selection pressure, an exponential ranking is used.

7.3 Tournament selection

Tournament selection works as follows: a number k (in case of a binary tournament, 2) of individuals is randomly chosen from the population and the fittest of these individuals is selected. The procedure is repeated λ times to create the new generation. While this selection mechanism is oftentimes used in

parent selection it may be used in survivor selection as well. There are two main benefits of this method: the algorithm itself is rather simple and the selection pressure can be easily adjusted by changing the size of the tournament (k).

7.4 Ranking methods and Comparison chart

To assess the performance of our algorithm with modified parent selection we had to perform parameter tuning once more. We varied each of the parameters and if improved performance was observed we changed it even more. We kept on changing all parameters that way until there was no change possible that would perform better. In other words, we performed simple hill climbing on each parameter separately. Here we again assumed that there is only one optimum in parameter space, as long as parameters stay in reasonable range. To make the results believable in any way, we run the algorithms 10 times for each parameter setting and we used MBF again as a performance metric.

RWS and tournament selection is better than proportional selection in terms of maintaining steady pressure toward convergence. Tournament and proportional roulette wheel is able to achieve optimal solution for small size instances; however the quality of solution reduces as the size of instance increase. Tournament selection is more efficient than Proportionate roulette wheel selection in terms of convergence rate. Results are summarized in Figure 6.

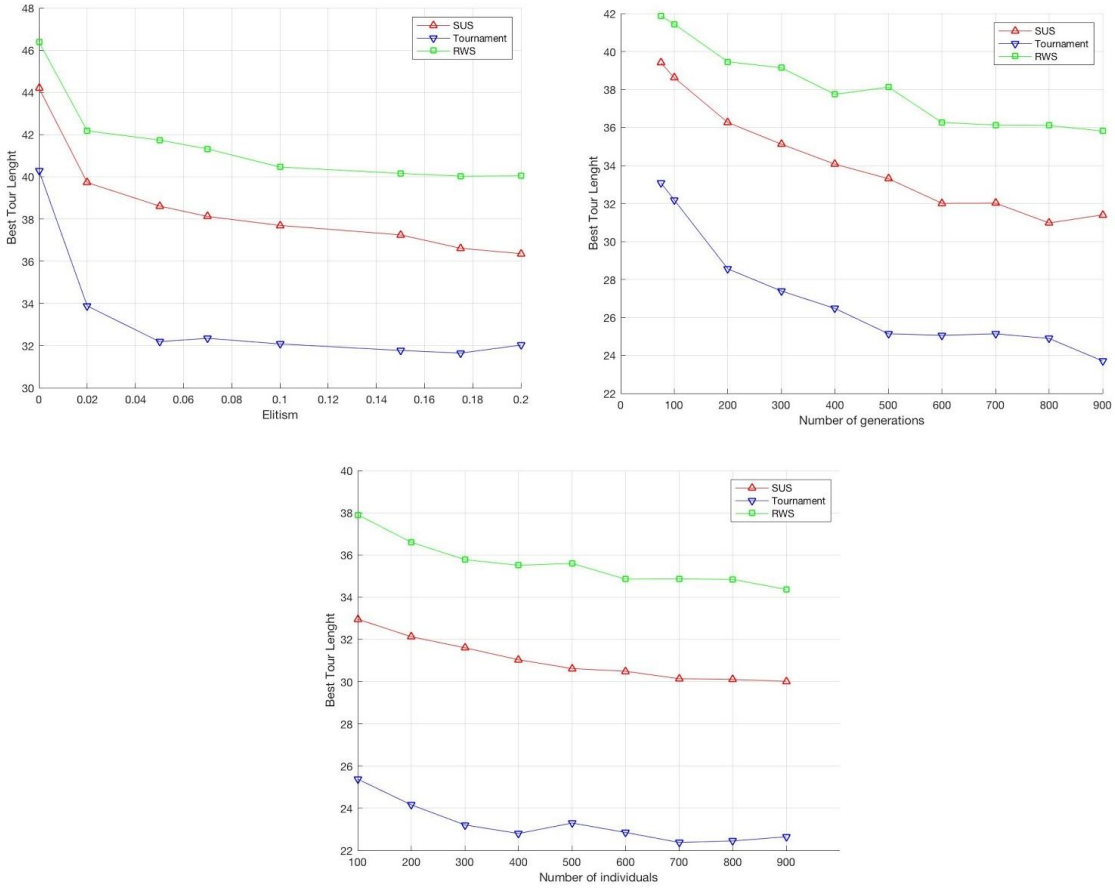


Figure 6: (Upper Left) Sensitivity of Tour length vs Elitism, (Upper Right) Sensitivity of Tour length vs Generation, (Bottom) Sensitivity of Tour length vs Individuals

Conclusion

We have modified the template algorithm using path representation, ERX and PMX as recombination operator and insertion mutation as mutation operator. This better performance might probably be attributed to a number of changes, one of which may be different recombination operator, which does not seem to be better by itself, however, it seems to make the algorithm more robust against premature convergence, allowing higher elitism percentage, which allows to exploit the solution space more effectively. To balance it out, increasing mutation probability allows to explore the search space more thoroughly, finding better local optima and preserving density in the population, which counterweights elitism in a way. Additionally, more optimal code allows to decrease clock time, while still providing superior performance. We found that in general having more operators is a good thing, because one operator can move you throughout the space in a way that another can't. It helps get the algorithm off of local optima.

References

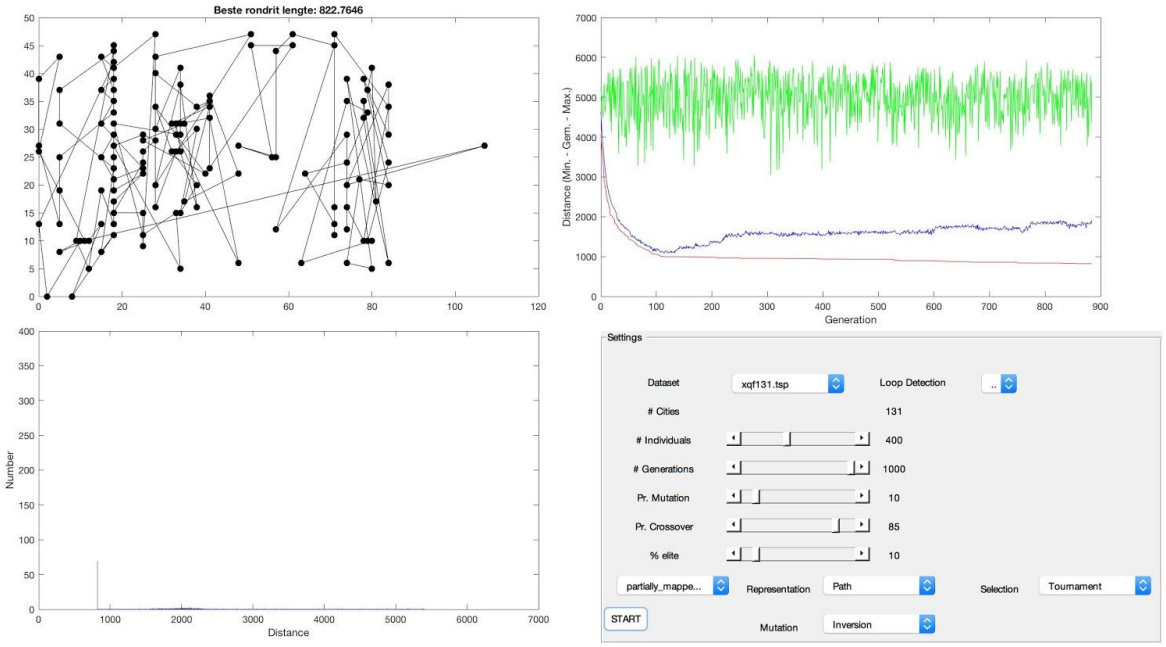
[1] Morrison, R.W.: Performance measurement in dynamic environments. In: A.M. Barry (ed.) Proceedings of the Bird of a Feather Workshops, Genetic and Evolutionary Computation Conference, pp. 99–102. AAAI, Chigaco (2003)

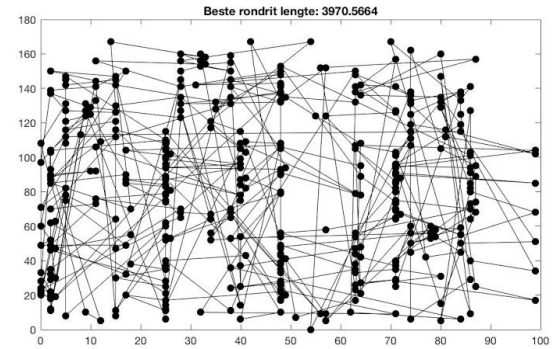
[2] Larrañaga, Pedro, et al. "Genetic algorithms for the travelling salesman problem: A review of representations and operators." *Artificial Intelligence Review* 13.2 (1999): 129-170.

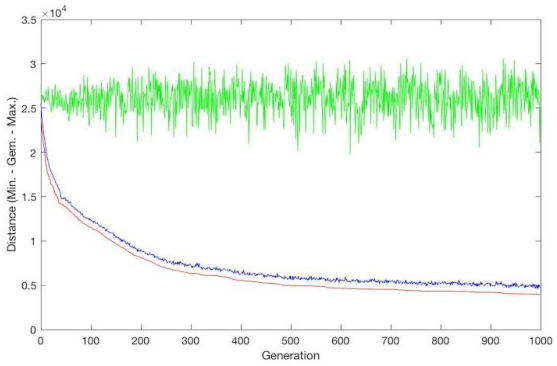
[3] Otman Abdoun, Jaafar Abouchabaka, Chakir Tajani, Analyzing the Performance of Mutation Operators to Solve the Travelling Salesman Problem. arXiv preprint arXiv:1203.3099

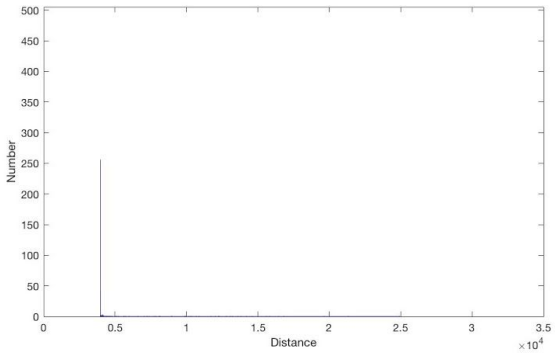
Appendix

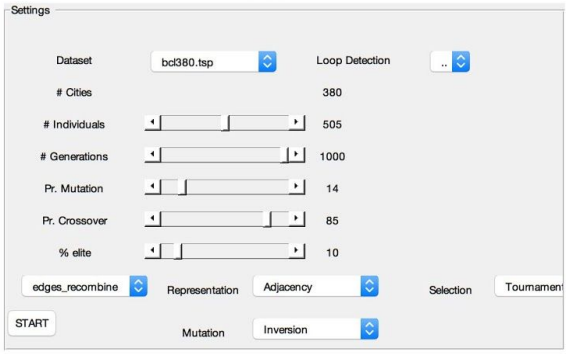
The appendix contains the results of some of benchmark problems(xqf131, bcl380,rbx711) tested against the modified algorithm.

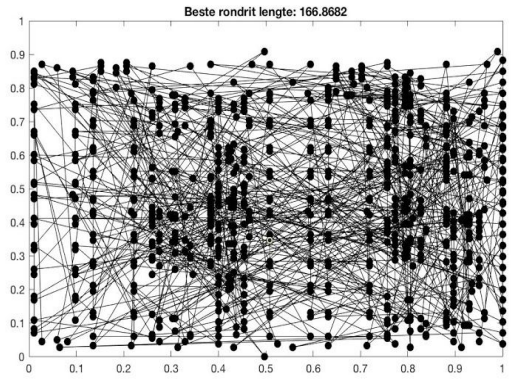


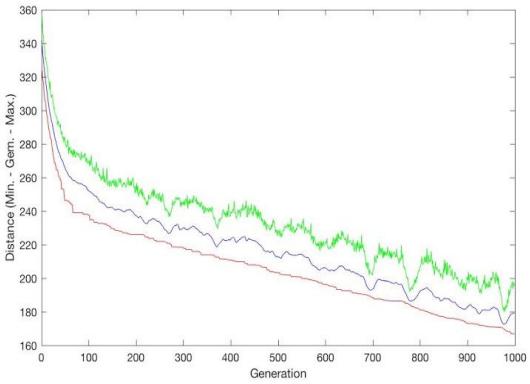

Beste rondrit lengte: 3970.5664

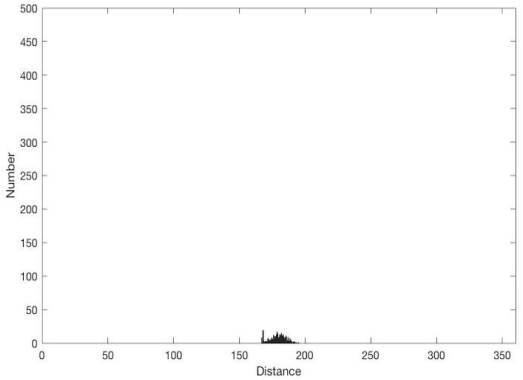

Distance (Min. - Gem. - Max.)
Generation


Number
Distance


Settings
Dataset
bcl380.tsp
Loop Detection
Cities
380
Individuals
505
Generations
1000
Pr. Mutation
14
Pr. Crossover
85
% elite
10
edges_recombine
Representation
Adjacency
Selection
START
Mutation
Inversion


Beste rondrit lengte: 166.8682


Distance (Min. - Gem. - Max.)
Generation


Number
Distance


Settings
Dataset
rbx711.tsp
Loop Detection
Cities
711
Individuals
500
Generations
1000
Pr. Mutation
10
Pr. Crossover
85
% elite
5
edges_recombine
Representation
Adjacency
Selection
Sus
START
Mutation
Insertion