Genetic Algorithms and Evolutionary Computing Report - Project

Maxwell Szymanski, Felix Cammaerts

Abstract

This paper explores the use of genetic algorithms on two different representations of the travelling salesman problem. The influence of different parameter settings, as well as different parent selection and mutation methods are compared to see what settings suit different algorithm instances the best. The use of a suitable stopping criterion is also discussed in this paper.

1 Existing genetic algorithm

The performance criterium we opted for is the fitness of the best solution at the end of each run. The parameter sets we run our tests on our all possible combinations of

`population_sizes = [20,50,100,200]`, `mutation_probabilities = [0.1, 0.5, 0.9]` and `crossover_probalities = [0.1, 0.5, 0.9]`, the rest of the variables remained the same as the initial parameter set, see table 6. We performed our tests on the datasets `rondrit016.tsp`, `rondrit050.tsp` and `rondrit127.tsp` as those are respectively the smallest, average and biggest datasets in amount of cities.

We ran each test 30 times as it can be seen in Figure 6 that after 30 runs the average fluctuates less than 0.1. In this figure the average fitness of 100 runs of one GA have been calculated after which the average was taken over 5 samples. This allowed for the shades in the graph which display the deviation of the average stabilized after 30 runs as well.

The test results can be found in the appendix as tables 3, 4 and 5. It is clear that increasing the population size increases the best fitness of the GA. However the best mutation and crossover probability vary a bit on the used dataset. For `rondrit016.tsp` the best results are yielded when having a mutation rate of 0.5 and a crossover probability of 0.1. For `rondrit050.tsp` this is also valid however when the population size is 20 the combination of a mutation rate of 0.9 and a crossover probability of 0.1 has a slight edge. When using the `rondrit127.tsp` dataset the best results are yielded when using a mutation rate of 0.9 and a crossover probability of 0.1.

2 Stopping criterion

The stopping criterion we implemented is one in which we look whether the fitness of the best individual keeps improving after a certain amount of generations. We decided that after 50 generations in which no improvement to the best fitness can be found, the algorithm should be stopped. We choose this value as you can see in Figure 9 that this stopping criterion would stop our algorithm at around generation 300 after which only very small improvements are being made to the best individual. To test the performance we used a test set with the initial parameter set however allowing up to 1000 generations, see table 7. We compared the speed of the algorithm with our stopping criterion implemented to the speed of the algorithm without stopping criterion. We ran both algorithms 30 times to measure the time. The algorithm with stopping criterion finished executing after `9.2969s` while the algorithm without stopping criterion only finished after `25.7344s`. So we see that the stopping criterion is clearly speeding up the algorithm, in this case by a factor of 3 approximately. In a second test however we used the initial parameter set again but this time only allow up to 100 generations. We also ran each algorithm 30 times in this case. This time the algorithm with stopping criterion stopped after `4.5156s` which is ever so slightly slower than the algorithm without stopping criterion `4.2656s`. This is approximately the same as the algorithm with stopping criterion didn't have an effect on the execution of the algorithm as there were not enough generations with an equal fitness for the stopping criterion to be used. The reason the times still differ a little is because genetic algorithms are stochastic models.

3 Other representation and appropriate operators (main task)

Alternative representation

For the alternative representation, the *path representation* was chosen, which naturally shows the order in which the cities are visited as a list. Note that when the representation changes, the crossover and mutation operators change as well. Out of the possible crossover operators available for the path representation, *partially mapped crossover* (PMX) was chosen. This operator is relatively simple to implement, as it works as follows:

- Take a random substring between a and b, and swap those sections. This also creates a mapping that will be used in resolving conflicts later on (if for example x is swapped from the first parent with y from the second parent, the mapping (x, y) is saved).
- Simply keep all cities before a and after b as is. If any conflict were to occur (for example, x is already present in the new child between a and b since it was copied from the parent), use the corresponding mapping to replace it with y.

PMX is easy to implement and requires less computations than for example cycle crossover. For mutation, *simple inversion mutation* is used that inverts the subset of cities between two random positions a and b.

Parameter tuning

For parameter tuning, a non-iterative approach was used on the parameters `population size`, `mutation probability` and `crossover probability`. The first one could take values over $[20, 50, 100, 200]$, and the last two probabilites range over $[0.05, 0.1, 0.3, 0.5, 0.7, 0.9, 0.95]$. Per combination of parameter values, the algorithm was run 10 times to measure the following:

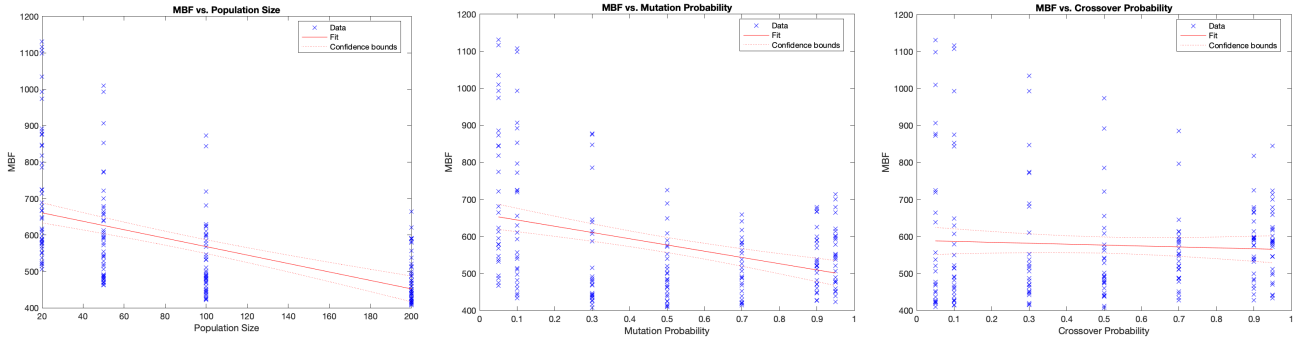
- The *mean best fitness* (MBF) averaged over the ten runs, together with its variance.
- The *average number of evaluations to a solution* (AES), together with its variance.

This results in four metrics (AES, MBF and their variances) for 245 combinations of parameters resulting from 2450 runs on the `rondrit016.txt` dataset, to see their effect on algorithm performance, robustness and number of iterations. A third common metric would be success rate, or how well/if the algorithmic instance approximates the real optimal solution. However, the fitness function is linearly proportional to the real distance, and the success rate (a constant, the optimal distance, over the real distance) is inversely and linearly proportional to the distance. Therefore, the success rate is inversely and linearly proportional to the MBF, and any conclusions made about the MBF apply to the algorithm's success rate as well.

Two factor interactions are neglected to see what the individual influences of each parameter are on the MBF and AES. For each parameter, a simple linear model is included in the figure as well. In general, one can see in figure 2 that increasing the mutation probability and population size generally procudes better quality results. For example, setting the population size to 20 gives varying results, with the worst case being almost twice as bad as the worst case results for a population size of 200. However, these factors also increase the number of evaluations, and thus the running time. So when choosing the right combination of parameter settings, a balance has to be found between the solution quality and running time. Since the running times are fairly low in these examples, it is not a necessity to keep them as low as possible. For the population size, a mean value of around a 100 accomodates the tradeoff between running time and solution quality well. For the mutation probability, a strong increase in number of iterations can be seen in figure 2. Therefore, setting a lower value of around 0.1 should work faily well. One can see that the influence of the crossover probability on the solution quality and number of runs is low. Thus setting it to an arbitrary median value of 0.5 should increase performance without influencing the solution quality. The settings thus are the following: `population size:100`, `mutation probability:0.1` and `crossover probability:0.5`, giving the following results:

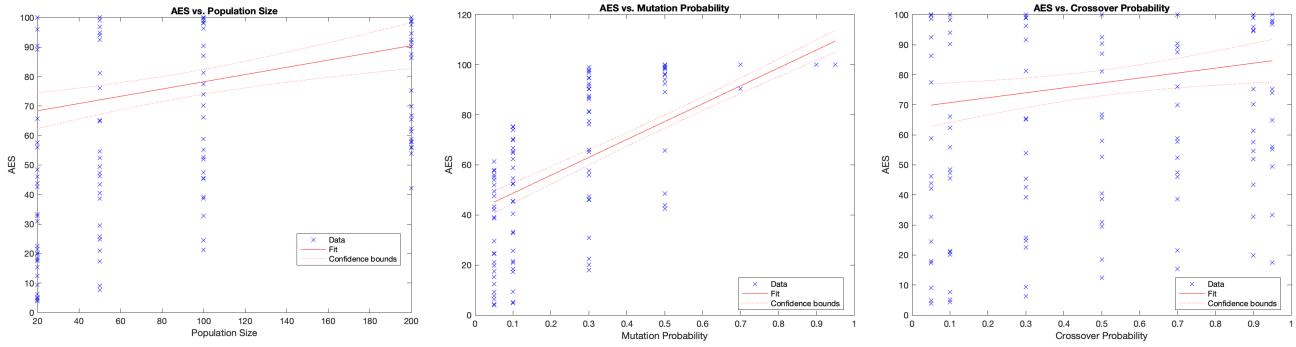
- A MBF of 504.03, scoring in the 40% best, with $\sigma_{MBF} = 28.51$.
- An AES of 57.10 iterations, scoring in the 25 % best, with $\sigma_{AES} = 9.74$.

which gives a good tradeoff between a fast algorithm that delivers fairly good results. And most importantly, the varainces are relatively low, with other settings usually resulting in a higher variance and thus giving worse results in some runs despite having a low MBF.



Population Size Mutation Probability Crossover Probability

Figure 1: Influence of parameters on the mean best fitness (MBF)



Population Size Mutation Probability Crossover Probability

Figure 2: Influence of parameters on the average number of evaluations (AES)

Performance and comparisons

Here, the path and adjacency representation are compared against each other on a slightly larger, new dataset that has not been used before for parameter tuning, namely `rondrit067.txt`. The algorithm instance is set to the same parameter settings to test both representations, as can be seen in table 1, except for the crossover method used (alternating edges and partially mapped crossover for adjacency and path representation respectively).

Figure 3 shows that both representations give similar results. However, it is important to note that number of iterations is not a good indicator to compare absolute running times, since more computations per iteration in one representation may result in a longer running time, but the same number of iterations. Therefore, average running times are also considered as a measure to benchmark both representations. For path representation, an average running time of $1.32s$, whereas for adjacency representation, the result was around $2.74s$. This can be explained, among other things, by the fact that the default implementation of the adjacency representation converts the chromosomes to a path representation every time for example the inversion mutation is called, and then converts it back. In general, it can be concluded that path representation works well in most cases compared to adjacency representation.

4 Local optimisation

We opted to use the 2-opt optimisation heuristic as this allows to explore individuals which would not be reached using solely partially mapped crossover. The performance criteria we used here is the best solution at the end of the run of the GA. As parameters we used the same parameter set and we performed 30 test runs. In figure 9 it can be seen that this local search heuristic has an average best fitness of `2.965` at the end of each run. This is a better performance than the algorithm without local search heuristic has. It is also clear that the algorithm with 2-opt converges very quickly after about 30 generations the GA has pretty much reached an optimal already.

Parameter	Setting
NIND	100
MAXGEN	100
NVAR	26
PRECI	1
ELITIST	0.05
GGAP	1-ELITIST
STOP PERCENTAGE	.95
PR CROSS	.50
PR MUT	.10
LOCALLOOP	0
CROSSOVER	alt. edges (adj.) / PMX (path)

Table 1: Parameter settings to compare path vs. adjacency representation.

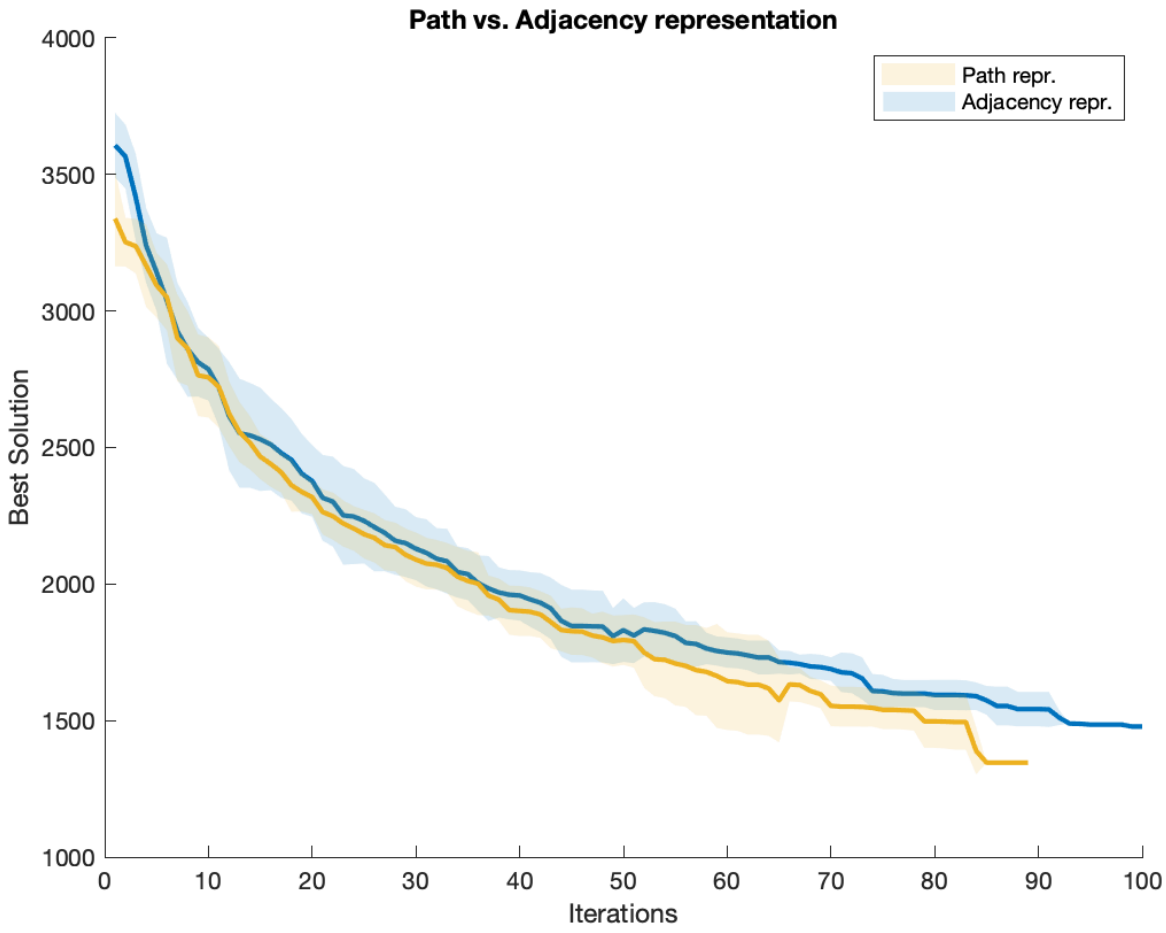


Figure 3: Path vs. adjacency representation

Dataset	Average shortest path	Known shortest path
XQF131	37857	564
BCL380	22650	1621
XQL662	47385	2513
RBX711	57958	3115
BELGIUM	1579.8	-

Table 2: Results of our GA on benchmark problems

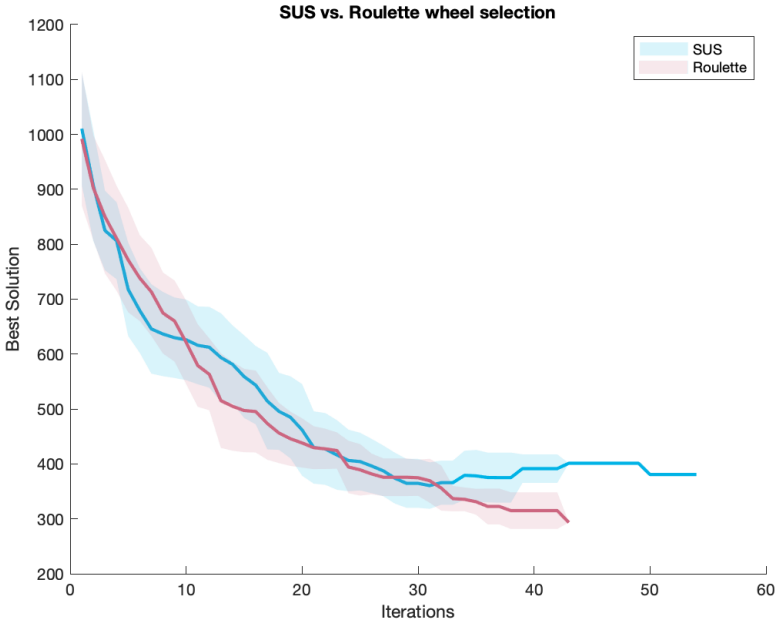


Figure 4: SUS compared to roulette wheel selection

5 Benchmark problems

For this we used the parameter settings as in table 7. Unfortunately we could not use our two-opt implementation for these benchmark problems as it would take too long. To evaluate we ran the GA 30 times on each benchmark and took the average of the best fitness at the end of each run. The results are found in table 2. We see that these results are quite bad compared to the optimal results, most of them are a few times longer than the known optimum. However we do beleive that if we ran it with two-opt as well the results would greatly improve.

6 Other task(s)

Two alternative parent selection methods were implemented, namely roulette wheel selection and tournament selection. The former is fairly simplistic and has little to no control over any sort of parameters, whereas the latter has two parameters that influence the selection pressure: tournament size k and whether or not replacement after selection is used. In both cases, fitness based selection is used instead of rank based selection.

Roulette wheel selection

The roulette wheel algorithm is one of the simplest parent selection implementations. However, it offers little to no control over any parameters. Figure 4 shows a comparison between the *stochastic universal sampling* (sus) selection method (implemented by default) and the roulette wheel selection method. Both samples are averaged over 10 runs, and are plotted together with their variances. One can see that overall, both methods perform similarly, with their average amount of runs lying between 40 and 60. However, due to the fact that the user has no control over any parameters of both selection methods, a third implementation, namely *tournament selection*, is preferred.

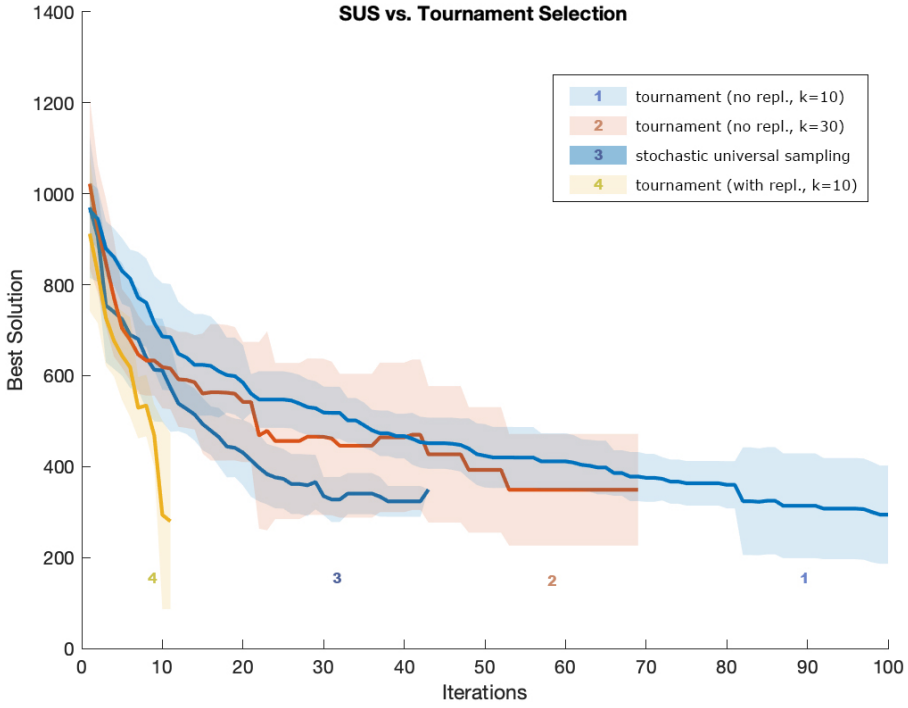


Figure 5: SUS compared to different settings of tournament selection

Tournament selection

For tournament selection, a deterministic variant is chosen where the individual with the highest fitness is always selected. With the selection of the fittest member being deterministic, only two other parameters remain: the tournament size k and whether or not the individuals are being replaced after selection. Figure 5 shows the influence of changing the tournament size, together with changing the replacement, plotted against the default SUS for comparison. The figure clearly shows that increasing k increases the selection pressure, which generally leads to finding local optima faster. However, the variance (width around the means in every plot) also increases compared to sus. This can be attributed to the higher selection pressure, that favors exploitation over exploration. Making replacement possible after an individual is selected (but keeping k at 10) shows that this increased selection pressure leads to even more drastic results. While in general, a solution is found in the least amount of iterations, the quality of the solutions varies. This can be seen in the fourth line having the highest variance around its average solution quality. In conclusion, increasing the tounament size can be favored over using replacement to solve these kinds of problems.

Time spent on the project

1. For each student of the team: estimate how many hours spent on the project (NOT including studying textbook and other reading material).
   (a) Maxwell: 48h.
   (b) Felix: 45h.
2. Briefly discuss how the work was distributed among the team members.
   - Different sections were distributed amongst the two team members, with Maxwell working on section 3 (representation) and 6 (other task), and Felix on section 1 (existing GA), 2 (stopping criterion), 4 (local optimisation) and 5 (benchmark problems). During the project, sufficient information was exchanged (for example, on how to conduct tests and generate informational plots) to help each other.

A Appendix

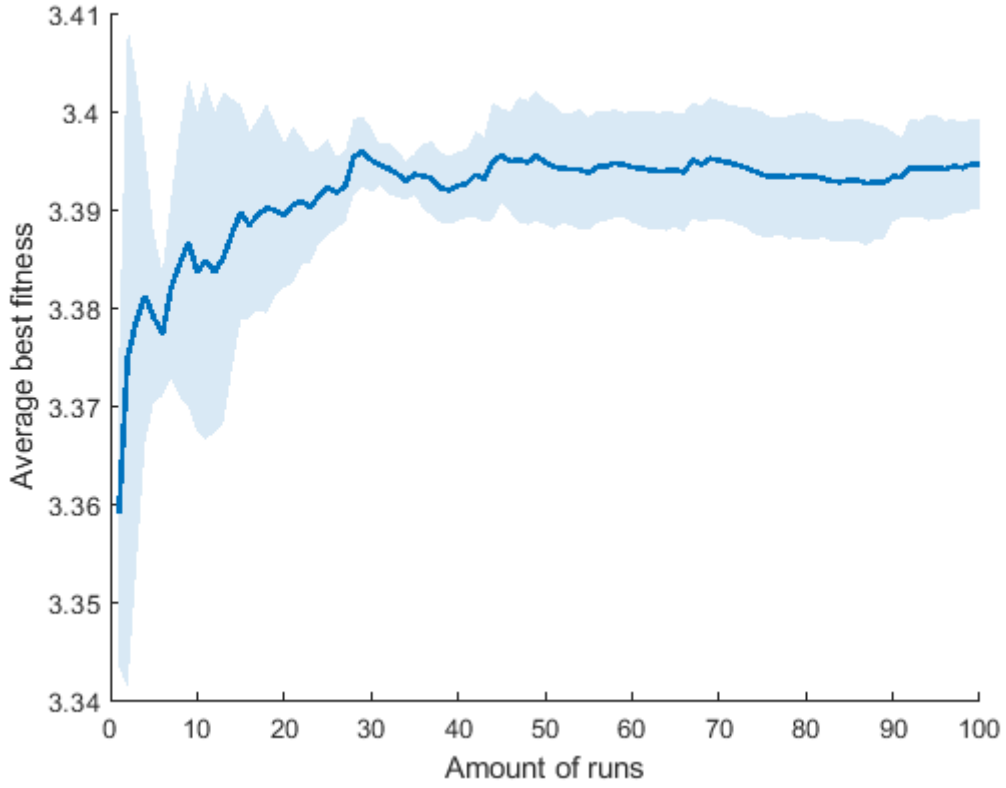


Figure 6: Average fitness after each run of genetic algorithm with preset parameters for rondrit016.tsp.

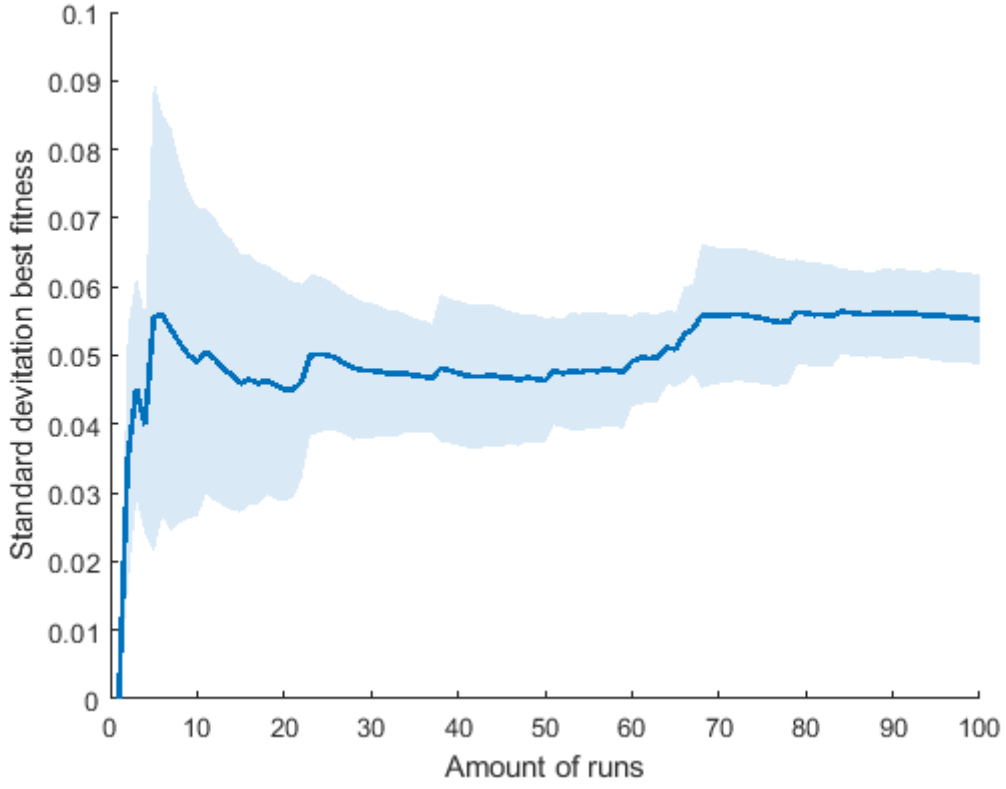


Figure 7: Standard deviation of the fitness after each run of genetic algorithm with preset parameters for rondrit016.tsp.

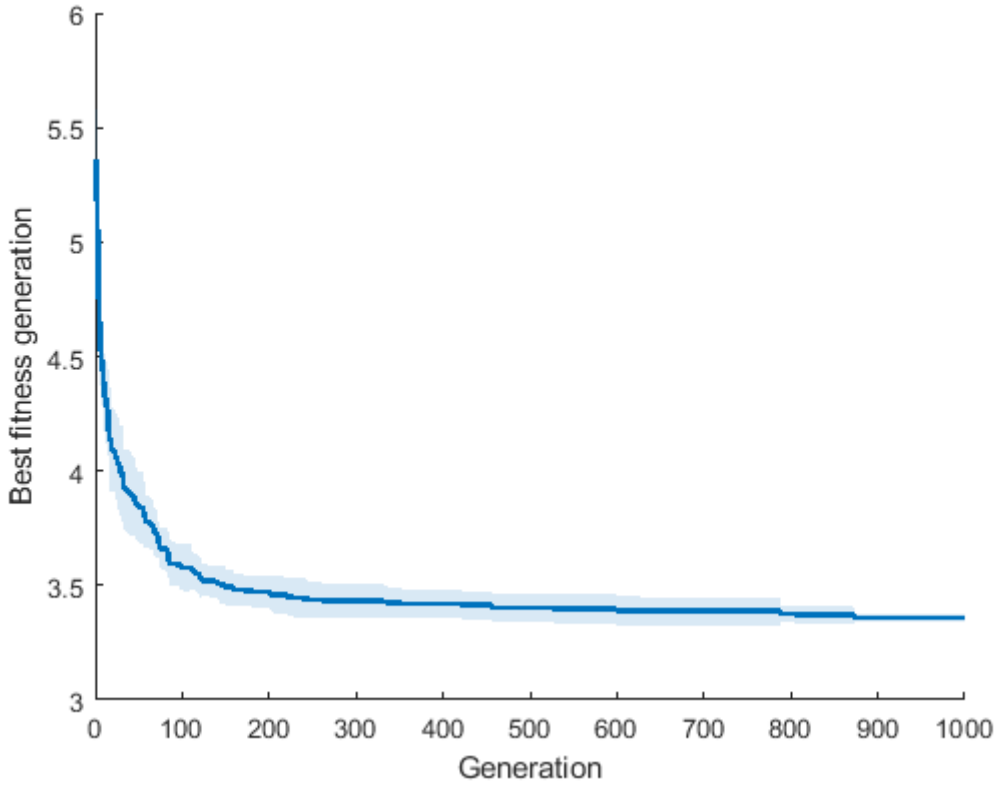


Figure 8: Best fitness after each generation of rondrit016.tsp.

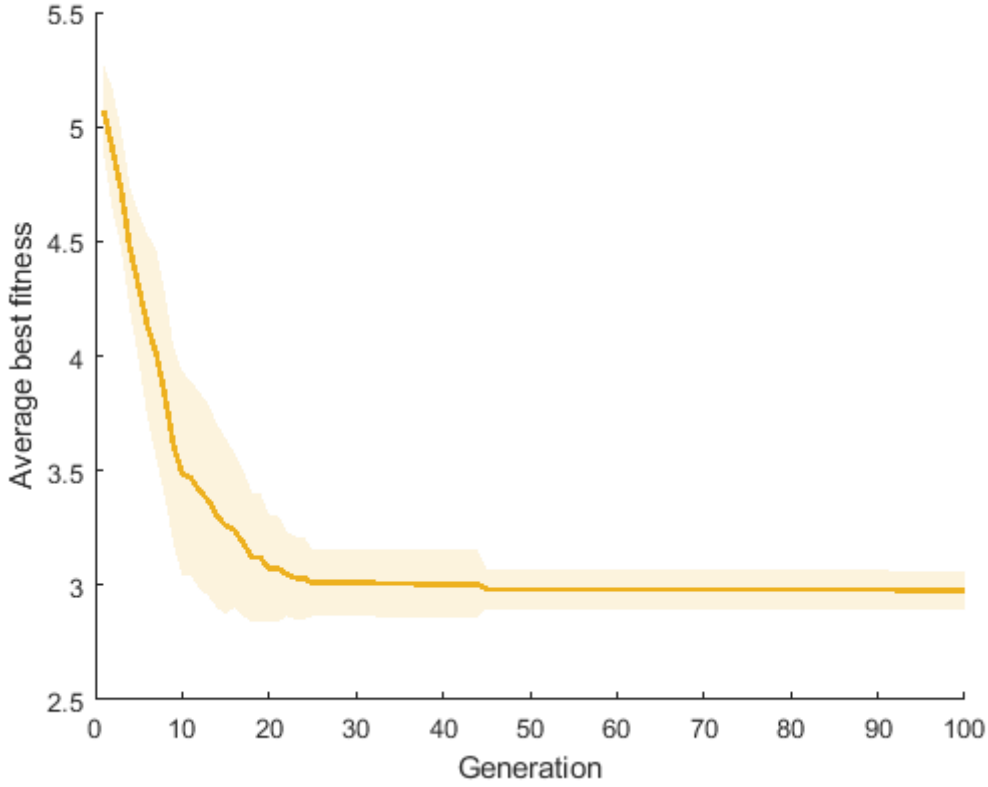


Figure 9: Average best fitness after each generation of rondrit016.tsp using 2-opt.

Dataset	Population size	Mutation probability	Crossover probability	Avg best fitness
rondrit016.tsp	20	0.1	0.1	4.4697
rondrit016.tsp	20	0.1	0.5	3.9475
rondrit016.tsp	20	0.1	0.9	3.6674
rondrit016.tsp	20	0.5	0.1	3.4387
rondrit016.tsp	20	0.5	0.5	3.5202
rondrit016.tsp	20	0.5	0.9	3.7271
rondrit016.tsp	20	0.9	0.1	3.5529
rondrit016.tsp	20	0.9	0.5	3.6324
rondrit016.tsp	20	0.9	0.9	3.8755
rondrit016.tsp	50	0.1	0.1	3.629
rondrit016.tsp	50	0.1	0.5	3.4585
rondrit016.tsp	50	0.1	0.9	3.5817
rondrit016.tsp	50	0.5	0.1	3.4102
rondrit016.tsp	50	0.5	0.5	3.4301
rondrit016.tsp	50	0.5	0.9	3.6851
rondrit016.tsp	50	0.9	0.1	3.4318
rondrit016.tsp	50	0.9	0.5	3.499
rondrit016.tsp	50	0.9	0.9	3.7419
rondrit016.tsp	100	0.1	0.1	3.5082
rondrit016.tsp	100	0.1	0.5	3.4189
rondrit016.tsp	100	0.1	0.9	3.4105
rondrit016.tsp	100	0.5	0.1	3.3899
rondrit016.tsp	100	0.5	0.5	3.404
rondrit016.tsp	100	0.5	0.9	3.5387
rondrit016.tsp	100	0.9	0.1	3.3987
rondrit016.tsp	100	0.9	0.5	3.4152
rondrit016.tsp	100	0.9	0.9	3.5908
rondrit016.tsp	200	0.1	0.1	3.4174
rondrit016.tsp	200	0.1	0.5	3.3893
rondrit016.tsp	200	0.1	0.9	3.3929
rondrit016.tsp	200	0.5	0.1	3.3674
rondrit016.tsp	200	0.5	0.5	3.3875
rondrit016.tsp	200	0.5	0.9	3.4459
rondrit016.tsp	200	0.9	0.1	3.3688
rondrit016.tsp	200	0.9	0.5	3.4111
rondrit016.tsp	200	0.9	0.9	3.5109

Table 3: Results of experiments on existing genetic algorithm using rondrit016.tsp.

Dataset	Population size	Mutation probability	Crossover probability	Avg best fitness
rondrit050.tsp	20	0.1	0.1	17.4596
rondrit050.tsp	20	0.1	0.5	13.8954
rondrit050.tsp	20	0.1	0.9	15.8289
rondrit050.tsp	20	0.5	0.1	12.6691
rondrit050.tsp	20	0.5	0.5	13.4892
rondrit050.tsp	20	0.5	0.9	16.3634
rondrit050.tsp	20	0.9	0.1	12.5782
rondrit050.tsp	20	0.9	0.5	13.7061
rondrit050.tsp	20	0.9	0.9	16.266
rondrit050.tsp	50	0.1	0.1	12.9594
rondrit050.tsp	50	0.1	0.5	11.9664
rondrit050.tsp	50	0.1	0.9	15.9328
rondrit050.tsp	50	0.5	0.1	11.1829
rondrit050.tsp	50	0.5	0.5	12.6892
rondrit050.tsp	50	0.5	0.9	15.8262
rondrit050.tsp	50	0.9	0.1	11.4006
rondrit050.tsp	50	0.9	0.5	12.9356
rondrit050.tsp	50	0.9	0.9	15.5243
rondrit050.tsp	100	0.1	0.1	11.6058
rondrit050.tsp	100	0.1	0.5	11.0588
rondrit050.tsp	100	0.1	0.9	14.6467
rondrit050.tsp	100	0.5	0.1	10.3616
rondrit050.tsp	100	0.5	0.5	11.6652
rondrit050.tsp	100	0.5	0.9	14.541
rondrit050.tsp	100	0.9	0.1	10.468
rondrit050.tsp	100	0.9	0.5	11.7728
rondrit050.tsp	100	0.9	0.9	14.7995
rondrit050.tsp	200	0.1	0.1	10.5701
rondrit050.tsp	200	0.1	0.5	10.2619
rondrit050.tsp	200	0.1	0.9	14.0714
rondrit050.tsp	200	0.5	0.1	9.8048
rondrit050.tsp	200	0.5	0.5	10.9036
rondrit050.tsp	200	0.5	0.9	14.0614
rondrit050.tsp	200	0.9	0.1	10.08
rondrit050.tsp	200	0.9	0.5	11.3638
rondrit050.tsp	200	0.9	0.9	14.4532

Table 4: Results of experiments on existing genetic algorithm using rondrit050.tsp.

Dataset	Population size	Mutation probability	Crossover probability	Avg best fitness
rondrit127.tsp	20	0.1	0.1	24.8371
rondrit127.tsp	20	0.1	0.5	21.7449
rondrit127.tsp	20	0.1	0.9	23.8091
rondrit127.tsp	20	0.5	0.1	21.3554
rondrit127.tsp	20	0.5	0.5	21.2436
rondrit127.tsp	20	0.5	0.9	23.6466
rondrit127.tsp	20	0.9	0.1	20.5545
rondrit127.tsp	20	0.9	0.5	21.2561
rondrit127.tsp	20	0.9	0.9	23.7391
rondrit127.tsp	50	0.1	0.1	21.6688
rondrit127.tsp	50	0.1	0.5	20.3521
rondrit127.tsp	50	0.1	0.9	23.8594
rondrit127.tsp	50	0.5	0.1	19.7283
rondrit127.tsp	50	0.5	0.5	20.4358
rondrit127.tsp	50	0.5	0.9	23.5532
rondrit127.tsp	50	0.9	0.1	19.3507
rondrit127.tsp	50	0.9	0.5	20.4588
rondrit127.tsp	50	0.9	0.9	23.3851
rondrit127.tsp	100	0.1	0.1	20.407
rondrit127.tsp	100	0.1	0.5	19.4711
rondrit127.tsp	100	0.1	0.9	22.5058
rondrit127.tsp	100	0.5	0.1	18.7146
rondrit127.tsp	100	0.5	0.5	19.3056
rondrit127.tsp	100	0.5	0.9	22.5534
rondrit127.tsp	100	0.9	0.1	18.5864
rondrit127.tsp	100	0.9	0.5	19.7016
rondrit127.tsp	100	0.9	0.9	22.4282
rondrit127.tsp	200	0.1	0.1	19.4346
rondrit127.tsp	200	0.1	0.5	18.7499
rondrit127.tsp	200	0.1	0.9	22.1479
rondrit127.tsp	200	0.5	0.1	18.0862
rondrit127.tsp	200	0.5	0.5	18.7996
rondrit127.tsp	200	0.5	0.9	22
rondrit127.tsp	200	0.9	0.1	17.8947
rondrit127.tsp	200	0.9	0.5	19.0199
rondrit127.tsp	200	0.9	0.9	21.8703

Table 5: Results of experiments on existing genetic algorithm using rondrit127.tsp.

Parameter	Setting
NIND	50
MAXGEN	100
NVAR	16
PRECI	1
ELITIST	0.05
GGAP	1-ELITIST
STOP PERCENTAGE	.95
PR CROSS	.95
PR MUT	.05
LOCALLOOP	0
CROSSOVER	xalt edges

Table 6: Initial parameter set.

Parameter	Setting
NIND	50
MAXGEN	1000
NVAR	16
PRECI	1
ELITIST	0.05
GGAP	1-ELITIST
STOP PERCENTAGE	.95
PR CROSS	.95
PR MUT	.05
LOCALLOOP	0
CROSSOVER	xalt edges

Table 7: Parameter settings to check the performance of the stopping criterion.