Project Genetic Algorithms

Robbert DeHaven
Brent Sprangers

December 2018

1 Parameter evaluation

- **Individuals and Generations:** These parameters directly impact the processing power needed in terms of individuals created over the lifetime of the algorithm. Higher values result in better results, but at diminishing returns. High values for individuals and low values for the number of generations (eg. 90/30) seem to do better than the other way around (30/90) while having the same computational budget. A higher amount of individuals allows for a faster improvement over fewer generations.
- **Mutation % and Crossover %:** The GA has rather poor results when no mutation occurs. In comparison, a mutation only GA scores around 15% better on the 50 city tour with 90 individuals over 175 generations. The crossover operator seems to make lager step improvements that occur less frequently, while the mutation operator makes smaller more consistent improvements
- **Elite %:** Reducing the elitism percentage to zero allows the best distance to increase during a run. High values for elitism slow down progress as there are less new individuals introduced into the pool. High values also encourage loss of population diversity and premature convergence.
- **Loop detection:** Is a local optimization heuristic that greatly improves results especially in the first stages of the GA. Using loop detection resulted in about a 50% reduction in tour length using the 48 city tour.

2 Stopping Criterion

A stopping criterion can be added to prevent the algorithm from making useless calculations. This occurs when the improvement of the result in relation to the required extra computation time drops below a certain threshold. Stopping criteria can be fitness based or diversity based. A diversity based criterion is already implemented in the template, it is set up to prevent premature convergence. In parallel to this criterion, we implemented a fitness based stopping criterion based loosely on the examples given in [3]. It takes as input: the current generation, the best fitness values of all generations, X and REQ_IMPR. Every generation the criterion will check if a required improvement REQ_IMPR has occurred over the past X generations. These parameters can easily be adapted to fit a specific problem better: time-sensitive applications can have a lower X value and/or a higher REQ_IMPR, applications that require optimal results can have higher X values and/or lower REQ_IMPR. The table below gives the results for some example values of X and REQ_IMPR.

X	REQ_IMPR	start fitness	stop fitness	end fitness	stop generation
5	0.01	17.06	10.86	13.11	19.6
10	0.01	17.12	10.79	12.22	39.9
20	0.01	17.05	10.86	11.7698	74.6
20	0.05	17.05	10.83	11.95	61.4

Table 1: Results for some example values of X and REQ_IMPR.

The results are averages of 100 runs with the following parameters: alternating crossover, inversion mutation, Pcr = 0.95, Pmu=0.1, elitism = 0.05 and population = 50 on the test dataset with 25 cities. We later found that after optimizing the GA different values could be used as the fitness value improvements per generation was better.

Besides the diversity-based criterion using fitness values that was already implemented, an additional diversity-based criterion was experimented with. This criterion was implemented for the adjacency representation and counted the amount of unique partial routes. Here partial routes were limited to routes between 2 cities but can be expanded to linked routes between more cities. There are 2 factors that influence the number of unique city pairs: dataset size and population size. A dataset containing n cities has $\sum_{x=1}^{n} x$ unique city pairs. As the dataset increases, the amount of unique pairs increases. The population also plays a role, large populations have higher odds of containing more unique pairs. This implies that small datasets with large populations will have almost no benefit from this criterion as the number of unique cities found will remain around the maximum during the entire run of the GA. Furthermore, for very large datasets with large populations, this criterion can be computationally expensive. For these reasons the fitness-based criterion proposed earlier is preferred. The same principle of unique city pairs could also have been used as an adaptive parameter to control the mutation rate of the GA. Whenever the number of unique pairs drops below a certain percentage or hard limit the mutation rate can be increased to prevent premature convergence. Figure 1 shows an example run. This dataset has 4950 unique pairs. It drops sharply as the worst pairs are naturally removed from the population, at around 2000 there seems to be a balance between new pairs being introduced through mutation and crossover and removal of bad pairs.

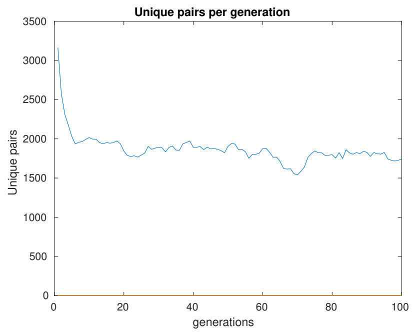

Figure 1: Evolution of unique city pairs. Dataset size=100, individuals=50

3 Path Representation

We implemented the path representation as a different representation. Everything that follows is implemented with respect to this representation. Of course one can choose to change to adjacency representation using *adj2path.m* and *path2adj.m* which were already present in the template, but we did not test this.

3.1 Mutation

Apart from the inversion operator which was already present in the template we implemented the following mutation operators:

- **Insert mutation** The insert operator was implemented as described in the book[3]. The operator randomly selects a city, removes it from the tour and inserts it back at a randomly selected place.
- **Scramble mutation** The scramble operator was implemented as described in the book. A subpath from the tour is selected and replaced by a permutation of this subpath.

To test the exploratory capability of the different mutation operators we ran the genetic algorithm for different mutation probabilities and a crossover probability of zero on the given dataset of 100 cities.
Results for the mean length of the best tour for three different mutation probabilities are plotted in figure 2.
There is no significant difference between the inversion and insert operator with a 95% confidence level but the scramble operator was found to be significantly worse. This is no surprise because the permutation throws away a lot of information from the chromosomes. There is even a chance that the whole chromosome is changed.
To get an idea of which operators take more computational time, mean times per run for 100 individuals and 50 generations with a 100% mutation probability and 0% crossover probability are given in table 2, this without stopping criterion. The scramble operator takes significantly more time than the inversion and insert operator which perform equally good with respect to time.

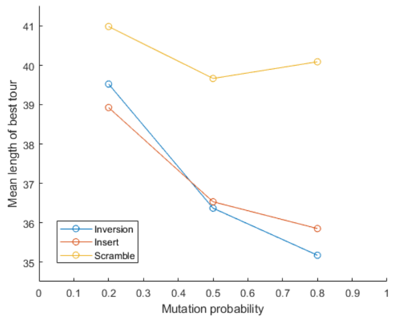

Figure 2: Comparison mutation operators

Operator	Inversion	Insert	Scramble
Time(s)	0.39	0.34	2.08

Table 2: Computation time for 100 individuals and 50 generations for different mutation operators.

3.2 Recombination

The implemented crossover operators are:

- **Partially mapped crossover** Implemented as described in the book.
- **Alternating edge crossover** This operator was already present in the template but we adjusted it to work with the path representation.
- **Order crossover** Implemented as described in the book. This operator tries to preserve the relative order as much as possible.
- **Sequential constructive crossover** This operator is implemented as introduced in [1].

To test the exploiting behavior of the different recombination operators we ran the genetic algorithm for different crossover probabilities and a mutation probability of zero on the given dataset of 100 cities.
Leaving scx out of consideration, no significant differences were found at low probabilities. For intermediate probabilities, the alternating edge and the order operator are significantly better than the pmx operator with a 95% confidence interval and for high probabilities, only the order operator was still significantly better.
The difference with sequential constructive crossover, however, is remarkable. For all probabilities, this operator gives solutions with a length that is almost a factor 2 better than the other operators. This is because, as pointed out by the authors who introduced scx, the operator preserves characteristics of the parents very good, a high amount of edges is preserved and the operator tries to preserve the better edges[1], in contrast to the other operators where edges are kept more at random.
To check how the crossover operators perform with respect to time we tested them the same way as the mutation operators. Results are given in table 3, where we can see that better performance of the sequential constructive crossover comes with a big computational cost. A single run takes over 20 seconds, whereas for the other operators a single run takes about 2 seconds.

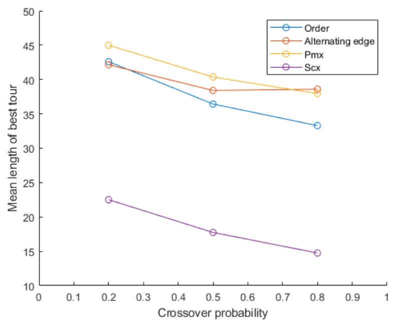

Figure 3: Comparison crossover operators

Operator	Order	PMX	Alternating Edge	SCX
Time(s)	2.74	1.24	1.33	13.18

Table 3: Computation time for 100 individuals and 50 generations for different crossover operators.

3.3 Parameter tuning

3.3.1 Method

To do some parameter tuning we use an algorithm loosely based on the algorithm described in [2], that can be described as a random walk up the utility hill. We start with an initial set of parameters that give a good result. Next, we choose one or two parameters at random and perform the genetic algorithm with the altered set of parameters. If the new set gives a better mean result than the initial set, we keep the new set and forget about the initial one. If this is not the case we forget about the new set and go on with the initial set. These steps are then repeated until we have made a chosen amount of steps in the parameter space or if a trial amount of steps, 10 in this case, did not result in a better parameter set.
As possible altered parameter sets, we allowed changes of ± 0.1 for the mutation probability, the crossover probability, and the elitist percentage. To limit computation time we fixed the ratio of the number of individuals and the maximum number of generations such that the amount of generated chromosomes is 5000. The mutation or crossover operator was not considered as a degree of freedom, but we tested this parameter tuning algorithm separately for the inversion, insert and scramble mutation operator and only considered the sequential constructive crossover operator.

3.3.2 Results

The evolution of the best tour length during the walk is shown in figure 4. The values of the parameters resulting in the best values are given in table 4. It's remarkable that apart from the number of individuals and generations the parameters all converged to the same values.
A more extensive run, with the previous results as initial values, with a mean taken over more runs and more possible trial steps, lead to the parameter values in table 5, for scx and inversion only because this method is rather time-consuming.

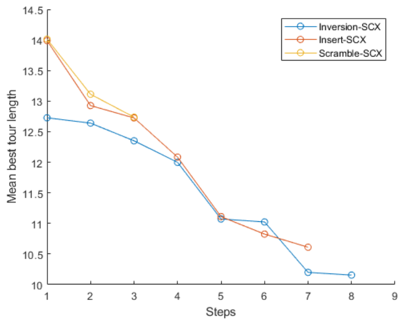

Figure 4: Evolution of tour length during parameter tuning parameter tuning

Mutation	Crossover	Individuals	Generations	Elitist	Mutation prob.	Crossover prob.	Mean best tour
Inversion	SCX	96	52	0.1	0.6	1.0	10.15
Insert	SCX	156	32	0.1	0.6	1.0	10.61
Scramble	SCX	50	100	0.1	0.6	1.0	12.73

Table 4: Best parameters for different operators.

3.4 Local Optimization

We implemented the 2-opt local optimization method and compared it with the implemented local loop detection.
In all cases, the solutions found using any local optimization method are significantly better with a 95% confidence level. Only for insert mutation local loop correction is significantly better than 2-opt.

4 Parent selection method

In this section, we implemented two other parent selection strategies alongside stochastic universal sampling (SUS), which was already implemented in the GA toolbox. We chose to implement fitness-based selection (FPS) and tournament selection (TS). For FPS we tested the standard version and a version with sigma scaling implemented with c=2 as described in [3]. For tournament selection, we tried out tournament sizes k = 2,4 and 10. The tournaments are deterministic and individuals are chosen without replacement. All tests were run on the parameters described in table 5. In figure 5 (Appendix A) the blue line indicates the mean value of mean fitness for that generation over 50 runs of the GA with this parameter set, the red line represents the mean value of the best fitnesses for that generation. We can see from table 7 that the differences between SUS, FPS, and TS with k=2 are very minor. More interesting are graphs with TS with tournament sizes 4 and 10. It is clear that as k increases the selection pressure increases. This is visible in the graphs as a

Mutation	Crossover	Individuals	Generations	Elitist	Mutation prob.	Crossover prob.	Mean best tour
Inversion	SCX	179	28	0	0.3	0.95	10.15

Table 5: Parameters after longer run taking parameters of the first row of table 4 as a starting point.

Mutation	Crossover	No opt.	Time(s)	2-opt	Time(s)	Local loop	Time(s)	Both	Time(s)
Inversion	SCX	10.33	16.25	9.40	19.07	9.16	15.36	9.13	18.05
Insert	SCX	11.09	16.17	9.56	20.93	9.33	16.60	9.36	19.67
Scramble	SCX	13.07	11.5	10.21	15.53	9.93	11.40	9.95	13.71

Table 6: Results of parameter tuning

steeper drop in the earlier generations and in later generations the diversity drops (lower standard deviation) and the GA stalls. The usefulness of TS is limited as the populations we are working with are rather small and the absolute fitness values can be found quickly.

Survivor selection method	Mean best tour
SUS	10.5
FPS	10.46
FPS sigma	10.43
TS k=2	10.73
TS k=4	11.56
TS k=10	14.03

Table 7: Results parent selection methods.

We chose to continue with FPS as a parent selection method. While FPS with sigma scaling performed slightly better, it was implemented as an afterthought after all benchmarks had been run.

5 Survivor selection

An extra survivor selection method based on round-robin tournaments was implemented and tested. The results are displayed in figures 7 and 8, and table 8. As expected, the uniform and fitness-based methods returned similar charts and values. This is due to the parameter set used, more specifically the parameter elitism=0. This results in both methods taking no individuals from the previous generation. The round-robin based method implemented by us performs tournaments on the combined pool of parents and children ($\mu + \lambda$). Even though there is no elitism in our parameter set, it allows strong individuals from the previous generation to survive. As the tournament size q increases children selection becomes less random due to a smaller chance of a lucky draw of opponents. We observed during run-time that as q increased, run-time increased noticeably. The shape of the graphs of the different q's are very similar and not much can be deduced from them (they were added for completeness). In the single examples, we can see that the standard deviation of RR with q=20 appears to be a bit smaller but this can be due to the stochastic nature of GA.

Survivor selection method	Mean best tour
uniform	10.3683
fitness based	10.4152
RR q=5	12.2688
RR q=10	11.9616
RR q=20	11.8905
RR q=50	11.8067

Table 8: Results survivor selection methods.

After some initial benchmarking, we concluded that having no elitism was leading to worse results. This was most likely caused by getting stuck in a local optimum during the random walk of the parameter tuning algorithm. With this new information, we decided to rerun the tests for uniform and fitness based survivor selection with elitism parameter set to 0.05. The results are presented in table 9. The changes are again very small and probably not statistically significant. We decided to use fitness-based survivor selection in the following tests.

Survivor selection method	Mean best tour
uniform	10.3802
fitness based	10.3665

Table 9: Results survivor selection methods with elitism=0.05.

6 Benchmarking

After some initial benchmarking, we determined that the parameter set found by using random walk returned better results when we reset the elitism parameter to 0.05. All results from the benchmarks are found using these new parameters coupled with fitness proportionate parent selection and fitness-based survivor selection. Benchmark tours used were XQF131, BCL380, and XQL662. Due to how time-consuming solving XQL662 is this benchmark is only run once, the other two are averaged over 5 runs. Next to the parameter values for individuals and the number of generations found during parameter tuning (179/29), we also benchmarked values 393/64. This represents a total of around 25000 evaluations instead of the original 5000 and keep the same ratio between individuals and generations.
Allowing ourselves some more freedom in the numbers of individuals/generations, Because the values were fine tuned for the limit of 5000, we allowed ourselves some more freedom in the numbers of individuals/generations, and also benchmarked with 250/500. This gives some more computational power and a higher probability of coming closer to the solution. These numbers are not optimal because an optimization with a total number of 125000 created individuals would take too much time.
The results are shown in table 10.

Benchmark	Individuals/Generations	Mean best tour	Optimal tour	Ratio
	179/29	629.45		1.116
XQF131	393/64	613.07	564	1.087
	250/500	579.47		1.026
	179/29	2110.94		1.302
BCL380	393/64	1902.84	1621	1.174
	250/500	1879.81		1.154
	179/29	3562.37		1.418
XQL662	393/64	3391.27	2513	1.349
	250/500	2884.2		1.147

Table 10: Benchmark results

The GA's performance obviously decreases as the dataset sizes increase. The XQF131 benefitted least from the fivefold increase in computing power. This is probably because it gets exponentially more difficult to improve the solution as the algorithm approaches the optimal solution. We cannot draw the same conclusion for the largest dataset as it was only run once and cannot be used as a generalization.

References

[1] Zakir H. Ahmed. Genetic algorithm for the traveling salesman proble m using sequential constructive crossover operator. *International Journal of Biometrics Bioinformatics*, 2010.

[2] Olfa Chebbi and Jouhaina Chaouachi. Effective parameter tuning for genetic algorithm to solve a real world transportation problem. *2015 20th International Conference on Methods and Models in Automation and Robotics (MMAR)*, 2015.

[3] A.E. Eiben J.E. Smith. *Introduction to Evolutionary Computing*. Springer, 2015.

A

Graph results of parent selection

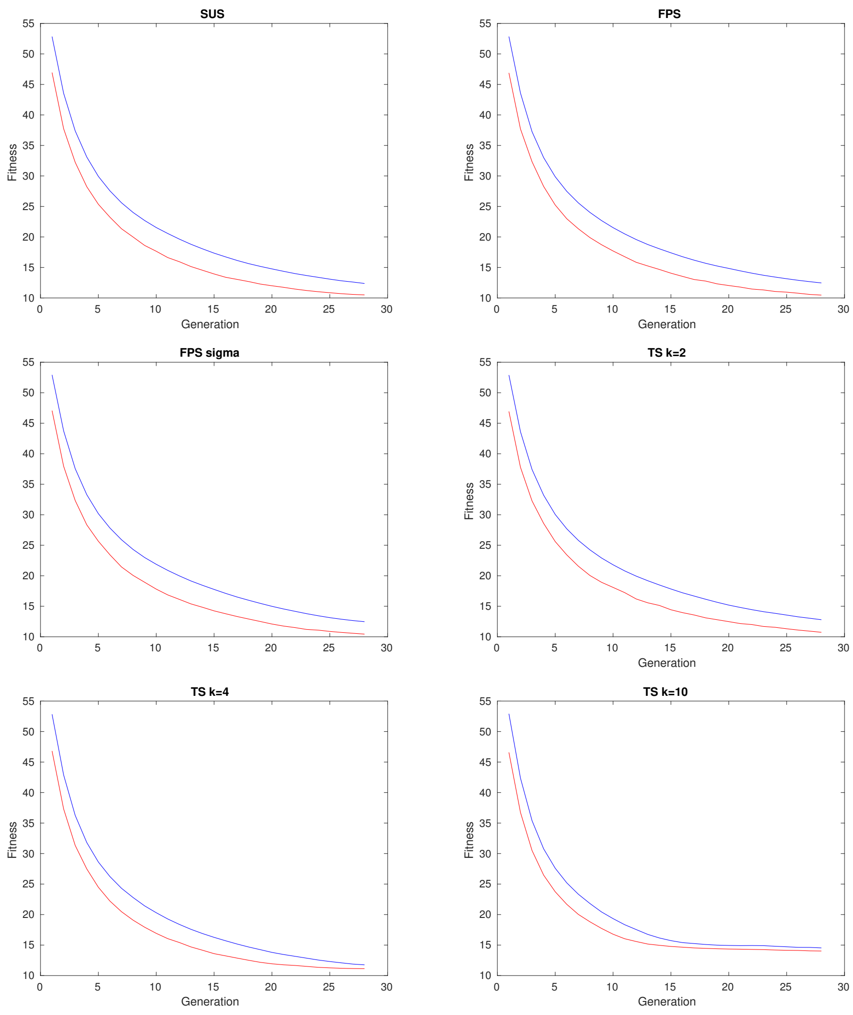

Figure 5: Mean values for different parent selection methods.

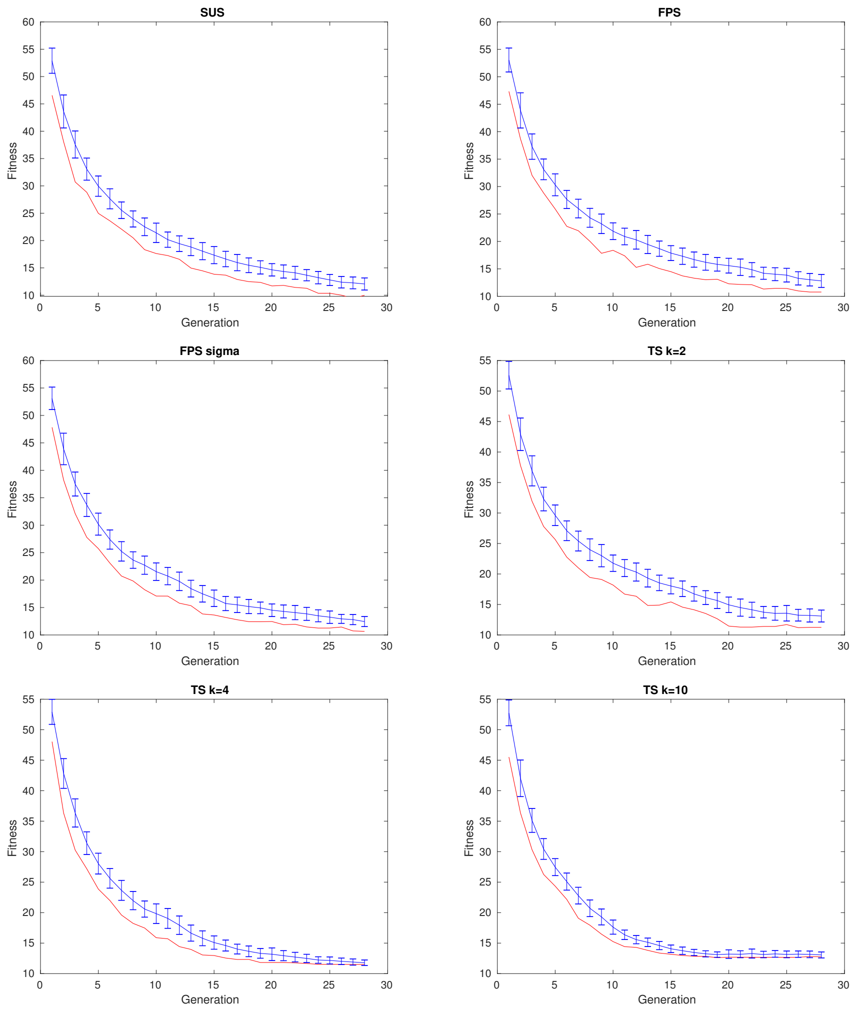

Figure 6: Example single runs with bars indicating standard deviation.

B

Graph results of survivor selection

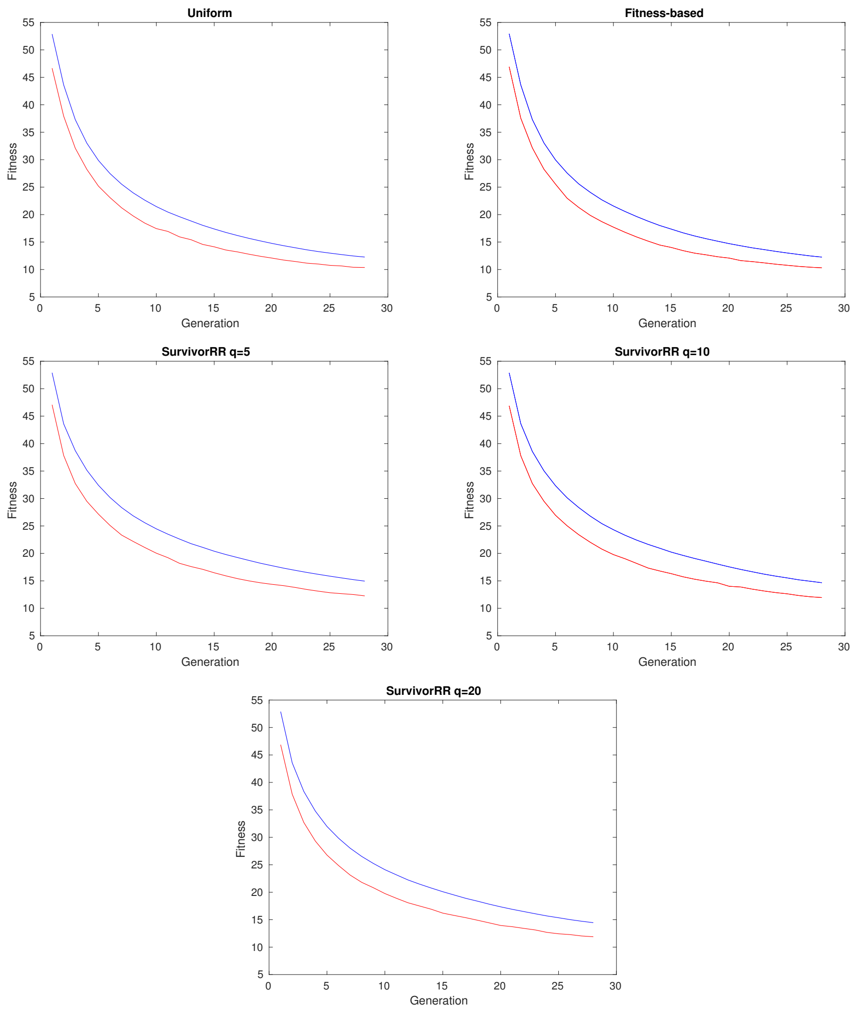

Figure 7: Mean values for different survivor selection methods.

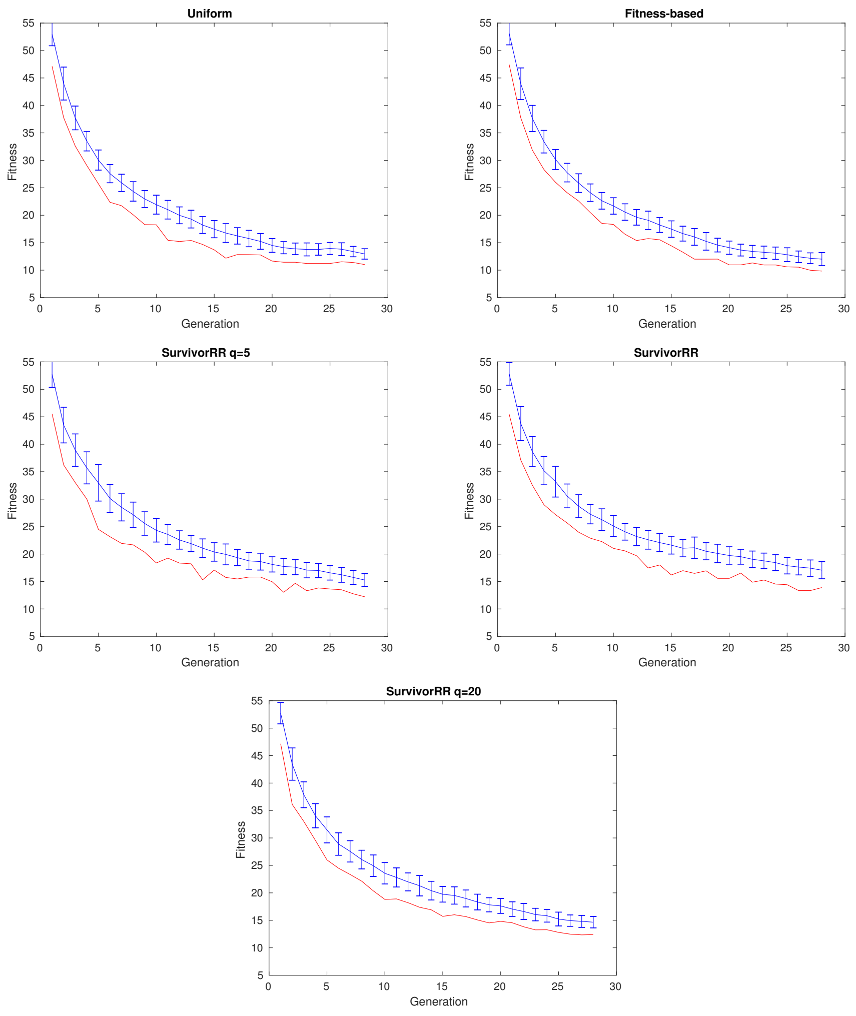

Figure 8: Example single runs for survivor selection methods with bars indicating standard deviation.