# Genetic Algorithms and Evolutionary Computing Project

Sebastian Wehkamp & Elias Regopoulos

January 6, 2020

## 1 Introduction

The aim of this project is to improve and evaluate a Genetic Algorithm which solves the Travelling Salesman Problem. A basic toolbox is provided which solves the problem in a way which is not optimal. This report is structured according to our methodology. First we implemented all of the improvements after which we performed several experiments and visualised the results.

## 2 Implementation

This section discusses the various improvements and modifications made to the program.

### 2.1 Stopping criterion

The first improvement is implementing a stopping criterion which performs early stopping. Currently there is already a stopping criterion implemented which by default triggers if 95% of the individuals within a generation have a fitness in a range of 1e−15. This however is almost never the case.

As an improvement on this we decided to implement a stopping criterion which triggers when for the last generations the best solution remains the same. The number of generations which give the same solution can be specified with respect to the total amount of generation e.g. 10%. Depending on what you set this value to it triggers much more often than the initial stopping criterion saving a large amount of generations. Setting the number to low however might result in stopping too early and missing out on valid better solutions. In our implementation it is possible to choose between no stopping criterion, the default stopping criterion, the improved stopping criterion discussed above, or both stopping criteria.

### 2.2 A new path representation

The template program already provides the adjacency path representation. A logical other representation to add would have been the path representation, however since this was such a standard option, and also already partly implemented (i.e. the conversion from/to the adjacency representation) we decided to opt for something different, the binary path representation.

#### 2.2.1 Motivation

Larranaga et al[1] study a variety of different representations in their 1999 paper. Representations more closely related to the adjacency one, like the ordinal representation, are linked to poor results, whereas the binary representation did show some success in smaller TSPs, like the ones in the assignment.

### 2.2.2 Representation

The individuals are first converted into the classical path representation. Then, each of the towns' IDs is converted to its binary form, with an equal number of digits ($\log_2 N$) for each city. From then on, the representation is treated like a single, continuous bit string. The conversion between the two representations is implemented in the *path2bin.m* and *bin2path.m* files.

### 2.2.3 Crossover

The crossover implemented is the classical, single-point crossover described by [1]. A single point is chosen, which serves as the "pivot" to recombine two parents into two offspring. This crossover point is not limited to the edges of each town's binary representation, and can thus result in cutting one of the numbers in half. This can result in illegal paths, containing duplicates, or even unknown town IDs. Therefore, a repair algorithm is needed to ensure the legality of the path at all times, and is applied directly after the Crossover. Both the crossover and the repair function are implemented in the *bin_recombin.m* file.

### 2.2.4 Mutation

Mutation is done in a classical manner, like to any other bit string: For each bit, there is a small chance that it will flip its value. This, too, can result in illegal paths, so a repair function is needed here as well. Since the repair function was needed for both the crossover and the mutation, and the underlying library didn't seem to include it, the whole algorithm was implemented from scratch for both operators. The mutation operator is implemented in the *bin_mutate.m* file.

### 2.2.5 Problems

During the implementation of the above operators, two bugs appeared that unfortunately could not be resolved in time. So while the code should work in isolation, we could not get it to work within the project, and - if binary representation is selected - will most likely result in an error.

## 2.3 Using a local optimisation heuristic

This task required us to implement a local optimisation heuristic which could improve the results. For our implementation we chose to make use of the euclidean distance heuristic using a crossover operation described by [2]. In their crossover operation they start at a randomly chosen town. The decision between the two possibilities of which town to visit next is made using probabilities based on the euclidean distance. If there are no unvisited options available a random town is chosen. The result is a crossover operation which always results in a valid path and if the two parents are identical then the child will also be identical with them. The choice of using probabilities based on the euclidean distance instead of simply choosing the closest town of the two is a deliberate decision. Choosing the nearest town is an implementation of a greedy algorithm which limits the robustness of the total algorithm.

## 2.4 Two new selection methods

Besides the already implemented Stochastical Universal Sampling we decided to implement two new sampling techniques, fitness proportional selection and tournament selection. The implementation of fitness proportional sampling is straightforward since the universal sampling was already provided. For the tournament selection we chose a fixed tournament size of five. Although both new selection methods are most likely slower in terms of performance they will hopefully result in an improved evolutionary process.

# 3 Results

This section will discuss the influence of various parameters on different datasets. Every parameter is scaled into twenty possible values which are all performed on three different datasets. Three kinds of datasets are used, small, medium, and a large one. For every possibility the shortest path is measured and plotted. All of the results are obtained using default parameters except for the variable at hand. Together with these results a moving average is plotted with a length of five.

## 3.1 The effect of population size

Beforehand the hypothesis is that a larger number of individuals results in larger search space resulting in better performance. Figure 1 shows the influence of population size one different sizes of datasets. As expected in general a larger population size results in better results. For all three datasets a large part of the improvements are obtained for the first 500 individuals. If the population grows larger than 500 the improvement is minimal or absent as the length of the shortest path stabilises. This is probably due to the fact that most of the search space is already covered with that amount of individuals.

A disadvantage of the increased population size is the increase in computational cost and time required to compute every generation. Both of these increase linearly as the computational cost function is equal to $N_{\text{gen}} * P_{\text{gen}}$.

## 3.2 The effect of number of generations.

The hypothesis is that a bigger amount of generations increases the solution as it has no disadvantage in terms of solution quality. The quality increase in expected since a larger number of generations should result in more refined solutions after every iteration. Figure 2 depicts the influence of number of generation on the three datasets. In the case of the small dataset shown in Figure 2a the shortest path length stabilises after 400 generations. This is probably to that there is no more refinement possible. For the medium and large datasets shown in Figure 2b and Figure 2c the solution quality still appears to increase for the largest population sizes. Similar to a larger population size increasing the number of generation results in linear increase in computational complexity.

## 3.3 Elitism

ELitism is a powerful improvement for the solution quality according to the literature but too much of it kills the diversity of the population and harms the evolutionary process. The hypothesis therefore is that at first an increase of elitism improves the solution quality after which it decrease again. The results are shown in Figure 3. As expected at first a larger percentage of elitism increases the quality of the solution after which the shortest path length increases again. This is especially clear for the medium and large dataset shown in Figure 3b and Figure 3c although something similar is happening for the small dataset shown in Figure 3a.

## 3.4 Mutation

The idea of mutation is to randomly modify individuals in order to keep the population diverse and to overcome local optima. If the mutation rate is set to high however it might remove good solutions from the population. Therefore similar to the elitism rate it is expected that at first it increases the solution quality after which it decreases again. Figure 4 shows the results which are a bit different from our expectations. For the small and medium sized datasets there is either a decrease or stable solution quality. For the large datasets the general trend is that it improves the solution quality with mutation rates up to 50%.

## 3.5 Crossover rate

The crossover rate regulates the speed of the evolutionary process similar to elitism. In the case of elitism a higher percentage results in saving the best individuals every generation whereas a low

crossover rate preserves random individuals. The results are depicted in Figure 5. In general the best results were obtained using a crossover rate of around 35% across all datasets.

### 3.6 Loop detection

The results of turning loop detection on and off are shown in Figure 6, Figure 7, and Figure 8 for the small, medium, and large dataset respectively. Overall not only better final results were achieved with loop detection on but they were also achieved much faster. For all three datasets a good solution was obtained much quicker compared to the results with loop detection turned off.

### 3.7 Early stopping

The early stopping rate is set to % of the maximum number of generations which seemed appropriate to us. The expected behaviour is that at the cost of finding a more optimal solution the computation stops early to save computational resources and time. The results shown in Figure 9, Figure 10, and Figure 11 show exactly the expected behaviour. The time and resources saved varies between 60% to 30% depending on the dataset size.

### 3.8 Local optimisation

The hypothesis for this parameter is that the local optimisation technique should lead to faster convergence and possibly a better solution. A risk of this variable is that it reduces the variety of the population. For this reason the mutation rate should be tweaked in order to keep the individuals diverse. The results are for the small, medium, and large dataset are shown in Figure 12, Figure 13, and Figure 14 respectively. The first notable difference between turning the local optimisation on and off is that the average is much further away from the shortest path length with the optimisation technique on. Although the final result is slightly worse with the local optimisation technique on it converges faster initially. This is as expected since cities close to each other are more likely to be chosen resulting in a better initial solution which might not be optimal in the end. So although it does not improve the result it is obtained faster. This property could be combined with early stopping to save even more resources than early stopping did without the local optimisation.

### 3.9 Critique

Although we attempted to isolate the influence of the variables tested and most results are as hypothesised, the problem of parameter tuning is much more complex. Many different combinations of variables are possible all of which can influence each other. Therefore it is not possible to select the best performing value from the above tested and say that that combination results in the best performance. Although possibilities exist using grid search or random search to automate this the execution time would be huge. For this reason we opted for the same values for all variables in all tests (the default ones) in order to allow for at least some form of comparison.

### 3.10 Selection methods

This section contain a performance comparison between stochastic universal sampling, fitness proportional selection, and tournament selection. In the case of tournament selection the size was set to five individuals. The results are shown in Figure 15, Figure 16, and Figure 17 for the small, medium, and large datasets respectively. All of the algorithms result in very similar performance overall. Compared to the other variables we tested like population size, number of generations, elitism, mutation rate, and crossover rate the selection methods seem to only marginally influence the final result.

### 3.11 Benchmark problems

Various benchmark problems were provided on which the performance of our algorithms should be tested. The goal is to obtain a result close to optimal using various parameters.

### 3.11.1 XQF131

This benchmark contains 131 cities covering the whole area arbitrarily and an optimal length of 564. The first test we did as a baseline was to run the algorithm with default settings which result in a shortest path length of 3141 which is really far from optimal. The first parameters to improve on were number of individuals and generations. Although in general for these parameters goes that more is better some of the other parameters' performance depends on this. Setting these to higher values already resulted in an improvement as the path length returned was around 2400. The next step was to enable loop detection as that always resulted in a better performance, set the mutation rate to around 20%, the crossover rate to 90%, and the % of elitism to 10. Early stopping was not used as we wanted to obtain a best as possible result and did not care about computational time. The selection methods did not show any relevant improvement and all of them performed similar so the default selection method, stochastically universal sampling, was used. The final result obtained was 591 which is a great improvement compared to the initial results however still not as close to the optimal result as we would have wanted.

### 3.11.2 BCL380

This dataset contains 380 cities which are spread all over the area but ordered in equally spaced columns. The optimal length for this path is 1621. Similar to the previous section first a baseline was established with default parameters. This baseline length was 12824 which is, as expected, far from optimal. The structure of this problem is such that in general the closest city is the correct city to select with a couple of exceptions. This use case is perfect for our new local optimisation technique which has a higher probability of selecting the closer city. As for the other parameters we tested several combinations but the result was that the best performing combination was the same as above so a mutation rate of 20%, crossover rate of 90%, and elitism set to 10%. The first noticeable thing is that the computation time takes long for this benchmark to complete. This is expected as the number of cities is relatively large and the local optimisation technique reduces computational performance. The result itself with a shortest path length of 5616 was a huge improvement compared to the baseline but the optimal path length was still far away.

As the result obtained was disappointing so the first thing we tried was to disable the local optimisation technique to see if our hypothesis was correct. The resulting path length was 4115 which, although better than the previous result, was still far from optimal. It appears that our hypothesis that this benchmark was a good use case for our local optimisation technique was wrong. We were unable to conclude why it was not performing as we expected.

The distribution plot indicated that we might be introducing too much noise with the high mutation rate. For this reason we lowered the the mutation rate to 5% and the crossover rate to 75%. The result we obtained with 4063 was better than the previous ones but still much longer than optimal. As a slower evolution appeared to result in better performance we slowed down the evolution even further by reducing the crossover rate to 60% and increase elitism to 25%. This steady evolution did increase the solution quality as the result was 3750. Although we would still consider this a bad result which still took a long time to compute we were unable to find a parameter combination resulting in a better performance.

### 3.11.3 XQL662 and RBX711

From the previous benchmark with 380 cities we got a disappointing result as it appears that the larger the number of cities the less optimal the performance. This is most likely due to the problem being more complex. In the previous benchmark section we concluded that a slow and steady evolution resulted in better performance compared to introducing more randomness. For this reason we first did the baseline which resulted in a length of 20404 and 30579 for XQL662 and RBX711 respectively. This is really far from the optimal length of 2513 and 3115. After this we tried the same parameters which gave the best result for the BCL380 problem. The best result obtained with slight tweaks was 7120 and 9480. So although we improved on the baseline, which is easy enough, we are still about three times longer than the optimal path length.

Similar to the previous benchmark there seems to be a trend in more cities resulting in worse performance. Although we tried many different variable configuration none of them seem to even get close to the known optimal path length. As we invested many hours in trying to obtain better results and none of them paying off we moved on to the last benchmark.

#### 3.11.4 Belgium Tour

The last benchmark problem was a much simpler problem with only 41 cities. So according to our previous results in which we concluded we perform better in simpler problems we should be able to obtain proper results in this case. The first step was to obtain a baseline which had a length of 1017. From the way the solution looked like it was already clear this is far from optimal. As this was a simpler problem more generations and individuals are feasible so the first thing we did was increase these. The mutation rate was set to 15%, crossover rate to 80%, and the elitism rate to 10%. The result was a path length of 430. Although we did not know the optimal path length for this benchmark this looks like a very good solution. A noticeable thing about this solution is that although we increased the number of generations to a 1000 this was completely unnecessary as the optimal solution was already reached after about a 100. This is the perfect case for early stopping or we could just lower the maximum number of generations.

## 4 Conclusion

In the end most of our time was spent on the implementation of the binary representation and performing the benchmarks. We tested many different combinations most of which had a long run time. The performance of our algorithm on smaller and easier problems was descent. If the number of cities grew to over a hundred the performance dropped drastically.

Another conclusion we could make was that a good set of variables on one problem does not result in a good solution for the next problem. Although it did provide is with some starting point from which we could make modifications to the variables for specific problems. This is completely in line with the literature where this has been studied in many different occasions.

An improvement for us for the next time would be to obtain the results in a more systematic way. We tried to do this by automating the performance tests of the separate variables but all of the interacting variables make it very hard show the effect of a single variable. Another thing is that we spent a lot of time on finding good parameters for the benchmark problems which might have been automated since the run time of every trial is very high for the complex problems.

## 5 Time spent on the project

- Sebastian: Implemented stopping criterion, local optimisation heuristic, and two new selection methods. Wrote introduction, wrote/performed section 2.1, 2.3, 2.4, 3.1-3.7, 3.9-3.12, performed the benchmarks.
- Elias: Implemented the binary representation (conversion from/to path representation, along with crossover and mutation operators) and wrote the respective sections.

## References

[1] R.H. Murga I. Inza P. Larranaga, C.M.H. Kuijpers and S.Dizdarevic. Genetic algorithms for the travelling salesman problem: A review of representations and operators. *Artificial Intelligence Review*, 13:129–170, 1999.

[2] Károly F Pál. Genetic algorithms for the traveling salesman problem based on a heuristic crossover operation. *Biological Cybernetics*, 69(5-6):539–546, 1993.

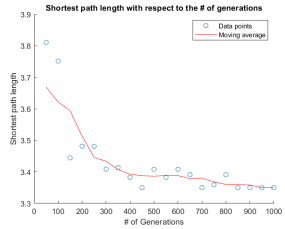
# A Appendix


(a) Results on the small dataset


(b) Results on the medium dataset

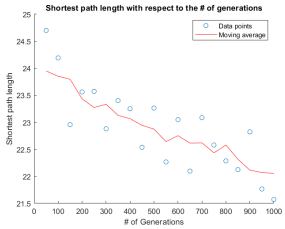

(c) Results on the large dataset

Figure 1: The influence of population size on different datasets




(a) Results on the small dataset

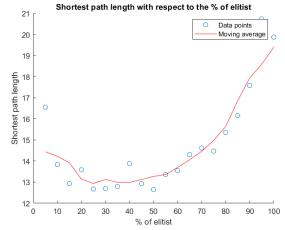

(b) Results on the medium dataset



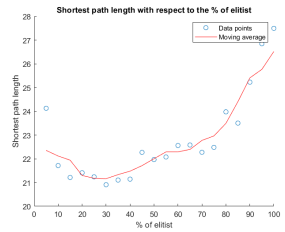

(c) Results on the large dataset

Figure 2: The influence of number of generations on different datasets


(a) Results on the small dataset

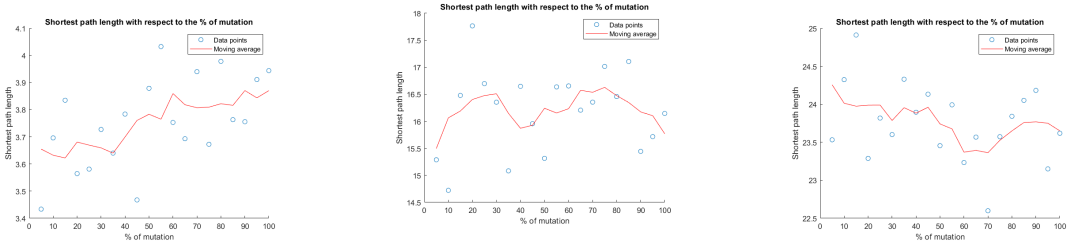



(b) Results on the medium dataset



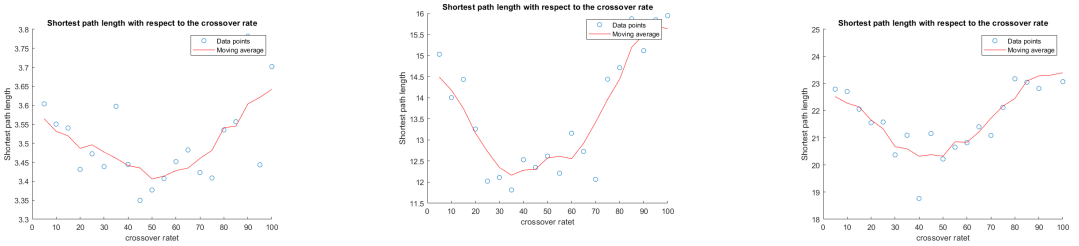

(c) Results on the large dataset

Figure 3: The influence of % of elitism on different datasets



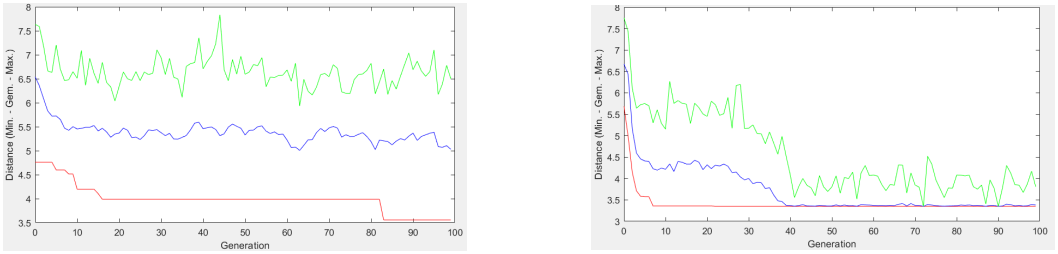

(a) Results on the small dataset (b) Results on the medium dataset (c) Results on the large dataset

Figure 4: The influence of mutation rate on different datasets



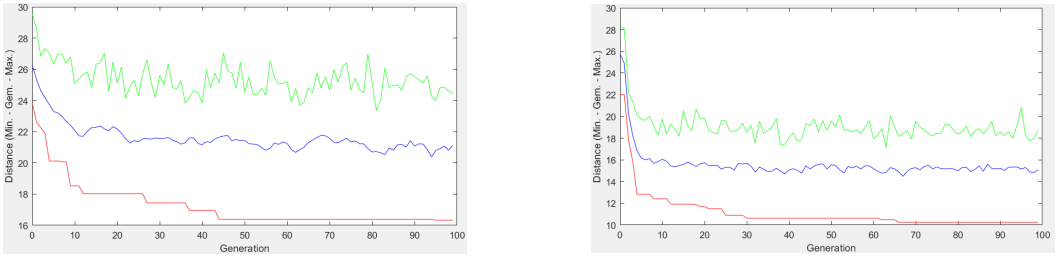

(a) Results on the small dataset (b) Results on the medium dataset (c) Results on the large dataset

Figure 5: The influence of Crossover rate on different datasets



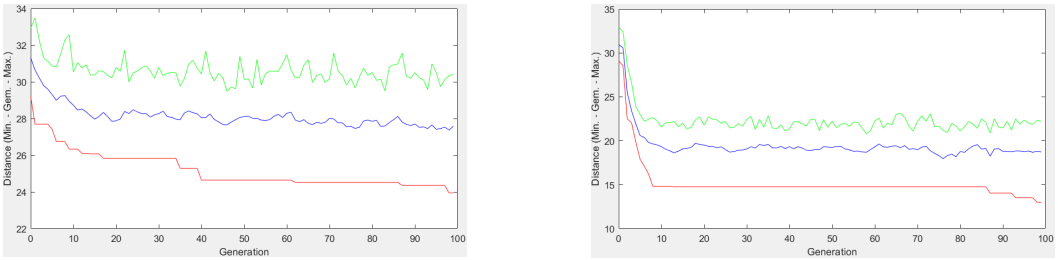

(a) Loop detection off. (b) Loop detection on.

Figure 6: The influence of loop detection on the small dataset.



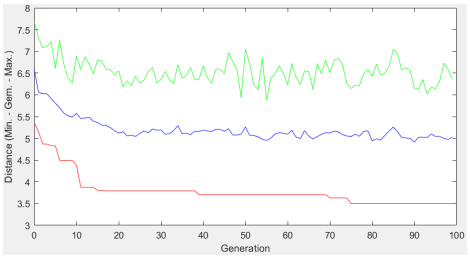

(a) Loop detection off. (b) Loop detection on.

Figure 7: The influence of loop detection on the medium dataset.

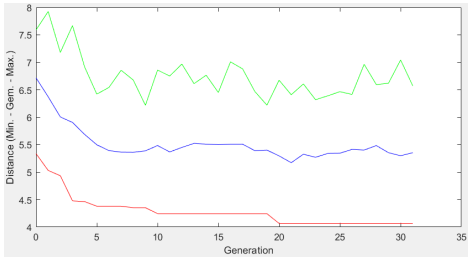



(a) Loop detection off. (b) Loop detection on.

Figure 8: The influence of loop detection on the large dataset.

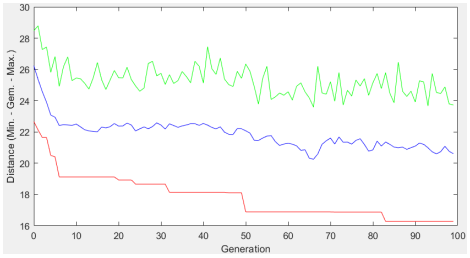



(a) Early stopping off.

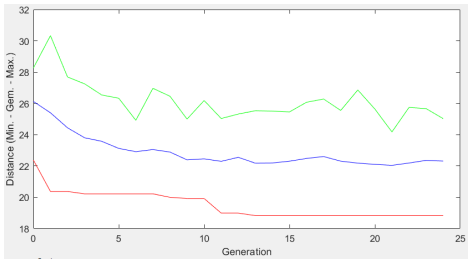



(b) Early stopping on.

Figure 9: The influence of early stopping on the small dataset.

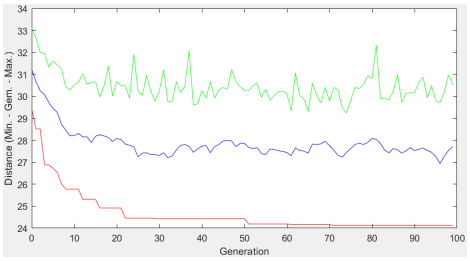



(a) Early stopping off.

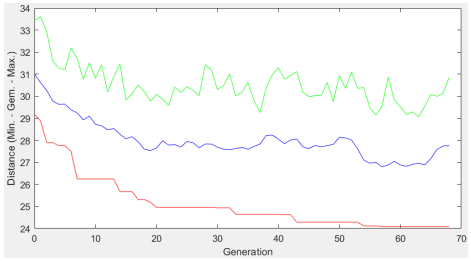



(b) Early stopping on.

Figure 10: The influence of early stopping on the medium dataset.

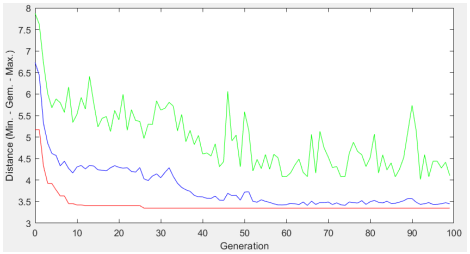



(a) Early stopping off.

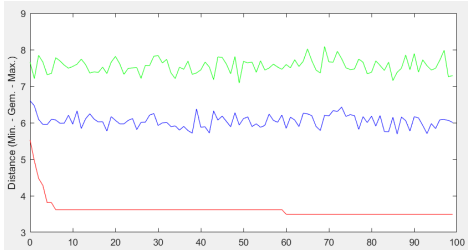



(b) Early stopping on.

Figure 11: The influence of early stopping on the large dataset.

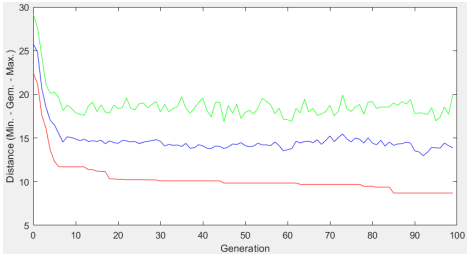



(a) Heuristic crossover off.

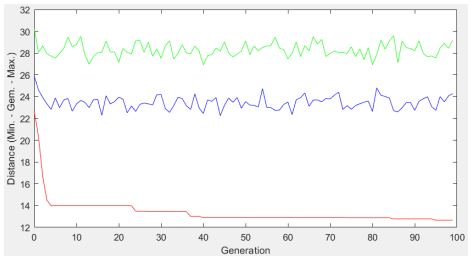



(b) Heuristic crossover on.

Figure 12: The influence of the local optimisation on the small dataset.

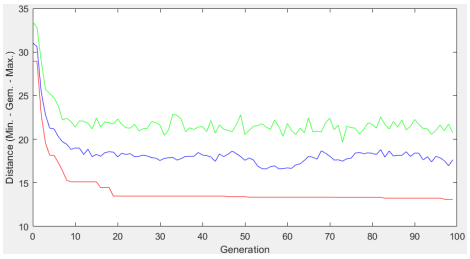



(a) Heuristic crossover off.

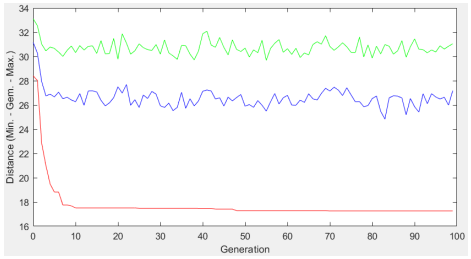



(b) Heuristic crossover on.

Figure 13: The influence of the local optimisation on the medium dataset.

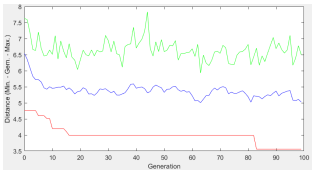



(a) Heuristic crossover off.

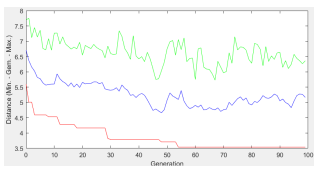



(b) Heuristic crossover on.

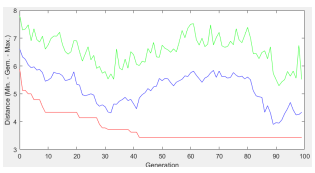
Figure 14: The influence of the local optimisation on the large dataset.



(a) Results of universal stochastic sampling

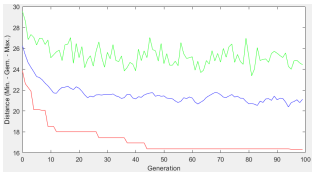


(b) Results of fitness proportional sampling

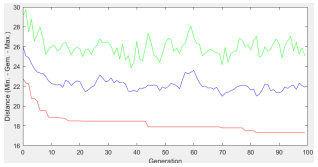


(c) Results of tournament selection with a size of five

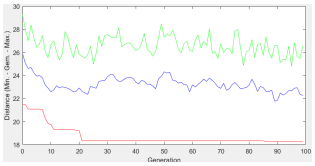
Figure 15: The influence of various selection methods on the small dataset.



(a) Results of universal stochastic sampling

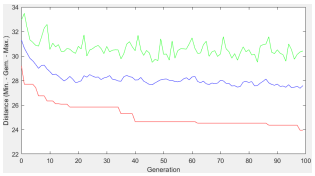


(b) Results of fitness proportional sampling

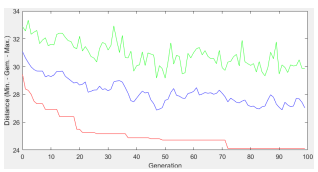


(c) Results of tournament selection with a size of five

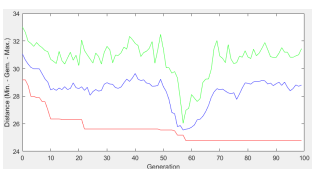
Figure 16: The influence of various selection methods on the medium sized dataset.



(a) Results of universal stochastic sampling



(b) Results of fitness proportional sampling



(c) Results of tournament selection with a size of five

Figure 17: The influence of various selection methods on the large dataset.