

Genetic Algorithms and Evolutionary Computing: Project

2011-2012

Li Quan  
Wout Swinnen

# Contents

# Introduction

In this project, some of the key concepts of solving combinatorial optimization problems with the use of genetic algorithms are explored. To be more precise, we will implement a genetic algorithm and investigate its properties and performance when applied to the travelling salesman problem or TSP.

The TSP is the classic example of a route planning problem. The aim of the TSP is to find a route between cities that is as short as possible, where every city has to be visited once, and where the point of arrival has to be the same city as the starting city. There are many more complicated forms of this TSP, but these will not be discussed in this project.

The genetic algorithm or GA is an elegant solution to this problem, among many other problems. It is a concept that has grown over the last half century, which mimics the mechanisms of natural evolution and genetics as known from biology. Just like our genes, genetic algorithms tend to make sure the strongest solutions to a problem survive, and the weaker ones do not. To do so, it makes use of genetic operators such as crossover and mutation, and often also a selection process comes at hand.

The implementation of the genetic algorithm will not start from scratch, but is based on an existing GA toolbox that contains a graphical user interface and other functionalities relevant to these genetic processes. We will of course only comment on code that has been changed and has been created by ourselves.

The main target of this project is to investigate the performance of these implementations. In particular, we will implement the crossover operators ERX, EERX, PMX, OX, CX and the hybrid crossover for the path representation, together with the mutation operators of exchange mutation, insertion mutation, simple inversion mutation, inversion mutation and hybrid mutation. As an optional task, we have chosen to program the selection processes of proportional and tournament selection aside from the already given stochastic uniform sampling. These operators and processes will be tested by comparing solution speeds, minimal tour distances, genetic diversity, etcetera… We will also briefly discuss the performance of the adjacency representation with respect to these implementations.

# Benchmark Problems

In general, the optimal solution of the TSP’s is not known. That’s why it is crucial to have several benchmark problems, of which the global optimal solution has been found and of which the search space is fully explored. This allows a more profound comparison of the performance of different parameters, operators, etc… The following GA studies will therefore also be based on the solution of 5 of these benchmark problems. In particular, we will make use of benchmark problems with 131, 380, 662 and 711 nodes, and one with 41 nodes based on Belgium’s most important cities. For all of these, we know the optimal solution distance and its configuration. In bijlage de afbeeldingen met de oplossingen?

# GA Parameters and Performance

It is easily seen that by varying different solution parameters, the outcome of the genetic process will differ greatly. For example, when observing a natural evolution situation, the effects of population size on the genetic diversity among the animals is obvious. The same accounts for genetic algorithms, where the tweaking of the solution parameters is of utmost importance for acquiring strong results.

By running the program repeatedly experimenting with different parameter values, many conclusions can be drawn by studying the population and its dynamics. Parameters that have great influence on the performance are for example the population size, the number of generations allowed, the mutation probability, the crossover probability and the percentage of elitism solutions kept (juist?).

As for the size of the solutions population and the number of generations, it speaks for itself that increase of these parameters will improve the solution quality, but the computational effort will also increase with them. That’s why a compromise between these two should be found. A bigger population in each generation will allow more diversity within the population, and thus a wider spread among the search space, whereas a larger number of consecutive generations will allow a deeper search for locally optimal solutions.

Crossover probability will also affect the wideness of the search for solutions. As crossover introduces disorder and makes great changes to the genotype of the individuals, a greater probability will boost the genetic diversity among the population. This will allow a more diffuse exploration of the search space, and the general optimum will more likely be found. However, when the probability of crossover gets too high, it will also inhibit a more profound search for local optima. This is opposed to mutation, which makes mere small changes to the genotype, and a greater probability of mutation will aid in local fitness optimization.

Wat is die elite precies? Is het parent selection? Offspring selection? Het is toch parent selection he? Anders heb ik beneden nog wat foutjes gezegd bij de selection operators.

It speaks for itself that the use of heuristics such as a loop detection method will generally improve performance.

# Tour Representation

To make it possible to apply genetic operators to tour configuration, a way of representing this phenotype in genetic code is necessary. This representation will of course also affect the performance of the genetic algorithm, and which genetic operators are available. We will discuss three of these representations that use a vector form notation, together with their performance and properties.

## Adjacency Representation

The adjacency representation describes a tour with a list of n cities where city j is listed in position I if and only if the tour leads from city i to city j. The adjacency representation does allow a unique representation of each possible tour, and it also allows schemata analysis, but it has many disadvantages. The use of normal crossover operators will very likely introduce illegal tours in the population, and they have to be complemented with repair operators. The use of other crossover operators designed for the adjacency representation on the other hand will generally lead to very poor results.

The given MATLAB toolbox initially contained adjacency representation implementations and several operators. Because of its global poor performance, we will not discuss this representation to its full extent, and replace it with another, namely the path representation.

## Ordinal Representation

The ordinal representation also describes a tour with a list of n cities, but this time an ordered list of cities serving as a reference point is also used. The ordinal representation will again allow the use of the classical crossover operators. Unfortunately, it also offers generally poor results, and thus it will not be discussed in further detail.

## Path Representation

The most straightforward representation for describing TSP tours is the path representation. In the path representation, once again a list of n cities is used. In this list, the j-th element with value i denotes that the j-th city to be visited is city i. Also in this case the classical crossover operators will not be possible. Using this representation however, crossover operators have been designed that do generate strong results. For this reason we have implemented the path representation for the TSP, together with several crossover and mutation operators. In a following section, we will discuss the implementation and the performance of these operators.

# GA Operators: Adjacency Representation

As a reference for the path representation crossover operators, it is best to have a measure of the performance of the adjacency representation and its available crossover operator, the alternating edges crossover. As for the mutation operators, this is not important because the mutation operators discussed are applicable to both representations. We will set the reference for the crossover operators by using a mutation rate of 10%, the mutation being simple inversion, 90% crossover probability, 5% elitism using Stochastic Uniform Sampling for selection, a random initial population and no further applied heuristics. The crossover probability is set high so that the outcome of the algorithm is very much characterized by the applied crossover operator. The number of individuals in each generation is 50, and the algorithm is stopped after 100 generations.

Next we will test the performance by running the genetic algorithm several times for each of the 5 benchmark problems. To be precise, we will run the algorithm 5 times for each problem, and take a record of the needed calculation time and the distance of the best solution for every run. By repeatedly solving different problems, we will also get a measure of how much the applied operators depend on the problem size. And, because benchmark problems have been used, we can also express the quality of the best solution in a relative manner with respect to the global optimum solution. As a measure for this, we divide the distance of the best found solution by the distance of the global optimum. This percentage gives thus a measure of how close we are to the global optimum. We will call this measure xMin in our tables.

The results for the alternating edges crossover are given in table X. The algorithm has been run only 3 times instead of 5 for each problem, simply because otherwise we would have to wait way too long to get the results for the larger problems. Performance comparison with others is not possible yet, but several remarks can readily be made. For example, it can be seen that the computation time rises less than linearly versus the problem size. It can also be seen that we drift away from the optimal solution as the problem size increases because the amount of generations allowed becomes too limited to give good results. We conclude by mentioning that the average solving time and the average minimal distance are of very poor quality (as will be seen when compared to the other crossover operators). This is because the alternating edges crossover also destroys the good subtours in the tour while mixing individuals.

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Table X: Alternating Edges Crossover** | | | | | | | | | | |
|  | *Belgium 41* | | *131* | | *380* | | *662* | | *711* | |
| **Run** | **Time** | **Distance** | **Time** | **Distance** | **Time** | **Distance** | **Time** | **Distance** | **Time** | **Distance** |
| 1 | 24 | 1486 | 32,3 | 3250 | 69,5 | 21046 | 110 | 45983 | 111,3 | 56082 |
| 2 | 23,7 | 1635 | 31,9 | 3338 | 68,8 | 21757 | 105,9 | 46538 | 114,5 | 56283 |
| 3 | 23,6 | 1519 | 32,3 | 3380 | 70,3 | 21781 | 107,3 | 46732 | 119,1 | 55268 |
| Average | 23,7 | 1546,6 | 32,1 | 3322 | 69,5 | 21528 | 107,7 | 46417 | 114,9 | 55877 |
| Best | 23,6 | 1486 | 31,9 | 3250 | 68,8 | 21046 | 105,9 | 45983 | 111,3 | 55268 |
| xMin(%) |  |  |  | 17,36 |  | 7,7 |  | 5,47 |  | 5,64 |

# GA Operators: Path Representation

Now we will discuss the implementation and performance of the path representation and its operators. As for the implementation, we have added the code for every operator in the appendix A. To make the programs fully functional for the path representation, many changes had to be done to the different files. To save the reader from all of these, we will only include the MATLAB code that contains the essence of the genetic operators in the appendix. For performance measurements, we have again set up tables with results of several runs of the algorithm, which can be analyzed and compared with one another.

## Crossover Operators

Again, for the crossover operators we have ran the algorithm 5 times per problem, using the same configuration as in the previous paragraph to emphasize crossover results.

### Order Crossover (OX)

Our MATLAB implementation of the Order Crossover can be found in Appendix A.1.1. The results of the performance measurements are given in table X+1. As we can see the computation times of the OX are drastically smaller than those of the Edge Recombination Crossover, especially for the larger problems. The problem size dependence of the OX is way less important than for the previous case, the exponent of the calculation time growth versus problem size is clearly a lot smaller than 1, and thus even further away from a linear trend. We can deduce that the problem size dependence is mainly affected by the used crossover operator, because in the previous case time cost rose firmly, while now it remains nearly constant. This means that running the algorithm without crossover would be mainly independent of the size of the problem. This also means that the OX operator only has a very small contribution to the overall time cost of the run of the algorithm.

We also see that the found solutions are a lot closer to the global optimum. The OX seems to be pretty fast and efficient. This is probably because the OX operator makes use of the essential property of the path representation, which is that the order of the cities is most important, and not their position in the genetic code.

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Table X+1: OX Crossover** | | | | | | | | | | |
|  | *Belgium 41* | | *131* | | *380* | | *662* | | *711* | |
| **Run** | **Time** | **Distance** | **Time** | **Distance** | **Time** | **Distance** | **Time** | **Distance** | **Time** | **Distance** |
| 1 | 6,8 | 952 | 8,3 | 2332 | 7,8 | 17290 | 8,3 | 38775 | 8,5 | 48131 |
| 2 | 6,8 | 931 | 7,1 | 2314 | 7,9 | 16928 | 8,2 | 39335 | 8,6 | 48709 |
| 3 | 6,8 | 1002 | 7 | 2389 | 7,7 | 17081 | 8,3 | 39317 | 8,6 | 47980 |
| 4 | 6,8 | 985 | 7 | 2309 | 7,8 | 17066 | 8,4 | 38891 | 8,6 | 48180 |
| 5 | 6,9 | 928 | 7 | 2186 | 7,8 | 16670 | 8,3 | 38486 | 8,6 | 48918 |
| Average | 6,82 | 959,6 | 7,28 | 2306 | 7,8 | 17007 | 8,3 | 38960,8 | 8,58 | 48383,6 |
| Best | 6,8 | 928 | 7 | 2186 | 7,7 | 16670 | 8,2 | 38486 | 8,5 | 47980 |
| xMin(%) |  |  |  | 25,84 |  | 9,73 |  | 6,53 |  | 6,49 |

### Cyclic Crossover (CX)

Our MATLAB implementation of the Cyclic Crossover can be found in Appendix A.1.2. The results of the performance measurements are given in table X+2. When looking at the calculation times, we see that the cyclic crossover is slower than the order crossover. Time cost rises faster with problem size, which means that the CX operator clearly demands more Also, when comparing the best solutions found to these of the OX, we can also deduce that the CX does not find as good solutions as the OX.

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Table X+2: CX Crossover** | | | | | | | | | | |
|  | *Belgium 41* | | *131* | | *380* | | *662* | | *711* | |
| **Run** | **Time** | **Distance** | **Time** | **Distance** | **Time** | **Distance** | **Time** | **Distance** | **Time** | **Distance** |
| 1 | 6,6 | 1381 | 6,9 | 3032 | 8 | 21334 | 9,6 | 44788 | 9,3 | 56008 |
| 2 | 6,7 | 1223 | 7,1 | 2847 | 8,2 | 21104 | 10,3 | 43846 | 10,2 | 53786 |
| 3 | 6,6 | 1244 | 7 | 2828 | 8,4 | 20150 | 10 | 44187 | 10,2 | 55655 |
| 4 | 6,8 | 1354 | 7 | 2965 | 8,2 | 20936 | 9,5 | 46068 | 10,9 | 55939 |
| 5 | 6,6 | 1195 | 7 | 2858 | 8,6 | 20274 | 10,6 | 43828 | 9,3 | 55974 |
| Average | 6,66 | 1279,4 | 7 | 2906 | 8,28 | 20759,6 | 10 | 44543,4 | 9,98 | 55472,4 |
| Best | 6,6 | 1195 | 6,9 | 2828 | 8 | 20150 | 9,5 | 43828 | 9,3 | 53786 |
| xMin(%) |  |  |  | 20 |  | 8,05 |  | 5,74 |  | 5,79 |

### Edge Recombination Crossover (ERX)

Our MATLAB implementation of the Edge Recombination Crossover can be found in Appendix A.1.3. Hier zou ik nog wel wat informatie over de implementatie geven. The results of the performance measurements are given in table X+3. It is clear from the time measurements that the ERX is a heavy duty crossover operator. The time cost for ERX is enormous. This is also the reason why we have not ran the algorithm as many times for each problem size. When looking at the quality of the best TSP solutions however, it is obvious that the time cost is not for nothing. ERX gives by far the best tour results for the same number of generations when compared to the other crossover operators discussed.

This is due to the properties of the ERX. Recall that OX tried to preserve the good subtours in the individuals. However, when applying OX a lot of new, random, edges are introduced in the offspring. ERX aims to work around this disadvantage, and tries to preserve as much edges as possible. The fact that ERX clearly gives better minimal distance solutions, proves that in a TSP context, it is most valuable to try and preserve the high fitness edges, rather than trying to preserve some subtours.

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Table X+3: ERX Crossover** | | | | | | | | | | |
|  | *Belgium 41* | | *131* | | *380* | | *662* | | *711* | |
| **Run** | **Time** | **Distance** | **Time** | **Distance** | **Time** | **Distance** | **Time** | **Distance** | **Time** | **Distance** |
| 1 | 9,9 | 889 | 18,4 | 1824 | 119,3 | 13298 | 272,8 | 23010 | 315 | 32447 |
| 2 | 10 | 926 | 18,8 | 1573 | 118,7 | 14512 | 277,7 | 22584 |  |  |
| 3 | 9,9 | 1021 | 18,9 | 1716 | 120,4 | 13190 |  |  |  |  |
| 4 | 10,6 | 971 | 18,7 | 1619 |  |  |  |  |  |  |
| 5 | 10 | 979 |  |  |  |  |  |  |  |  |
| Average | 10,08 | 957,2 | 18,7 | 1683 | 119,7 | 13666 | 275,2 | 22797 | 315 | 32447 |
| Best | 9,9 | 889 | 18,4 | 1573 | 118,7 | 13190 | 272,8 | 22584 | 315 | 32447 |
| xMin(%) |  |  |  | 35,97 |  | 12,3 |  | 11,14 |  | 9,61 |

### Enhanced Edge Recombination Crossover (EERX)

Our MATLAB implementation of the Enhanced Edge Recombination Crossover can also be found in Appendix A.1.3. Hier zou ik nog wel wat informatie over de implementatie geven. The results of the performance measurements are given in table X+4. From the results of the table, we can deduce that the EERX does not differ much from the ERX. When we look at the time result, we can see that the time performance is just slightly better than the ERX. As for the quality of the best solutions, no significant improvements are seen. We conclude that EERX only enhances ERX slightly when considering time cost, but in general they are very much alike.

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Table X+4: EERX Crossover** | | | | | | | | | | |
|  | *Belgium 41* | | *131* | | *380* | | *662* | | *711* | |
| **Run** | **Time** | **Distance** | **Time** | **Distance** | **Time** | **Distance** | **Time** | **Distance** | **Time** | **Distance** |
| 1 | 10,9 | 892 | 22 | 1619 | 115,7 | 13947 | 267,7 | 24128 | 308 | 34244 |
| 2 | 10,7 | 946 | 21,4 | 1699 | 118,8 | 12968 | 281,7 | 24498 |  |  |
| 3 | 10,8 | 964 | 22 | 1780 | 119,8 | 13688 |  |  |  |  |
| 4 | 11 | 901 | 21,2 | 1586 |  |  |  |  |  |  |
| 5 | 10,8 | 896 |  |  |  |  |  |  |  |  |
| Average | 10,84 | 919,8 | 21,65 | 1671 | 118,1 | 13534 | 274,7 | 24313 | 308 | 34244 |
| Best | 10,7 | 892 | 21,2 | 1586 | 115,7 | 12968 | 267,7 | 24128 | 308 | 34244 |
| xMin(%) |  |  |  | 35,59 |  | 12,5 |  | 10,42 |  | 9,1 |

### Partially Matched Crossover (PMX)

Our MATLAB implementation of the partially matched crossover can be found in Appendix A.1.4. The results of the performance measurements are given in table X+5. The time results of the PMX are rather poor. The calculation time heavily depends on the problem size, which means that the PMX operator is quite time-demanding. As for the solution quality, PMX provides approximately equally good results compared to the OX, but nevertheless at a higher time cost.

PMX and CX are position preserving crossover operators, as opposed to the order crossover, which rather preserves sequences instead of positions. Since PMX and CX perform significantly worse when compared to OX, we can conclude that it is a better idea to preserve fitter subtours instead of fitter positions in the genes of the individuals.

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Table X+5: PMX Crossover** | | | | | | | | | | |
|  | *Belgium 41* | | *131* | | *380* | | *662* | | *711* | |
| **Run** | **Time** | **Distance** | **Time** | **Distance** | **Time** | **Distance** | **Time** | **Distance** | **Time** | **Distance** |
| 1 | 7 | 1090 | 8,7 | 2462 | 16 | 18317 | 30,6 | 39367 | 35,2 | 48260 |
| 2 | 7,1 | 1226 | 8,7 | 2242 | 16,5 | 18581 | 31,3 | 39728 | 34,2 | 50096 |
| 3 | 7 | 984 | 8,6 | 2535 | 16,6 | 18315 | 32,1 | 40302 | 35,1 | 50821 |
| 4 | 7,1 | 1238 | 8,5 | 2404 | 16,6 | 18373 | 32,7 | 41374 | 33,3 | 50139 |
| 5 | 7,2 | 910 | 8,6 | 2232 | 16,7 | 18244 | 29,6 | 39863 | 33,5 | 49296 |
| Average | 7,08 | 1089,6 | 8,62 | 2375 | 16,48 | 18366 | 31,26 | 40126,8 | 34,26 | 49722,4 |
| Best | 7 | 910 | 8,5 | 2232 | 16 | 18244 | 29,6 | 39367 | 33,3 | 48260 |
| xMin(%) |  |  |  | 25,32 |  | 8,89 |  | 6,39 |  | 6,46 |

### Hybrid Crossover

Our MATLAB implementation of the hybrid crossover can be found in Appendix A.1.5 Moet er nog in gezet worden. The results of the performance measurements are given in table X+6. The hybrid crossover consists of randomly choosing one of the five included operators (these are the OX, PMX, CX, EERX or leaving the genotype unchanged) with an equal probability for each operator, and next applying the chosen operator.

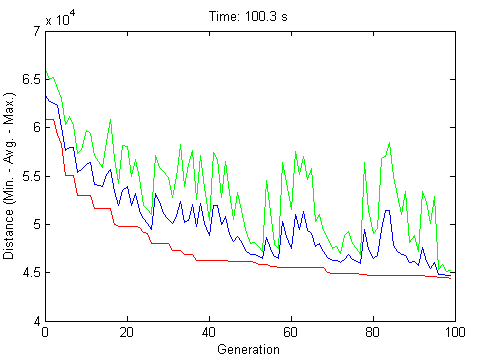
The hybrid crossover obviously is significantly slower than the other crossover operators, because of the presence of the heavy duty ERX and EERX. The time cost is not for nothing, because it does seem to perform slightly better than the order crossover, which is consistent with the ERX and EERX results. However, practically speaking, the quality of the best solutions is improved with a mere small percentage when compared to the order crossover, while the time cost is hugely increased for bigger problem sizes. The order crossover is still a better general solution than using these different crossover operators all together.

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Table X+6: Hybrid Crossover** | | | | | | | | | | |
|  | *Belgium 41* | | *131* | | *380* | | *662* | | *711* | |
| **Run** | **Time** | **Distance** | **Time** | **Distance** | **Time** | **Distance** | **Time** | **Distance** | **Time** | **Distance** |
| 1 | 8,2 | 1215 | 12,2 | 2360 | 39,6 | 17623 | 79 | 36022 | 100,3 | 44429 |
| 2 | 8,1 | 1200 | 12 | 2438 | 41,4 | 18668 | 82,5 | 36083 |  |  |
| 3 | 8,3 | 1097 | 11,9 | 2478 | 42,2 | 18579 |  |  |  |  |
| 4 | 8,3 | 1105 | 12,1 | 2295 |  |  |  |  |  |  |
| 5 | 8,1 | 1164 |  |  |  |  |  |  |  |  |
| Average | 8,2 | 1156,2 | 12,05 | 2392,75 | 41,0 | 18290 | 80,75 | 36052,5 | 100,3 | 44429 |
| Best | 8,1 | 1097 | 11,9 | 2295 | 39,6 | 17623 | 79 | 36022 | 100,3 | 44429 |
| xMin(%) |  |  |  | 24,57 |  | 9,2 |  | 6,98 |  | 7,01 |

It can still be questioned whether the outcome of the algorithm using hybrid crossover is a mere average of the outcome of all the crossover operators separately, or whether several crossover operators tend to cooperate positively, or tend to counteract each other. This cannot be deduced from the result table alone, and thus we leave this outside of our scope.

Misschien kunnen we wel nog iets afleiden uit de volgende grafiek?

Zouden we ook niet vaker zo’n grafiek includeren om nog wat extra commentaar te geven? Ook zo kunnen we nog wat verder ingaan op genetische diversiteit en dergelijke.



## Mutation Operators

In this section we will discuss the implementation and the performance of the chosen mutation operators. These are the simple inversion mutation, the inversion mutation, the insertion mutation, the exchange mutation, and the hybrid mutation. The most important code will be shown again in appendix A.

Similar to the crossover operators, we will compare the performances using tables with time and best solution measurements. This time however, the genetic algorithm parameters are different, to emphasize the effects of the choice of mutation operator. Again, the algorithm runs for 100 generations, 50 individuals each, 5% elitism with stochastic uniform sampling, and no extra heuristics applied. But now the crossover rate is reduced to 10%, using order crossover, and the mutation rate is boosted to 90%.

Also, because crossover is less important now, there will be a smaller variance in the time and distance measures for each run. That is why we have run the algorithm only 3 times for each problem size.

### Simple Inversion Mutation

Our MATLAB implementation of the simple inversion mutation can be found in Appendix A.2.1. The results of the performance measurements are given in table X+7. As we can see the variance of the time and distances is indeed smaller than in the case of a higher crossover probability. Comparison with other mutation operators is not possible yet, but what can be seen is the fact that the solution distances are significantly worse than for the measurements of the OX crossover, which happened with exactly the same operators but a higher crossover rate and lower mutation rate. This proves again that sufficient randomness has to be introduced into the genetics to make a decent exploration of the solution search space.

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Table X+7: Simple Inversion Mutation** | | | | | | | | | | |
|  | *Belgium 41* | | *131* | | *380* | | *662* | | *711* | |
| **Run** | **Time** | **Distance** | **Time** | **Distance** | **Time** | **Distance** | **Time** | **Distance** | **Time** | **Distance** |
| 1 | 6,8 | 1090 | 6,8 | 2530 | 7,5 | 19014 | 7,9 | 41125 | 8,1 | 53679 |
| 2 | 6,7 | 1090 | 6,8 | 2404 | 7,7 | 18897 | 8 | 42498 | 8 | 54119 |
| 3 | 6,7 | 1081 | 6,8 | 2496 | 7,4 | 18592 | 7,9 | 42802 | 8 | 52536 |
| Average | 6,7 | 1087 | 6,8 | 2476 | 7,5 | 18834 | 7,9 | 42141 | 8,0 | 53444 |
| Best | 6,7 | 1081 | 6,8 | 2404 | 7,4 | 18592 | 7,9 | 41125 | 8 | 52536 |
| xMin(%) |  |  |  | 23,47 |  | 8,73 |  | 6,11 |  | 5,93 |

### Inversion Mutation

Our MATLAB implementation of the inversion mutation can be found in Appendix A.2.2. The results of the performance measurements are given in table X+8. The measurements in the table show that the inversion mutation does not provide better solutions than the inversion mutation, the distance results are the same. What can be seen is that our implementation of this mutation operator requires slightly more calculation time than the previous operator. This is due to the fact that calculation of one extra random position has to be done.

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Table X+8: Inversion Mutation** | | | | | | | | | | |
|  | *Belgium 41* | | *131* | | *380* | | *662* | | *711* | |
| **Run** | **Time** | **Distance** | **Time** | **Distance** | **Time** | **Distance** | **Time** | **Distance** | **Time** | **Distance** |
| 1 | 7 | 1144 | 7,1 | 2454 | 7,5 | 18661 | 8,3 | 42392 | 8,2 | 52464 |
| 2 | 7,1 | 1200 | 7 | 2460 | 7,6 | 18942 | 8,1 | 42598 | 8,2 | 52096 |
| 3 | 6,9 | 1378 | 7 | 2523 | 7,6 | 19084 | 8 | 41498 | 8,2 | 51601 |
| Average | 7 | 1240 | 7 | 2479 | 7,6 | 18895 | 8,2 | 42162 | 8,2 | 52053 |
| Best | 6,9 | 1144 | 7 | 2454 | 7,5 | 18661 | 8 | 41498 | 8,2 | 51601 |
| xMin(%) |  |  |  | 22,27 |  | 8,39 |  | 5,93 |  | 6 |

### Insertion Mutation

Our MATLAB implementation of the insertion mutation can be found in Appendix A.2.3. The results of the performance measurements are given in table X+9. Also for the insertion mutation, no significant improvements in minimal distance are evident. As for the time cost, it seems that it is again slightly slower than the two inversion operators.

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Table X+9: Insertion Mutation** | | | | | | | | | | |
|  | *Belgium 41* | | *131* | | *380* | | *662* | | *711* | |
| **Run** | **Time** | **Distance** | **Time** | **Distance** | **Time** | **Distance** | **Time** | **Distance** | **Time** | **Distance** |
| 1 | 6,9 | 1173 | 7,2 | 2708 | 7,7 | 19089 | 8,2 | 42304 | 8,6 | 52498 |
| 2 | 6,9 | 1157 | 7,2 | 2430 | 7,9 | 18696 | 8,1 | 42091 | 8,4 | 52557 |
| 3 | 6,9 | 1329 | 7,1 | 2599 | 7,8 | 19219 | 8,2 | 42661 | 8,4 | 51922 |
| Average | 6,9 | 1219 | 7,1 | 2579 | 7,8 | 19001 | 8,1 | 42352 | 8,4 | 52325 |
| Best | 6,9 | 1157 | 7,1 | 2430 | 7,7 | 18696 | 8,1 | 42091 | 8,4 | 51922 |
| xMin(%) |  |  |  | 23,26 |  | 8,67 |  | 5,97 |  | 6 |

### Exchange Mutation

Our MATLAB implementation of the exchange mutation can be found in Appendix A.2.4. The results of the performance measurements are given in table X+10. We can see that the exchange mutation clearly is the fastest mutation operator. Again, there is no significant difference in solution quality.

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Table X+10: Exchange Mutation** | | | | | | | | | | |
|  | *Belgium 41* | | *131* | | *380* | | *662* | | *711* | |
| **Run** | **Time** | **Distance** | **Time** | **Distance** | **Time** | **Distance** | **Time** | **Distance** | **Time** | **Distance** |
| 1 | 6,9 | 1327 | 6,9 | 2765 | 7,7 | 18879 | 7,8 | 42210 | 7,9 | 52539 |
| 2 | 7 | 1328 | 6,9 | 2550 | 7,5 | 18901 | 7,9 | 41783 | 7,9 | 51372 |
| 3 | 6,8 | 1460 | 6,9 | 2629 | 7,4 | 19061 | 7,8 | 41971 | 8 | 52341 |
| Average | 6,9 | 1371 | 6,9 | 2648 | 7,5 | 18947 | 7,8 | 41988 | 7,9 | 52084 |
| Best | 6,8 | 1327 | 6,9 | 2550 | 7,4 | 18879 | 7,8 | 41783 | 7,9 | 51372 |
| xMin(%) |  |  |  | 22,12 |  | 8,59 |  | 6,02 |  | 6,06 |

### Hybrid Mutation

Our MATLAB implementation of the hybrid mutation can be found in Appendix A.2.5. Moet er nog in gezet worden! The results of the performance measurements are given in table X+11. The hybrid mutation consists of randomly choosing one of the five included operators (these are the simple inversion, inversion, insertion, exchange or leaving the genotype unchanged) with an equal probability for each operator, and next applying the chosen operator.

It seems that the hybrid mutation operator performs averagely when considering time cost, and also when considering solution quality. This is consistent with the fact that the hybrid mutation is mere a mix of several mutation operators.

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Table X+11: Hybrid Mutation** | | | | | | | | | | |
|  | *Belgium 41* | | *131* | | *380* | | *662* | | *711* | |
| **Run** | **Time** | **Distance** | **Time** | **Distance** | **Time** | **Distance** | **Time** | **Distance** | **Time** | **Distance** |
| 1 | 7,1 | 1136 | 7,2 | 2583 | 7,7 | 19171 | 8,2 | 42537 | 8,3 | 52367 |
| 2 | 7,3 | 975 | 7,2 | 2538 | 7,7 | 18240 | 8,2 | 42005 | 8,3 | 52428 |
| 3 | 7 | 1062 | 7,1 | 2639 | 7,9 | 19145 | 8,3 | 41484 | 8,2 | 50472 |
| Average | 7,1 | 1057 | 7,2 | 2586 | 7,8 | 18852 | 8,2 | 42008 | 8,3 | 51755 |
| Best | 7 | 975 | 7,1 | 2538 | 7,7 | 18240 | 8,2 | 41484 | 8,2 | 50472 |
| xMin(%) |  |  |  | 22,22 |  | 8,89 |  | 6,06 |  | 6,17 |

## Selection Operators

One of the fundaments of evolutionary computing is the selection principle. That is why we have chosen to also implement several selection operators, to see in what way the performance of the algorithm is affected by the choice of selection operator. The selection operators implemented are stochastic uniform sampling, which was already given in the toolbox, a roulette wheel selection and a tournament selection. All of these are parent selection operators.

Nog een klein beetje uitleg over selective pressure, genetische diversiteit, hoe de %elite omhoogdoen voor snelle premature convergence kan zorgen).

For obtaining the performance results for the selection operators, the algorithm was run 5 times per problem size, using a 90% crossover probability for the order crossover, a 10% mutation probability for the simple inversion mutation, 20% elitism to try to emphasize the effect of the selection operator, and 100 consecutive generations with 50 individuals each. No extra heuristics were applied.

### Stochastic Uniform Sampling

The stochastic uniform sampling is the parent selection technique that was already given in the MATLAB toolbox. The results of its performance measurements are given in table X+12. Hier zou ik toch nog wat schrijven over de werking van SUS (ook omdat er geen code is), en wat het verschil is met gewoon willekeurige individuen selecteren (zoals de vraag was bij de presentatie). Hier heb ik echter geen idee van. Comparison with other selection operators is not possible yet, but we can compare with table X+1, where the same parameters and operators were used, except for the lower elitism percentage of 5%. When comparing them, we see that boosting the elitism gives rise to longer calculation times. Als stochastic uniform sampling meer bewerkingen vereist dan gewoon willekeurige individuen selecteren als parents, dan kunnen we dit vermelden. Ik weet echter wel niets van deze selection operator en zijn implementatie. The quality of the solutions does not seem to improve by this boost of elitism from 5 to 20%.

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Table X+12: Stochastic Uniform Sampling** | | | | | | | | | | |
|  | *Belgium 41* | | *131* | | *380* | | *662* | | *711* | |
| **Run** | **Time** | **Distance** | **Time** | **Distance** | **Time** | **Distance** | **Time** | **Distance** | **Time** | **Distance** |
| 1 | 7,2 | 951 | 7,3 | 2268 | 8,1 | 17675 | 8,7 | 39012 | 8,7 | 47856 |
| 2 | 7,1 | 1048 | 7,3 | 2357 | 8,1 | 17451 | 8,6 | 39186 | 8,6 | 49247 |
| 3 | 6,8 | 927 | 8 | 1979 | 8,2 | 17274 | 8,5 | 38810 | 8,7 | 47947 |
| 4 | 7,2 | 992 | 7,4 | 2329 | 8,1 | 17879 | 8,6 | 40101 | 8,9 | 47876 |
| 5 | 7,2 | 1097 | 7,3 | 2306 | 8,1 | 16838 | 8,8 | 39240 | 8,8 | 47260 |
| Average | 7,1 | 1003 | 7,46 | 2247,8 | 8,12 | 17423,4 | 8,64 | 39269,8 | 8,74 | 48037,2 |
| Best | 6,8 | 927 | 7,3 | 1979 | 8,1 | 16838 | 8,5 | 38810 | 8,6 | 47260 |
| xMin(%) |  |  |  | 28,57 |  | 9,63 |  | 6,48 |  | 6,59 |

### Proportional Selection

Our MATLAB implementation of the proportional selection operator or roulette wheel can be found in Appendix A.X.X. Moet er nog in gezet worden! The results of the performance measurements are given in table X+13. There is no real difference in time cost for this operator when compared to the stochastic uniform sampling. Also, the quality of the solutions is more or less the same. The second data point for the 41-cities problem, is an example of a premature convergence of the genetic algorithm. The chance of this happening is increased by raising the elitism parameter, since genetic diversity is suppressed.

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Table X+13: Proportional Selection** | | | | | | | | | | |
|  | *Belgium 41* | | *131* | | *380* | | *662* | | *711* | |
| **Run** | **Time** | **Distance** | **Time** | **Distance** | **Time** | **Distance** | **Time** | **Distance** | **Time** | **Distance** |
| 1 | 7,3 | 973 | 7,4 | 2345 | 8,2 | 17723 | 8,8 | 40211 | 8,8 | 47788 |
| 2 | 4,4 | 1255 | 7,5 | 2357 | 8,1 | 17534 | 8,8 | 38354 | 8,8 | 48266 |
| 3 | 7,1 | 1043 | 7,4 | 2241 | 8,4 | 18026 | 8,5 | 39751 | 8,7 | 49763 |
| 4 | 6,1 | 1027 | 7,3 | 2281 | 8,1 | 17928 | 8,6 | 39514 | 8,9 | 49749 |
| 5 | 7,2 | 1005 | 7,3 | 2341 | 8 | 18087 | 8,6 | 39514 | 8,8 | 49030 |
| Average | 6,42 | 1060,6 | 7,38 | 2313 | 8,16 | 17859,6 | 8,66 | 39468,8 | 8,8 | 48919,2 |
| Best | 4,4 | 973 | 7,3 | 2241 | 8 | 17534 | 8,5 | 38354 | 8,7 | 47788 |
| xMin(%) |  |  |  | 25,19 |  | 9,25 |  | 6,55 |  | 6,52 |

### Tournament Selection

Our MATLAB implementation of the tournament selection can be found in Appendix A.X.X. Moet er nog in gezet worden! Since tournament selection also requires the definition of the variable K (size of the tournament subpopulation), we have chosen a value of (Wat is de waarde van K hier?). Also, when running the algorithm for different mutation rates, we noticed that the tournament selection attained better results when the mutation rate was made larger. This was not the case for the two previous selection operators. Optimal results with tournament selection were attained when mutation rate was set to approximately 35%. This is probably due to character of the tournament selection. When mutation is boosted, the search space is explored more deeply. However, when mutation rates get too high, it becomes too difficult to preserve relevant building blocks in the genetic code. Tournament selection tends to buffer this effect, since winners are chosen from many different randomly chosen subpopulations, which provides a greater genetic diversity in the selected parents. This decreases the chance that the relevant and valuable building blocks die out.

The results of the performance measurements are then given in table X+14, with a mutation rate of 35%. We can see that running the algorithm has become more cost-expensive with respect to the two previous cases. This is due to the fact that mutation rate has been boosted, and more mutation operators have to be applied and calculated. We can also see that indeed the quality of the solutions is slightly better than in the two previous cases, thanks to the cooperation of mutation and tournament selection.

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Table X+14: Tournament Selection** | | | | | | | | | | |
|  | *Belgium 41* | | *131* | | *380* | | *662* | | *711* | |
| **Run** | **Time** | **Distance** | **Time** | **Distance** | **Time** | **Distance** | **Time** | **Distance** | **Time** | **Distance** |
| 1 | 7,6 | 807 | 7,6 | 1912 | 8,2 | 15656 | 8,8 | 37581 | 9 | 46020 |
| 2 | 7,4 | 829 | 7,6 | 2021 | 8,6 | 15619 | 8,8 | 36780 | 9,1 | 47460 |
| 3 | 7,4 | 862 | 7,6 | 1919 | 8,2 | 15976 | 8,9 | 36479 | 9,1 | 45846 |
| 4 | 7,5 | 864 | 7,5 | 1895 | 8,3 | 15883 | 8,9 | 36939 | 9,3 | 47556 |
| 5 | 7,5 | 899 | 7,6 | 1927 | 8,3 | 16227 | 9 | 38224 | 9,4 | 45521 |
| Average | 7,48 | 852,2 | 7,58 | 1934,8 | 8,32 | 15872,2 | 8,88 | 37200,6 | 9,18 | 46480,6 |
| Best | 7,4 | 807 | 7,5 | 1895 | 8,2 | 15619 | 8,8 | 36479 | 9 | 45521 |
| xMin(%) |  |  |  | 29,76 |  | 10,38 |  | 6,9 |  | 6,85 |

# Heuristic Approach

A totally different approach to these combinatorial problems than using genetic algorithms is to use a heuristic approach. As opposed to the genetic algorithms, in which stochastic aspects are important, heuristic solutions are fully deterministic. To be able to compare the performance of genetic algorithms to other approaches in general, we also implemented one such heuristic solution, namely the Nearest Neighbor heuristic. The MATLAB implementation of this heuristic can be found in Appendix A.X.X. It works as follows: Nog wat over de implementatie en werking? Table X+15 shows the results attained by the Nearest Neighbor heuristic. Since it is a deterministic approach, only one run needs to be done for each TSP problem. The only difference between different runs lies in the initial population, which did not seem to cause any significant variation in the solution.

It seems that this heuristic approach provides significantly different results than the genetic approach. The calculation times as well as the minimal distance are of a whole different order. It seems that the heuristic approach definitely beats the genetic algorithms as a quick solution method to the TSP to gain decent results, and to gain them fast.

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Table X+15: Nearest Neighbor Heuristic** | | | | | | | | | | |
|  | *Belgium 41* | | *131* | | *380* | | *662* | | *711* | |
| **Run** | **Time** | **Distance** | **Time** | **Distance** | **Time** | **Distance** | **Time** | **Distance** | **Time** | **Distance** |
| 1 | 0,3 | 730 | 0,6 | 685 | 1,3 | 2007 | 3,3 | 3111 | 3,4 | 3765 |
| xMin(%) |  |  |  | 82,34 |  | 80,77 |  | 80,78 |  | 82,74 |

The drawbacks of the Nearest Neighbor are however also noteworthy. Since it is deterministic, it will never be able to calculate smaller tours than the ones given in the table. It is a good solution to gain quick and good results, but a genetic algorithm can be tweaked and can be given a lot of time (great number of generations, individuals,…) to find solutions that approach the globally optimal solution to 90%, 95%, or eventually even 100%. We must also bear in mind that the implementation of a heuristic such as this one demands specific knowledge of the problem. Genetic algorithms intrinsically know nothing of the problem that they are solving. That is why the same genetic algorithms can also be applied to a great variety of problems, while the Nearest Neighbor heuristic for example is very problem specific.

That’s why it can be a good deal to combine the powers of both approaches. For example, a heuristic approach such as this one can be used to process the initial population of a genetic algorithm so that it is closer to better results from the start, without too much time cost. Starting from this processed population, the genetic algorithm can try to find even better solutions. In this combination, we should however be careful that the initial population still maintains the necessary genetic diversity etcetera, so that the genetic algorithm can still make a significant contribution to the solution quality. This definitely demands some attention, since the heuristics tend to create very similar outputs, and thus tend to destroy genetic diversity.

# Conclusion

\* Over de implementatie

\* Wat zeggen over de optimale parameters

As for the crossover operators, our results give the following hierarchies in performance. When considering speed, the OX is definitely the fastest, followed by the CX, and the slower PMX. The slowest are by far the powerful EERX and its earlier, even slower version, the regular ERX. When considering only the quality of the best solutions for the same number of allowed generations, the best crossovers are clearly the EERX and ERX. Next in order is the fast subtour preserving OX, followed by the position preserving PMX and CX. The hybrid crossover seems to perform averagely with respect to all the available crossover operators.

Within the mutation operators, differences in performance were much less significant. When comparing the quality of the solutions, there were no real differences in the smallest calculated distance. The results were more or less the same. When considering the computational speed however, we could deduce that the exchange mutation was the fastest operator, followed by the simple inversion, and next the slightly slower inversion which needs to calculate one extra pseudorandom number, and at last the slowest insertion mutation. Again, the hybrid crossover is an average of the available operators.

The comparison of the three different selection operators also led to several conclusions. When comparing purely the selection operator however, the differences in performance were again not very distinct. We could only see that the tournament selection operator works slightly slower than the stochastic uniform sampling and the proportional selection, which work at more or less the same speed. If we compared only the different selection operators, also no differences in solution quality came to view. However, the tournament selection seems to outperform the other operators when working in a high mutation environment.

Considering the available crossover operators and mutation operators, it is very clear that the path representation is a massive improvement with respect to the adjacency representation. It is a more natural way of representing the phenotype of the tour, it allows crossover operators that have way better computational speeds, and it also allows operators that provide solutions that are much closer to the globally optimal solution. The path representation clearly beats the adjacency representation at every level.

\*Dan nog wat zeggen over de vergelijking tussen genetische algoritmes en heuristics

# References

Wat er volgens mij nog in moet komen:

Misschien nog de optimale run zoeken? Dus de optimale crossover met de optimale mutation en optimale parameters ed?

Meer uitleg telkens over de implementatie, en mss ook wat meer over hoe de implementatie de performance beinvloed heeft

Misschien nog bij een paar operators de grafiek weergeven van hoe de genetische diversiteit doorheen de generaties is (max avg min grafieken) en daar nog wat extra uitleg bij.