Genetic Algorithm

Raul Pârău

April 2025

Contents

1 Code

To write my own Genetic Algorithm, I had to implement the following methods: One method to compute the fitness level, one method to compute the fitness level of an individual, one method to apply mutation to an individuals, a method to identify two random individuals to cross over and lastly a method to cross these individuals. The exact implementation of all of these methods and a description of how they work can be found in my Github repository in the class "PracticalGA". In that class I also implemented my general approach for the algorithm. The other classes that can be found in my Github repository are for testing my algorithm by tweaking some parameters and writing the data into csv files.

2 Test Questions

2.1 How does population size affect the number of generations needed?

To answer this question I tested my algorithm with populations of size 20, 50, 100 and 200. Then I needed to settle for an appropriate tournament size every time, so I decided to make my tournament size 5 percent of the total population. I ran my code 20 times each to get the results and printed them to a csv file. Important to mention is, that for this test I used my first approach to a mutation method and a double point cross-over.
As shown in Figure 1, we can see that larger populations require fewer generations to reach the desired target String. The following table shows the exact average number of generations needed.

Population Size	**Generations Needed (Average)**
20	10.305
50	3.600
100	844
200	275

Table 1: Average number of generations needed depending on population size.

Figures 2 to 5 show the number of generations needed for each population size at each iteration. I noticed that larger population sizes (e.g., 200 individuals) resulted in greater variance in the number of generations required, likely due to increased genetic diversity and a higher chance of extreme outcomes influencing the results. To adjust for the influence of outliers I also computed the median of the generations needed. The values for the median can be found in the following table.

Population Size	**Generations Needed (Median)**
20	10.560
50	4.016
100	648
200	168

Table 2: Median number of generations needed depending on population size.

Looking at this data it seems as if the outliers have a bigger impact on the larger populations and that the outliers for those populations are generally higher numbers of generations needed. This makes sense, as it concludes that there were more “unlucky” runs where the algorithm got stuck on a local optimal solution.

2.2 How does mutation rate affect the number of generations needed?

2.2.1 First experiment

To answer this question I ran my genetic algorithm for different mutation rates. I settled for the following mutation rates: 0.05, 0.1 , 0.2 , 0.3 , 0.4 , 0.5 and 0.8. To get the data into a fitting format I used a logger that turned the data into a csv-file. Furthermore I had to change my genetic algorithm to take a parameter mutation rate, in order to get the data of all mutation rates into one csv file. For each mutation rate I ran my code 20 times and I used my first approach of the mutation method. This experiment can be found in the class ”TestForMutationRate”.
Figure 6 illustrates the average number of generations needed for each of these mutation rates. However the findings of this experiment surprised me a lot, as there was no trend to be seen in the data. It almost seems as if the mutation rate has no impact on the number of generations needed when I use my first approach of the mutation method. From these findings I concluded that the mutation method is not aggressive enough, so I implemented a second approach, where mutation is applied per gene (in this case per character) and not per individual (in this case per String).

2.2.2 Second experiment

The results of the second version of this experiment are shown in figure 7, where my expectations were matched with the result. The figure again displays the average number of generations needed per mutation rate, but as the mutation rate was more aggressive this time, I changed the values of the mutation rates tested. The new values of the mutation rates and the according median and average can be seen in table 3.

Mutation Rate	Median	Mean
0.001	303	405
0.01	35	46
0.02	20	21
0.05	11	13
0.1	11	12
0.2	14	14
0.3	103	114
0.4	45703	74324

Table 1: Median and Mean values for different Mutation Rates.

These results show exactly what I expected to happen. There seems to be a perfect spot for the mutation rate that keeps the generations needed the lowest. It seems as if this spot lies between 0.05 and 0.1. For mutation rates higher than 0.1 the number of generations needed gets higher with each step. For 0.4 the algorithm takes thousands of generations which is due to the fact that for every individual 40 percent of its chromosomes get mutated, which leads to too much randomization.

2.3 Leaving out the crossing operator

For me the code runs infinitely if I leave out the crossing operator, as then it just applies mutation which is too random to get to the target String "HELLO WORLD". With crossover there is the possibility to choose the "best" individuals to cross, so better offsprings are the result. With only mutation, it is possible that the best individuals get even worse due to the mutation. There is no progress made and leaving out the crossing operator violates the principle of a genetic algorithm.

2.4 Leaving out the mutation

The algorithm still works without mutation, but is more prone to getting stuck at one local optimal solution. Almost half of the times I ran the algorithm without any mutation it got stuck on a solution which was close, but not quite the target String. Without mutation there is a lack of genetic diversity, this can lead to the algorithm converging prematurely to a local optimal solution. After many iterations of the algorithm every individual will be the identical, effectively halting any further exploration of the solution space.

2.5 Finding the optimal values

In the previous testing of my algorithm I already discovered that the optimal mutation rate lies between 0.05 and 0.1 (If the second approach to the mutation method is used). Furthermore the test results showed that larger populations reached the target string quicker than smaller ones. One last parameter I need to check is the tournament size.
To do that I created a new class "FindPerfectValues" where I ran my algorithm with my second mutation method, a population size of 200 and a mutation rate of 0.05. Then I created an array of different population sizes, ran the algorithm 20 times for each population size and wrote the results for each iteration into a csv file, using a logger.
Figure 8 displays the average number of generations needed for each tournament size that I tested. You can see that the average number of generations needed is the lowest for a tournament size of 10. However the average number was pretty similar for most of the tournament sizes.
To get a better result I tested for the variance of generations needed by looking at the minimum and maximum of generations needed, the results can be found in the following table.

Tournament Size	**Average**	**Min**	**Max**	**Median**
2	17	14	27	17
5	9	7	14	9
10	7	6	9	7
20	9	5	22	8
30	8	5	16	8
40	8	5	22	7
50	10	5	28	9.5
75	8	5	16	7.5
100	10	6	26	9

Table 3: Results for different tournament sizes over 20 runs.

The results show that the tournament size of 10, not only provides the lowest average number of generations needed but also has a low variance as all values lie between 6 and 9. This shows that the algorithm does not get stuck on local optimal solutions and therefore runs the most efficient and quickest. Furthermore the algorithm is of course faster when using a tournament size of 10, than it would be for a tournament size of 40, as it does less iterations.

To conclude, the perfect value for mutation is 5 percent, and the larger the population the better. The optimal value for the tournament size is 5 percent of the population size.

List of Figures

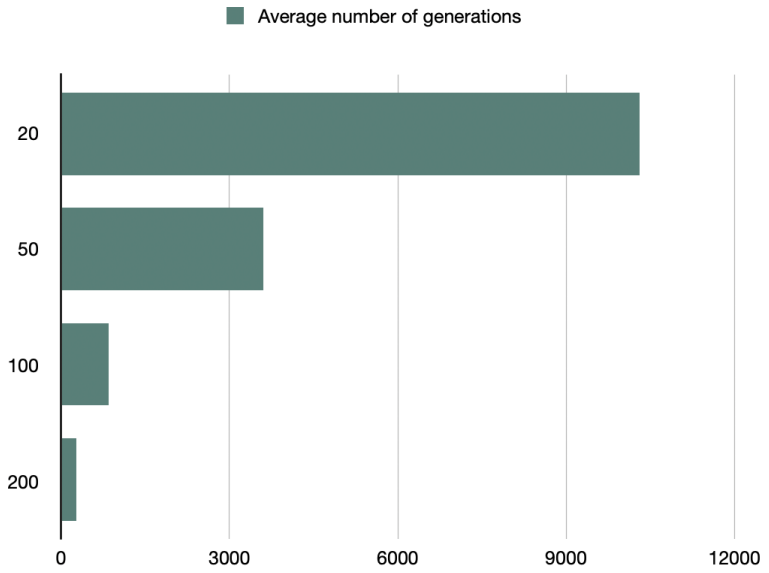

Figure 1: Average number of generations needed vs population size

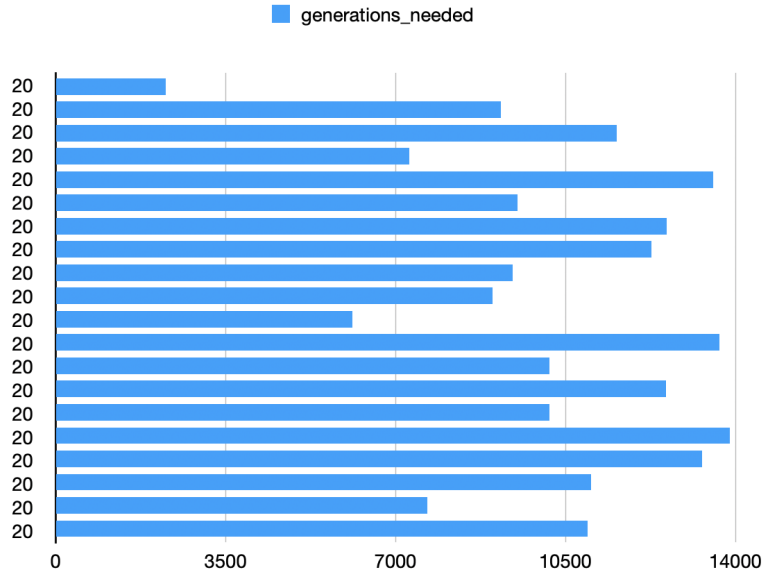

Figure 2: Number of generations needed vs population size size of 20

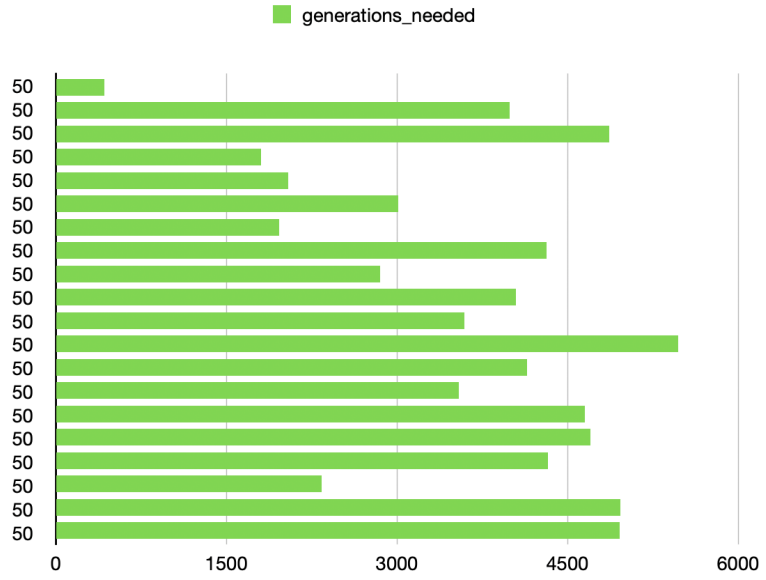

Figure 3: Number of generations needed vs population size size of 50

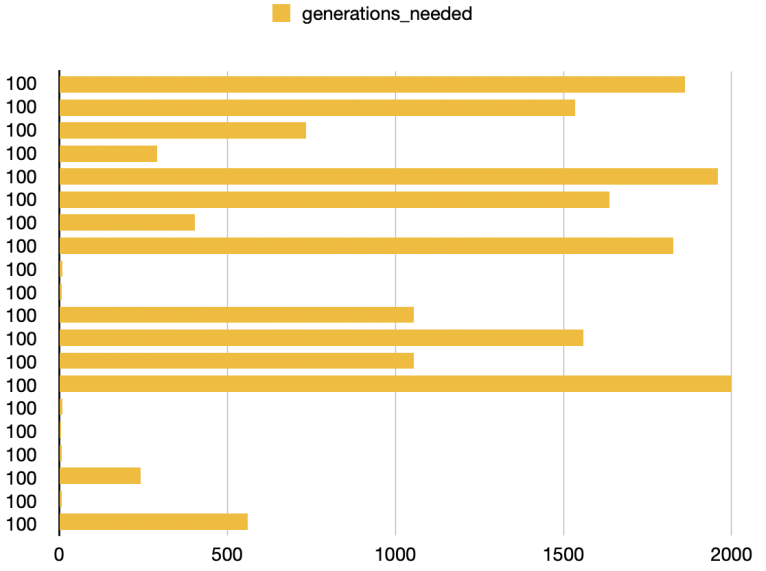

Figure 4: Number of generations needed vs population size size of 100

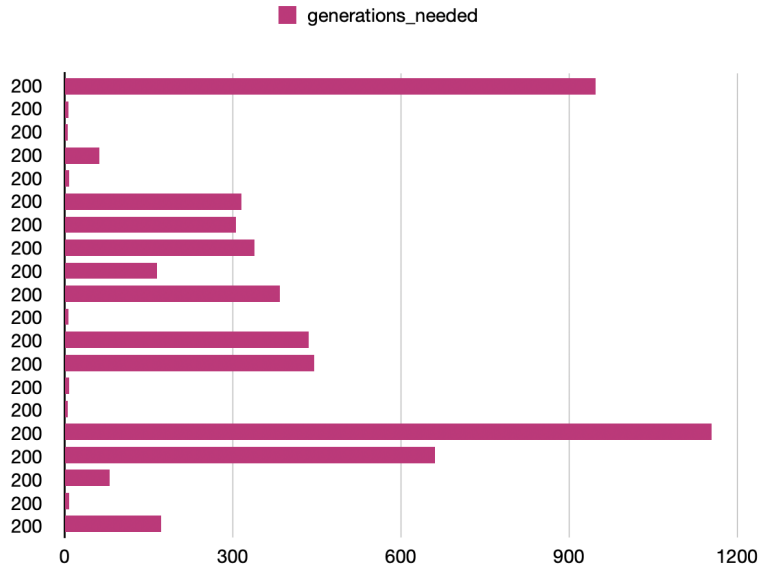

Figure 5: Number of generations needed vs population size of 200

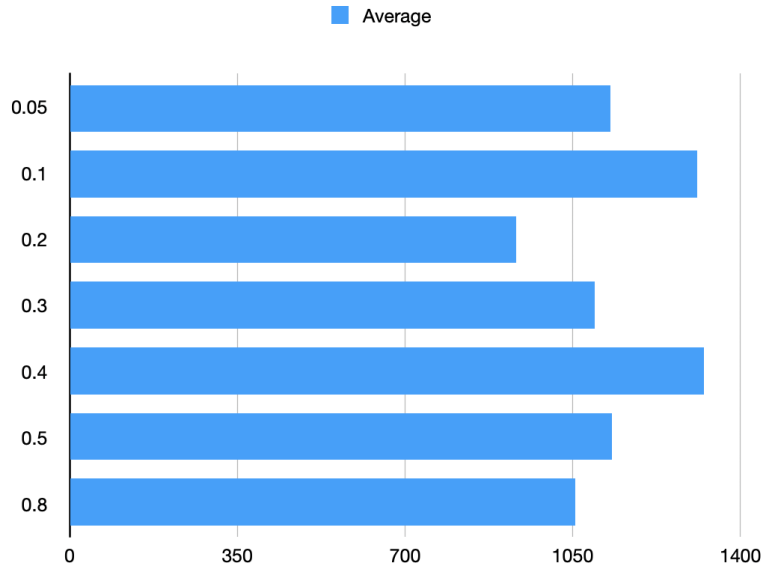

Figure 6: Number of generations needed (average) vs Mutation rate

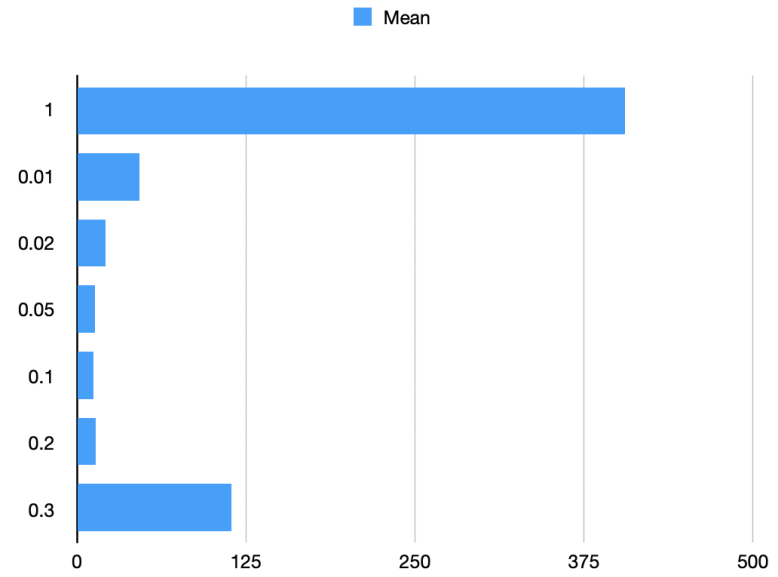

Figure 7: Number of generations needed (average) vs Mutation rate

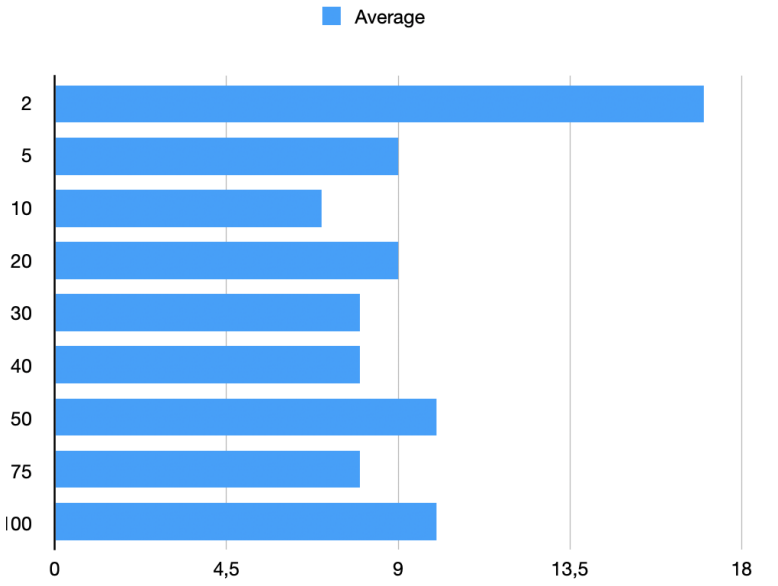

Figure 8: Number of generations needed (average) vs Tournament size