Genetic Algorithms

Nicolas Delahousse, Robbe Neyns

November 2019

1 Existing Genetic Algorithm

1.1 Dataset

We performed our tests on the *rondrit127.tsp* data-set. We chose this specific data-set for its size and complexity. We expect that the increased number of cities will better reflect the changes in parameter values on the obtained best tour length.

1.2 Parameters

The goal of the experimentation phase is to understand the effect of the different parameters on the algorithm's performance. The five parameters that can be tweaked are: the number of individuals, the number of generations, the probability of mutation, the probability of crossover, the percentage of Elite and whether loop detection is activated or not.

A specific series of steps were taken to better understand the influence of each parameter. First, a base case is determined of default values for each parameter. We chose to use the default values provided by the toolbox for each parameter. The values for the base case are summarised in the table below.

Table 1: Toolbox Default Parameter Settings

Cities	Individuals	Generations	Pr. Mutation	Pr. Crossover	Elite	Loop detection
25	50	100	0.05	0.95	0.05	false

Then for each parameter we ran a series of experiments where the parameter's values are changed incrementally while holding the remaining variable values constant. Since the results of genetic algorithms are non-deterministic, we ran each experiment five times and averaged the resulting best tour lengths. Doing this ensures a better understanding of the results because it compensates for the inherent variability of a genetic algorithm caused by the initialization of the population.

1.3 Test results

The results of the first experiments are summarised on Figure 1. All variables were tested except loop detection. During all the trials loop detection was turned off. The Figure was cropped to make trends more visible. The consequence is that the individuals look cut-off from the horizontal axis. The reason is that the smallest bin for the individuals' values is zero, for which no best tour length can be calculated.

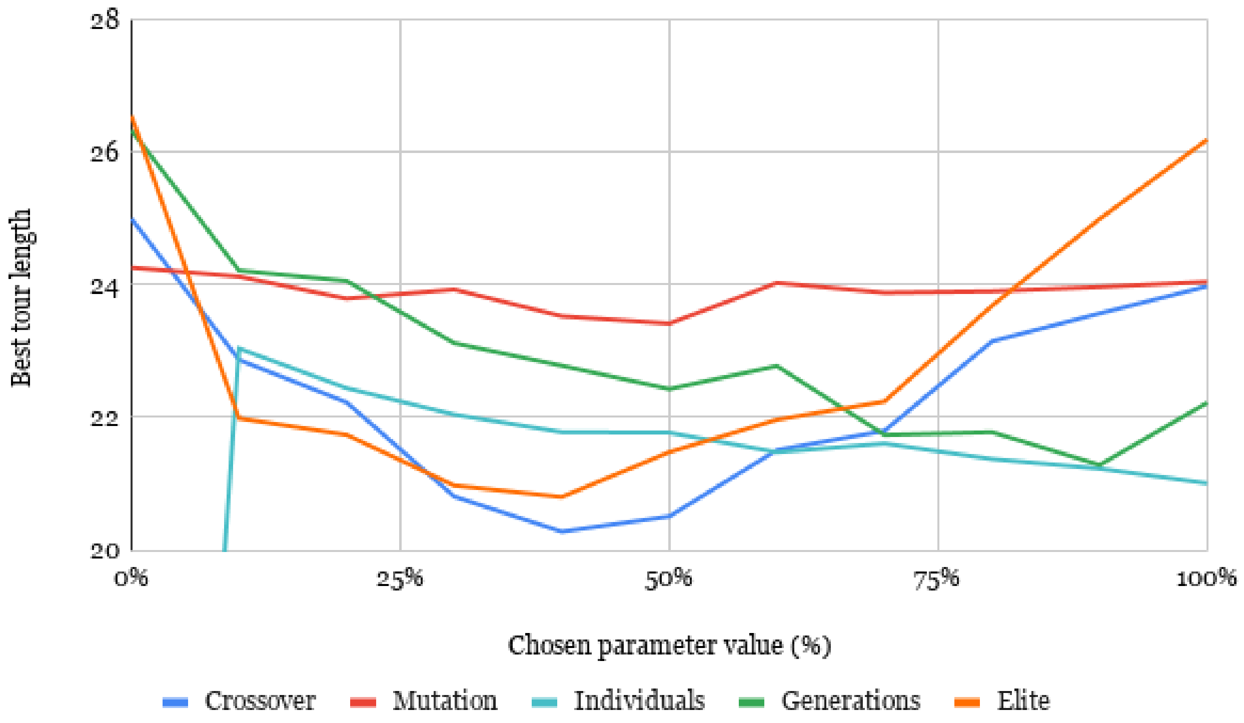


Figure 1: Toolbox Parameter Experiments

We observed that increasing the amount of individuals linearly reduces the best tour length. The mutation operator achieves the best tour length around the 50% mark, but the effects are negligible. The crossover operator displays a quadratic trend with a minimum achieved at the 40% mark. The Elite parameter also displays a quadratic shape with a minimum achieved at around 30%. As the number of generation increase gradual improvement in best tour lengths are observed.

The loop detection was always turned off the previous experiments. In the next experiment we performed five runs with and without the loop detection on and averaged the results. The results are summarised in the next table.

Table 2: Parameter values

Loop Detection	Average Best Tour Length
On	24.25
Off	14.00

The loop detection has a very significant impact on the average best tour length.

1.4 Discussion

Increasing the amount of generations for the genetic algorithm leads to clear improvements in best tour length. However, the most substantial improvements are made in the early generations. Therefore, defining the correct level of generations is more of a trade-off in available time and computation power. On average a number of 100 generations seems to be a good decision for this data-set. It allows the genetic algorithm to stabilize and generate the best results without being too time constraining or computationally intensive. Nevertheless, the optimal number of generations will be higher for a larger data-set.

We conclude that the percentage of crossover and the total number of individuals have the largest influence on the outcome.

2 Stopping criterion

As a stopping criterion we decided to use the variance of the best result in the N last iterations.

$$s^2 = \frac{\sum_{n=1}^{N}(x_i - \bar{x})^2}{(n-1)}$$

With s^2 being the variance, x_i the best result at iteration i and $\bar{x}$ the average best score over the last N generations. When the variance drops below a certain stopping threshold (θ_{stop}), the algorithm terminates prematurely. This stopping criterion was chosen mainly because of its adaptability. It has two variables, namely N and θ_{stop}, which can be adapted according to the value of the other parameters. For example, it is preferred to give N and θ_{stop} small values when we are dealing with a small data set. It is also interesting to adjust the parameters between model runs to find their optimal values for a given data-set.

2.1 Test results

The stopping criterion was tested on data-sets of different lengths but to improve the readability of this document, only the results of the stopping criterion on a data-set of medium size will be presented and discussed. The parameters of the base case have not been chosen with performance in mind. They are purely selected to show the effect of the stopping criterion. In Table 4 we can see an overview of the results from the experiments. It can be seen that the best result was obtained by taking an N of 0.4 * # generations. The algorithm continues until a good result is reached but stops when the largest improvements have been realised. A smaller N resulted in an earlier termination of the algorithm. For an N of 0.1 * # generations this was almost immediately after 70 generations, when a first stagnation in improvement occurs.

The variations in θ_{stop} had a smaller influence on the best path and the number of generations, taking a smaller value for this parameter does result in a later termination of the algorithm but not with the same margin that was observed when varying the value of N.

Table 3: Parameter values

Parameter	Value
# individuals	500
# Generations	100
Pr. Mutation	5
Pr. Crossover	95
% elite	5
Loop detection	off

In Table 4 is shown that we chose to take values that lie close to the initial model values for the parameters in the base case. The stopping criterion was applied to this base case with different values for N and θ_{stop}. The results of these experiments can be seen in Table 3.

2.2 Discussion

The best result was obtained with a rather large N (0.4 * # generations) and a θ_{stop} below 0.1. However, the value of θ_{stop} seemed to have less influence than the value of N. The reason for

Table 4: The table represents the influence of the stopping criterion. More specifically, it describes the difference in 'best path' with different values for N and θ_{stop}.

N	θ_{stop}	Shortest path	number of generations
/	/	5.1307	500
0.1 * # generations	0.05	7.9316	70
0.2 * # generations	0.05	7.4507	190
0.4 * # generations	0.05	5.9319	440
0.2 * # generations	0.1	7.5623	170
0.2 * # generations	0.01	6.4074	240
0.2 * # generations	0.005	7.04	312

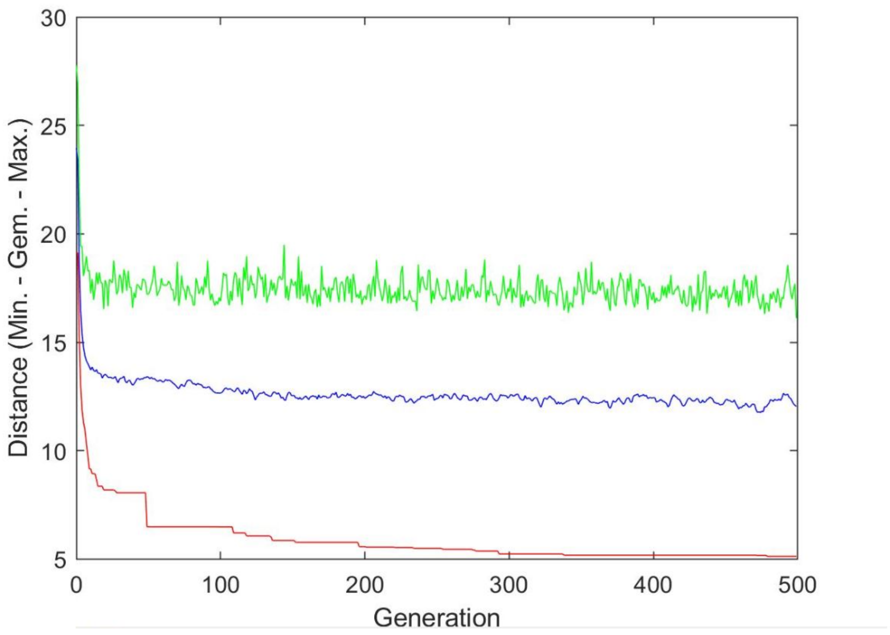


Figure 2: The image shows the evolution of the fittest value (indicated in red) for the base case (500 generations in total).

this becomes clear when we look at figure 2. As the algorithm advances it doesn't continuously improve over the course of n generations. It often follows a staircase pattern with prolonged periods of stagnation followed by a sudden improvement. This implicates that a short N will most likely lead to a premature activation of the stopping criterion. This is also why using the variance will yield better results than using absolute deviation. When using variance, the deviation of scores from the mean are squared, this gives more weight to extreme scores. Since we want our stopping algorithm to 'pay attention' to the sudden improvements, it is beneficial that extreme shifts get a higher weight.

Nevertheless, it might be that other operators (crossover or mutation) do not cause this staircase method in improvement and that a smaller N might be more appropriate. In other words, it probably depends on the other characteristics of the tsp solver what the optimal values for N and θ_{stop} are.

3 Other representation and appropriate operators

There are five representations that can be used to represent the traveling sales man problem Namely:

- The binary representation
- The path representation
- The ordinal representation
- The adjacency representation (used as default)
- The matrix representation

3.1 Path Representation

The most natural representation of the TSP is the path representation, it has also been proven to provide good results [5]. Furthermore, a large variety of crossover operators has been developed for this representation since it is the most used representation to solve the TSP problem. As such we have chosen to implement this representation into our algorithm. The concept is very intuitive: the path is represented as a list of N cities. The order of the cities in the list represents the order in which they will be visited.

3.1.1 Crossover operators

Different crossover operators were considered and tested: the modified cyclic operator (CX2) [4], the partially mapped operator (PMX) [3] and the ordered operator (OX) [2]. The PMX is the most basic of the three and didn't lead to very good results so it will not be discussed any further. Unfortunately the CX2 algorithm showed inconsistencies that were inherent to the algorithm. As a result, only the Ordered crossover operator will be discussed in more detail because it lead to the best results.

Order Crossover (OX The order crossover operator exploits the property of the path representation that the order of the cities is important and not their absolute position. So it benefits convergence if a part of the order of each parent is maintained in the child after crossover. As such the algorithm constructs 2 children by taking a sub-tour of each parent in which the order is preserved.

3.1.2 Mutation operators

Larrañaga [5] identifies 6 mutation operators for the path representation:

1. Displacement Mutation (DM)
2. Exchange Mutation (EM)
3. Insertion Mutation (ISM)
4. Simple Inversion Mutation (SIM)
5. Inversion Mutation (IVM)
6. Scramble Mutation (SM)

Out of these six mutation operators, DM IVM and ISM have been shown to perform the best [5]. The literature shows no clear "winner" among these three. In the end we decided to implement the ISM. The ISM takes a city in a given path and places it randomly at another location in the path.

3.1.3 Parameter tuning

The optimal parameter values were already analysed for the adjacency representation. However, using these same parameter values to perform tests with another representation might lead to sub-optimal results. As such we performed a second analysis to find the optimal parameter values for the path-representation.

Our main goal was to determine whether certain representations are naturally more sensitive to either recombination or mutation operators. Subsequently we wanted to find out what the optimal values were for both the percentage of mutation and recombination and what the results imply about the representation.

We formulated two hypotheses respective to these two goals.

(1) In the path representation recombination operators find better routes than mutation operators.

(2) Higher rates of recombination combined with lower rates of mutation are more optimal

We designed our experiments the same way as we did the benchmark test in section 1, namely in "sets". In the first set we studied the effect of changes in crossover rate. In the second set of experiments we studied the effects of changes in the mutation rate. In each experiment we changed the values of the variables ceteris paribus. The default values used are the ones provided by the toolbox as represented in the table below.

Table 5: Toolbox Default Parameter Settings

Cities	Individuals	Generations	Pr. Mutation	Pr. Crossover	Elite	Loop detection
127	50	100	0.05	0.95	0.05	false

For both the mutation operator and the crossover operator the experimental approach was the same. We increase the rate of the operator by a factor of 10% and ran a simulation five times, and took the average of the five best tour lengths.

We performed multiple experiments with different parameter values. For each experiment we noted the length of the best tour, the maximal tour length and the average tour length. Each experiment is carried out three times and the values of the three length are averaged to rule out the possibility of encountering a particularly bad or good run.

The first experiment serves as a base case. Here we take the default parameter values as provided by the toolbox.

3.1.4 Data set(s)

The test were carried out on the rondrit127.tsp dataset. It was chosen because it contains the most amount of cities. The quality of a recombination or mutation algorithm will better be illustrated under complex circumstances.

3.1.5 Test results

The results of the experiments are summarized below in Figure 3. This figure displays the results best tour length for combinations of crossover and mutation rates.

The figure displays the individual and combined effect of both operators well. First we describe of each variable individually, and then look at their combined effects.

The crossover operator has a distinct quadratic shape. Across the board the crossover operator achieves its shortest path for values between 40% and 60%, independent of mutation.

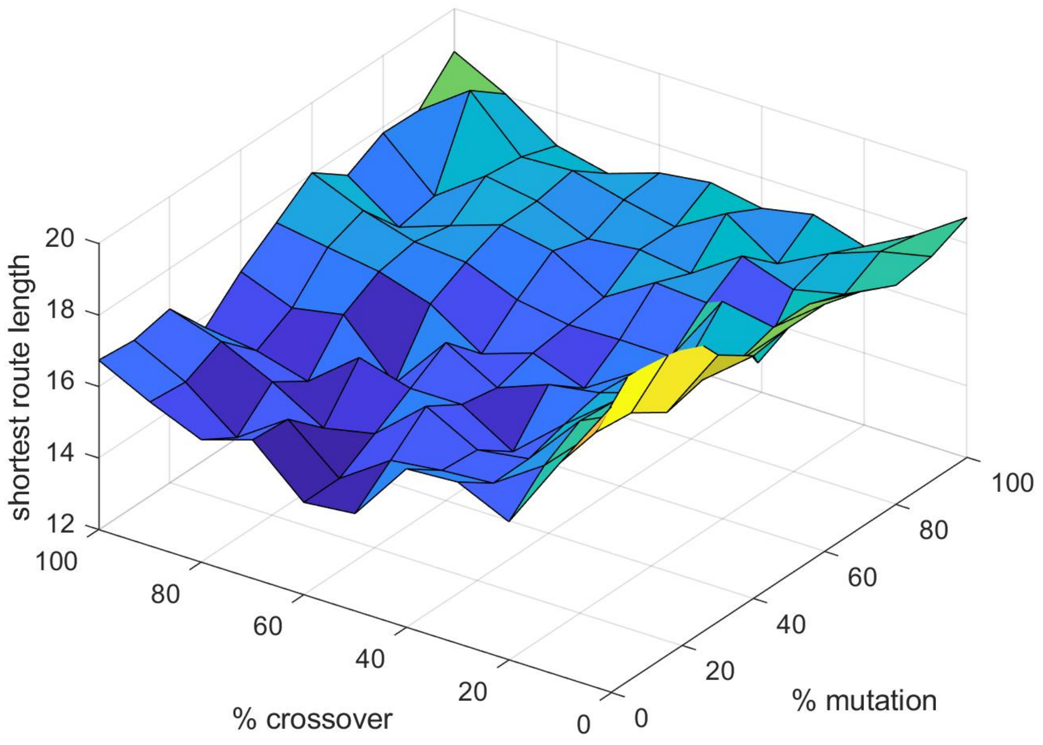


Figure 3: Effect of joint changes in crossover and mutation rate on shortest route length

The mutation operator exhibits a more subtle trend. The shortest tour length gradually increases as the mutation rate increases. The best values for the mutation rate are achieved between 20% and 30% independent of crossover rates.

It is clear that the rate of the crossover operator has a significant impact on the effect of the mutation operator. Indeed, the figure displays three prominent peaks and one minimum. The individual effects of the mutation and the recombination operator are reinforced. The overall shortest route length is achieved for crossover rates ranging between 40%-60% and mutation rate ranging from 0%-20%.

The peaks are achieved when the mutation rate is at its highest values and the crossover operator rate is either maximal or minimal.

The combined effect of the variables is consistent with the behavior of the individual variables. However, we notice that the crossover operator has a significant impact on the effect of the mutation operator. When we considered the impact of the mutation operator separately, it achieved its overall best results around a rate of 20%. However the results of the entire figure question the usefulness of the mutation operator. Indeed, the crossover operator always achieves the best results around the 60% mark with clear improvements as the mutation rate decreases.

3.1.6 Discussion of test results

To evaluate our results we used three criteria. Performance, convergence and efficiency. Performance is represented by the length of the best tour. The length of the best tour can be compared across experiments when the amount of generations is fixed between experiments. Efficiency describes the time the algorithm took to generate all the requested generations. This is important since some algorithms require more steps per generation. Convergence is defined by the time (or number of generations) the algorithm needs to perform the largest improvements.

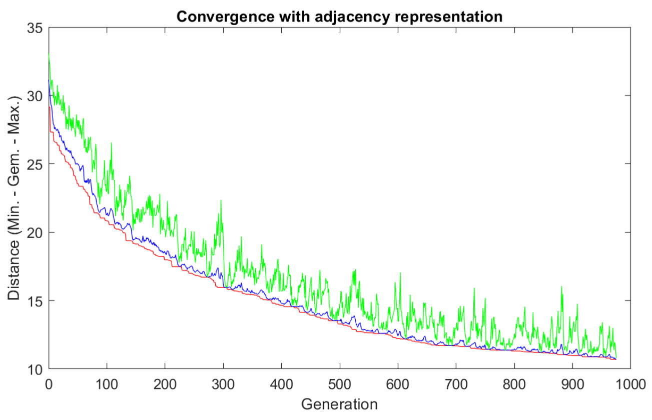
This is most easily observed by looking at the evolution of the best tour length throughout the total amount of generations.

In terms of performance we achieved the best tour length of 14 with a crossover rate of 60% and a mutation rate of 0%. We conclude that the implemented mutation operator ISM had no effect on the best tour length. This seems to contradict the claims made by Larranaga [5] about the effectiveness of the ISM mutation operator. However, Larranaga's paper does not tell what rates they used for the mutation operator. They performed an experiment for each combination of mutation and recombination operator. However they did not perform an experiment where only the mutation or the recombination operator would be used.

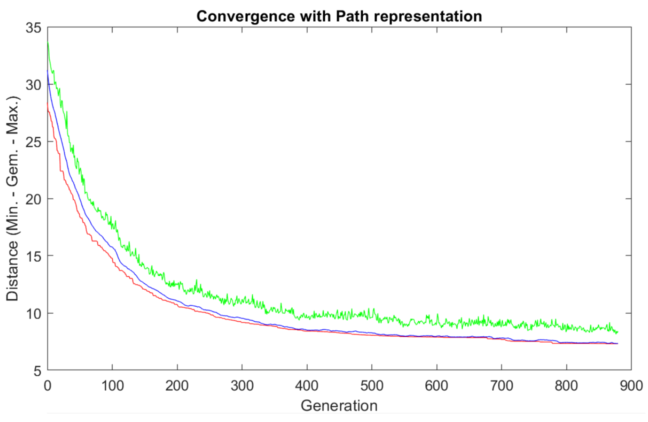
with the path representation we consistently achieved a shorter paths than with the adjacency representation.

In terms of efficiency we notice on Figure 7 that the run time of the path representation using the OX operator is consistently higher than the run-time achieved by the adjacency representation. Nevertheless this is only a small difference.

To test the convergences of the adjacency and path representation we ran separate experiments. We increased the generations to 1000 for both representations and set the parameter values to their optimal tuning values as defined by previous experiments. The goal was to observe evolution of the best tour length.



(a) Adjacency representation.



(b) Path representation.

Figure 4: Convergence in both representations.

The path representation has a much steeper slope than the adjacency representation. This indicates that the path representation converges faster than the adjacency representation.

While the adjacency representation computes slightly faster on average than the path representation, it takes more generations before it reaches optimal values.

4 Local optimisation

A heuristic can be used to improve the results and convergence speed of the algorithm. Different heuristics exist to improve the performance of a genetic algorithm when it is used to solve the tsp. A heuristic operator was already included in the base algorithm, namely the no-loop operator. It is commonly referred to as 2-opt local search method [1]. A local optimisation operator aims mainly at improving the population after crossover and mutation have been applied. However we reasoned that it is better to include a heuristic in the operators itself, so we chose to implement a heuristic crossover and mutation operator based on the distance between cities[7]. By combining both, an extra calculation (for the optimisation) is omitted and this has a positive effect on the efficiency of the algorithm.

4.1 Heuristic crossover operator (HX)

The crossover operator is fairly simple, it follows the following steps:

- STEP1: A city is chosen at random within the domain. We will refer to this value as c. The offspring is assigned this value at position 1.
- STEP2: 2 pointers are created for each parent. 1 pointer to the left of c and 1 to the right of c (figure 5).
- STEP3: Each pointer points at a different city. The city with shortest distance from c (we will name it h) is added to the offspring at position 2. Afterwards the pointer shifts from h to the city next to it and $c_{new} = h$.
- STEP4: the cities that have been added to the offspring are removed (equaled to zero) in both parents.
- STEP5: step3 and 4 are repeated until each city occurs once in the offspring.
- STEP6: The same process is repeated to create a second offspring with the same parents.

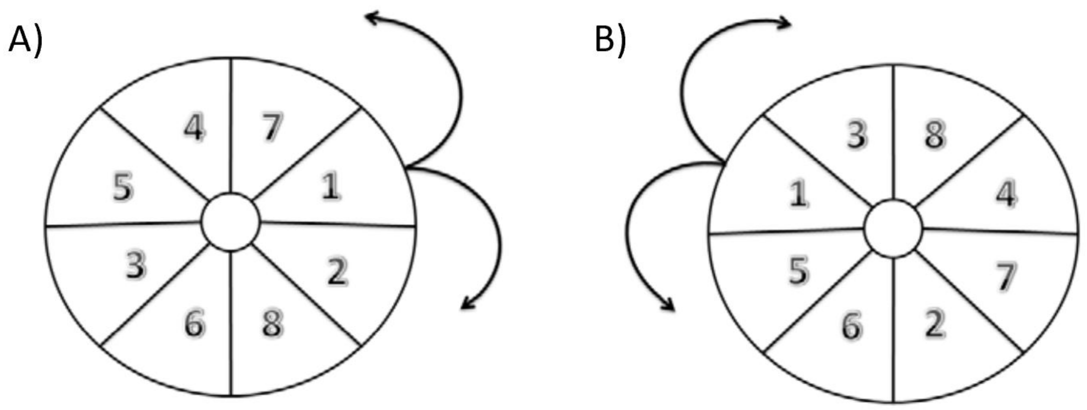


Figure 5: A) parent 1; B) parent2. Both images illustrate step 2 in the description of the heuristic crossover operator. First a random city is selected, in this example this is city 1. Then the 2 pointers are created, one to the left and one to the right of city 1.

4.2 Heuristic mutation operator

The heuristic mutation operator works as follows, for each selected individual:

- STEP1: a random city is chosen (city a).
- STEP2: the city which lies closest to city a is selected as well (city b).
- STEP3: the order of all the cities in between city a and city b is reversed.

4.3 Test results

Both operators were tested on the data-set with 100 cities, an overview of the model parameters for the test run can be seen in table 6.

Table 6: The parameter values used to test the performance of the heuristic crossover and mutation operators.

Parameter	Value
# individuals	500
# Generations	100
Pr. Mutation	5
Pr. Crossover	95
% elite	5
Loop detection	off

Three runs were performed with these parameters. The average of the best value for these three runs is 7.7814. The average of the worst value of these runs is 8.391. Both values lie very close to each other, indicating that most of the population evolved towards the optimal solution and that there is only a small variation in the population in the last generation. Interestingly, the optimal solution is better than the one obtained in the other experiments, indicating that this algorithm performs better. On this small data-set, the parameter settings shown in table 6 gave very good results. However, some additional test showed that it is important to make crossover the dominant process because the mutation operator has a tendency to get stuck in a local optimum. This became very clear in a test run with only mutation and no crossover. During this test, the whole population became stuck in a local optimum of 26 after 40 generations.

The algorithm also converges faster to an optimal solution. The optimal value was already reached after 15 generations. The consequent iterations were unnecessary and the algorithm would have been stopped prematurely if the stopping-criterion were activated.

Computationally, the algorithm didn't appear to be more expensive than the previous algorithms (figure 7). Moreover, since it reaches an optimal solution much faster, it can be argued that the algorithm is also less costly.

4.4 Discussion

The performance of the heuristic operators is better than the ordered crossover. Both in terms of performance and in terms of convergence, the optimal solution is found in less time then for the other configurations. However, it is important to choose the appropriate parameters. More specifically, a high mutation rate leads to premature convergence due to a sudden reduction of the search space. This causes the algorithm to get stuck in local optima. Nevertheless, if optimal parameter values are chosen, the algorithm reaches an optimal solution much faster than in the previous experiments.

5 Benchmark problems

5.1 Test results (incl. performance criteria and parameter settings)

Table **??** shows the result of our algorithm compared to the optimal solution. The parameter settings to obtain these results are shown in table **??**. We decided to apply the best version of our algorithm to these benchmark problems with the features that showed the best results on previous tests, which consist of:

- Variance based stopping criterion
- Heuristic crossover operator
- Heuristic mutation operator
- Rank based roulette wheel detection

The maximum number of individuals was put at 500, in the initial trials this seemed to provide enough variety in the population to obtain very satisfying results. The maximum number of generations was put at a very high number: 800. However the stopping algorithm was activated before the 800th generation for each data-set.

The percentage of crossover was put at 95, compared to a mutation percentage of 5. The choice was based on previous observations, this appeared to be a good configuration to avoid local optima.

Table 7: The parameter values used to test the performance of the final algorithm on the benchmark problems.

Parameter	Value
# individuals	500
# Generations	800
Pr. Mutation	5
Pr. Crossover	95
% elite	5
Loop detection	on
N_{stop}	0.1*#generations
θ_{stop}.	0.05

Table 8: Results of the experiments for the benchmark datasets

Run	Average tour length	Best tour length	Benchmark tour length	Number of generations
xqf131	571	566.42	564	225
bcl380	1677	1673.53	1621	290
xql662	2600	2588.37	2513	540
rbx711	3202	3201.25	3115	600
belgiumtour	659.82	659.82	?	90

5.2 Discussion

The version of our algorithm used for these tests gives very good results. The shortest path lies close to the optimal solution for all the benchmark problems. Furthermore, the algorithm requires a reasonable amount of time to reach its optimal solution. However, in these tests the exponential nature of the required time to reach an optimal solution relative to the problem size becomes very clear. The rbx711 data set took 10 times as long as the bcl380 even though the number of cities is only twice as large.

6 Other Tasks

6.1 1. Implement and use two other parent selection methods

6.1.1 Description of implementation

Two additional selection procedures were implemented and compared against each-other: tournament selection and rank based roulette wheel selection.

Tournament selection A classic tournament selection procedure with tournament size 2 was implemented. Upon each iteration, 2 random individuals are selected from the population. The individual with the highest fitness "wins" the tournament and is selected for reproduction/mutation. A tournament size of 2 (smallest possible) leads to a small selection pressure. Because the larger the number of individuals in a tournament, the higher the chance that a high-fitness individual is included in that tournament and thus the lower the chance that a low-fitness individual will win. Tournament selection is probably one of the most popular selection methods due to it's efficiency and simple implementation [3]. Tournament selection gives a chance to all individuals to be selected and thus prevents premature convergence. No fitness scaling or sorting of the individuals is required, which benefits the efficiency of the algorithm.

rank-based roulette wheel selection Rank-based selection is a procedure where the probability of an individual to be selected is based on its fitness rank relative to the whole population. The method relies on a mapping function which assigns a selection probability to each individual based on their rank. This function greatly influences the performance of the selection procedure. The following linear mapping function was used in our selection procedure:

$$Rank(Pos) = 2 - SP + (2 * (SP - 1) * \frac{(Pos - 1)}{(n - 1)}$$

The bias of the selection procedure can be controlled through the selective pressure (SP) which preferably takes a value between $2.0 >= SP >= 1$. Following the formula, the scaled rank of the best individual will be at best SP and for the worst individual 2-SP at worst.

The rank based selection scheme has been shown to provide better results than the traditional proportional roulette wheel selection. This is mainly due to it's effectiveness in avoiding local optima [6] and the adaptability of the selection pressure in the mapping function, which allows the user to easily test out different selection pressures. However, due to the necessity of ranking the individuals, the approach is computationally more expensive than its counterpart.

6.1.2 Description of the experiments

To compare the influence of the different selection methods, both were used on the "rondrit100.tsp" data-set with the parameters shown in table 9. The run-time on all the "rondrit" data-sets was also measured for both selection procedures (figure 7).

6.1.3 Test results

The best tour length after 100 iterations was 7.76 for the roulette wheel selection with a total run-time of 32.5 seconds. The best tour length for the tournament selection was 7.75 with a total run-time of 31.8 seconds. There is only a small difference between the two but this is also largely due to the fast convergence of the heuristic crossover and mutation operator.

Nevertheless, there is a clear difference in the diversity of the population after 100 generations. In figure 6 it can be seen that the diversity in the population is much smaller when using the tournament selection.

Table 9: The parameter values used to test the performance of the final algorithm on the benchmark problems.

Parameter	Value
# individuals	500
# Generations	100
Pr. Mutation	5
Pr. Crossover	95
% elite	5
Loop detection	off
Stopping Algorithm	on
Crossover operator	Heuristic crossover
Mutation operator	Heuristic mutation

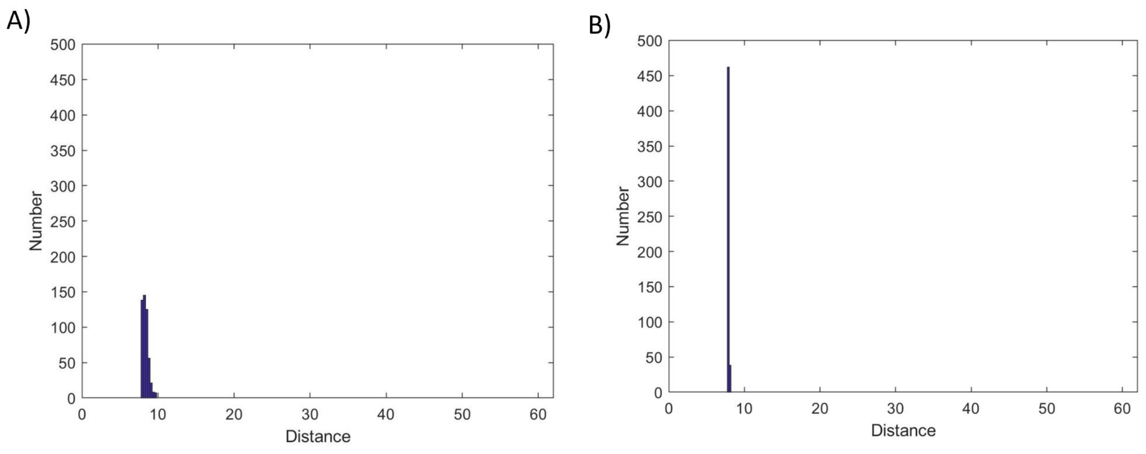


Figure 6: The figure shows the spread of the population after 100 generations. A) With the rank-based roulette wheel selection. B) With the tournament selection.

In terms of speed, figure 7 shows that the roulette wheel selection seems to give better results than the other selection methods. However the difference is small. The time consumption of both selection procedures also appears to increase at the same rate with problem size.

6.1.4 Discussion of test results

No significant difference in performance could be seen between the tournament selection and the rank based roulette wheel selection. Both provide good results in combination with heuristic crossover and mutation operator. However, the population maintained more diversity with the rank based roulette wheel selection. This didn't influence the performance of the algorithm on this (rather small) data-set but with a larger data-set, it decreases the chance of getting stuck in a local optimum. Additionally, the rank-based roulette wheel selection seemed to be faster for the considered data-sets. This is surprising because sorting of the population array is necessary and this was expected to demand more time. However, one can argue that the data-sets are not large enough and that this effect would only become visible when applying the algorithm to data-sets with more cities. Another possible explanation is that the MATLAB sort function is very efficient.

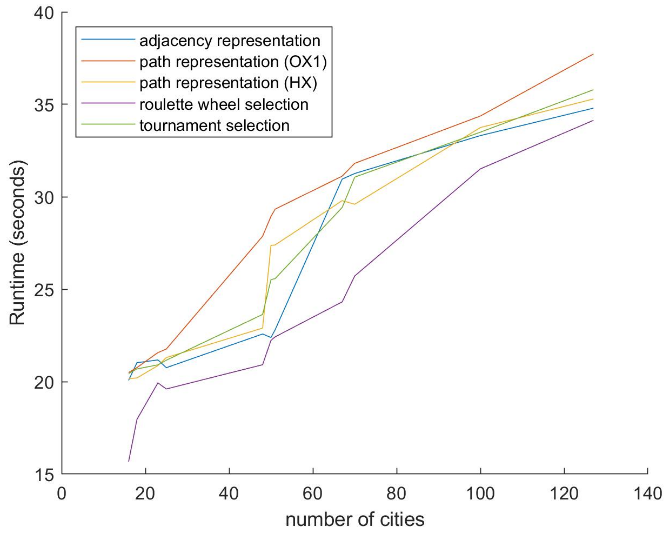


Figure 7: Runtime of different experiments plotted against the size of the dataset.

7 Time spent on project

7.1 Time Distribution

(a) Robbe Neyns: 50u (b) Nicolas Delahousse 45u

7.2 Task distribution

2. Briefly discuss how the work was distributed among the team members.
   Robbe Neyns:

   - Ordered crossover operator
   - heuristic crossover operator
   - heuristic mutation operator
   - selection operators
   - helping in writing the report.

   Nicolas Delahousse:

   - stopping criterion
   - part 1: Existing Genetic Algorithm (writing)
   - Part 3: other representation and appropriateoperators (writing)
   - researching mutation and crossover operators
   - implementing mutation operator

References

[1] Georges A Croes. “A method for solving traveling-salesman problems”. In: *Operations research* 6.6 (1958), pp. 791–812.

[2] Lawrence Davis. “Applying adaptive algorithms to epistatic domains.” In: *IJCAI*. Vol. 85. 1985, pp. 162–164.

[3] David E Goldberg, Robert Lingle, et al. “Alleles, loci, and the traveling salesman problem”. In: *Proceedings of an international conference on genetic algorithms and their applications*. Vol. 154. Lawrence Erlbaum, Hillsdale, NJ. 1985, pp. 154–159.

[4] Abid Hussain et al. “Genetic algorithm for traveling salesman problem with modified cycle crossover operator”. In: *Computational intelligence and neuroscience* 2017 (2017).

[5] Pedro Larranaga et al. “Genetic algorithms for the travelling salesman problem: A review of representations and operators”. In: *Artificial Intelligence Review* 13.2 (1999), pp. 129–170.

[6] Noraini Mohd Razali, John Geraghty, et al. “Genetic algorithm performance with different selection strategies in solving TSP”. In: *Proceedings of the world congress on engineering*. Vol. 2. 1. International Association of Engineers Hong Kong. 2011, pp. 1–6.

[7] Gohar Vahdati, Mehdi Yaghoubi, Mahdieh Poostchi, et al. “A new approach to solve traveling salesman problem using genetic algorithm based on heuristic crossover and mutation operator”. In: *2009 International Conference of Soft Computing and Pattern Recognition*. IEEE. 2009, pp. 112–116.