Czech Technical University in Prague

Faculty of Information Technology

MWSAT Solving with the Use of Genetic Algorithm

Anastasiia Solomiia Hrytsyna

January 15, 2022

B221 - Winter semester 2022/2023

TABLE OF CONTENTS

EXECUTIVE SUMMARY

This examination is dedicated to the optimization of genetic algorithm and evaluation of its performance for solving MWSAT problems of different sizes (different numbers of variables and clauses). The main task was to create a proper algorithm (see **Genetic Algorithm**), perform parameter tuning (see **White Box**) and, finally, check its performance on the bigger set of data (see **Black Box**).

During the examination we found out that the best model performance may be reached with the next parameters: population size of 200 individuals, Tournament parent selection method, Uniform crossover method, crossover rate - 80%, minimum mutation rate - 2% and maximum mutation rate - 30%. These settings and implemented model structure was enough to solve 60% of problems, find some legal evaluation (but not the best one) to the 19.3% of problems and did not solve at all the rest 20.1% of MWSAT problems.

INTRODUCTION

In this study the optimization of genetic algorithm was carried out. The main goal of the research is to determine which parameter combination has a better performance (faster finds optimal solution for MWSAT problems with different complexity). So, the central question is next:

Is a genetic algorithm suitable for solving different MWSAT optimization problems? If YES, what parameters provide the best results? And how good are these results?

More details about the task may be found on the link.

MATERIAL

Input Data

As input data (stored in **data_black_box**, **data_white_box** and **data_solutions** folders) we used formulas in CNF and DIMACS format (source here from SATLIB Collection). Selected input data corresponds to the main task and has different solving complexity (number of clauses and variables: 20/71, 20/91, 50/218, 75/325) and different solving strategy (M, N, Q, R), so the algorithm's output was fair.

Necessary Programs

To perform my experiment I used the next programs:

- **dataParser.py** - parses all necessary information from the input file.mwcnf to the ProblemMWSAT class item, so it would be easier to use in genetic algorithm
- **geneticAlgorithm.py** - the main python program to perform genetic algorithm for a single problem
- **whiteBox.py** - 1 test for parameter tuning
- **blackBox.py** - 2 test for final solution evaluation
- **white_box_analysis.ipynb** - jupyter notebook for 1 test results processing and understanding
- **black_box_analysis.ipynb** - jupyter notebook for 2 test results processing and understanding
- **results** - folder, where all of the performance outcomes and measurements of the testing phase are stored

Recourses

All of the experiments have been carried out on Windows 11 operating system (i5-1135G7 processor with 16GB RAM memory). Also, Python programming language was used.

GENETIC ALGORITHM

As it was already mentioned, for SAT problems solving genetic algorithm has been used. The next diagram shows how the whole process was carried out:

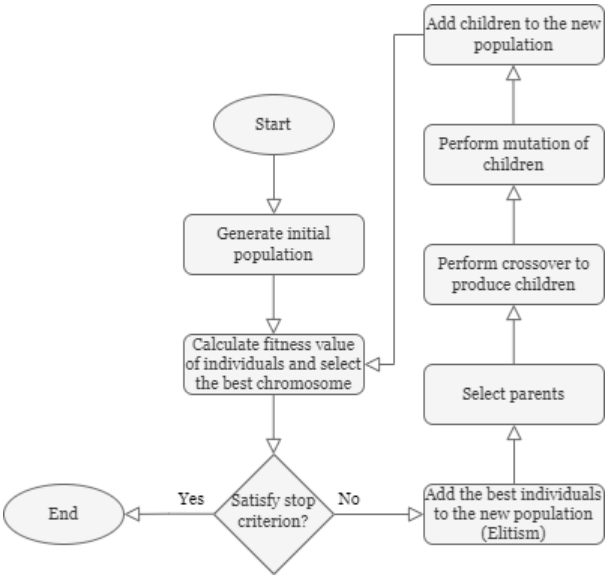

Fig. 1: Genetic algorithm

Initialization

Initial population was created from N (input parameter which marks the size of the population) chromosomes of an exact input size, which corresponds to the task. Each chromosome consists of a number of genes, which marks if the variable on the exact index is negated (0) or no (1). In the initial population these genes are **generated randomly**.

Parent Selection

In my genetic algorithm a few different selection strategies have been implemented:

- **Tournament Selection** - randomly choose 3 individuals and mark the best one as a parent
- **Roulette Wheel Selection** - select parent among array of chromosomes, where the probability of being selected increases with the bigger fitness value
- **Rank Selection** - similar to the previous method, however the probability of selection depends on the rank of each individual

Crossover

Uniform Crossover and **Single Point Crossover** have been used for crossover performing. The first method randomly decides for each gene if we swap it between parents or not, where the second method randomly selects one crossover point and creates children out of different parents left and right parts to the crossover point.

Mutation

In my task I performed a mutation with 25% of genes in chromosome in 3 different ways: **random bit** (gene) **flip**, **swap of two random genes** or **scramble mutation** (shuffle random subset of genes). Each time one random method out of these 3 was used.

Enhancements

I have also tried to better the performance of my genetic algorithm with some small improvements:

Elitism

To each generated population I added the top 3 chromosomes (the one with the biggest fitness value), so the best solutions to the problem survived and our algorithm did not get worse.

Changing mutation

I have also implemented changing mutations through generations (initial minimum and maximum rates are set in the input parameters). The algorithm starts with the minimum mutation rate and then, in case the average fitness value is not changing for too long, the mutation rate is increased by 1%. Additionally, it continues to get bigger till fitness remains the same. However, once fitness changes - mutation rate drops till minimum again.

Population reset

In case mutation rate gets till its upper limit (maximum mutation rate) and our average fitness is not changing I decided to reset the whole population (change it with the new one that is generated completely randomly), set mutation rate to the minimum and continue finding a solution one more time.

Fitness Function

Fitness value of the chromosome depends on the fact if it satisfies all clauses of the problem or not. In case that our chromosome does not satisfy all conditions - fitness value is the number of fulfilled clauses. Otherwise (if our solution is legal) fitness value is equal to the number of satisfied clauses (so it was always bigger than the one in the previous case) plus sum of all weights of non-negated variables. This way, solutions that satisfy all clauses and use more positive (1) variables have much bigger fitness values and are preferred than the others (so they are dominant during parent selection, for example). Still, other solutions' fitness values depend on how bad the chromosome is: the smaller value is used for worse solutions.

WHITE BOX

During white box testing I decided to try all possible combinations of the next parameters:

- **populationSize:** 50 or 200
- **parentSelectionMethod:** "Tournament", "Roulette Wheel" or "Rank"
- **crossoverMethod:** "Uniform" or "Single Point"
- **crossoverRate:** 0.8 or 0.99
- **minMutationRate:** 0.01 or 0.02
- **maxMutationRate:** 0 (without any changing mutation), 0.3 or 0.9

Maximum number of generations was set to 500 in each test. So, together we have 144 different combinations of parameters that were tested on the limited amount of data. For this test I used the first

problems of wuf20-71 (M, N, Q, R) and wuf20-91 (M and N only) - 6 problems overall. The complexity of my experiment does not allow me to run it on other more complex data, so I just tried to test each problem 3 times on the same parameter combination and select average results, so the outcome was not so biased. The whole test lasted about 4 hours and I tried to find out **what parameter combination is the most efficient**.

Experiments

First of all I checked **how successful my algorithm is**. As we can see from **Fig. 2** most often the solution was found all of 3 times (81%). Still other cases were happening - 13% for 2 times founded solutions, 4% for 1 out of 3 solutions and 3% of problems were not solved at all.

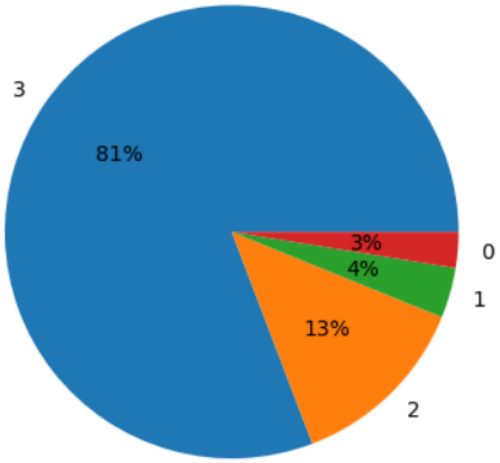

Fig. 2: Successful solution distribution (max number: 3 times)

I also found out that in the most common case if the solution exists - we will find it really quickly. For example we may check **Fig.3** created for the group of tasks that were solved all 3 times. As we may notice mostly the solution was found within 10 sec (however still there are some calculations that need more time).

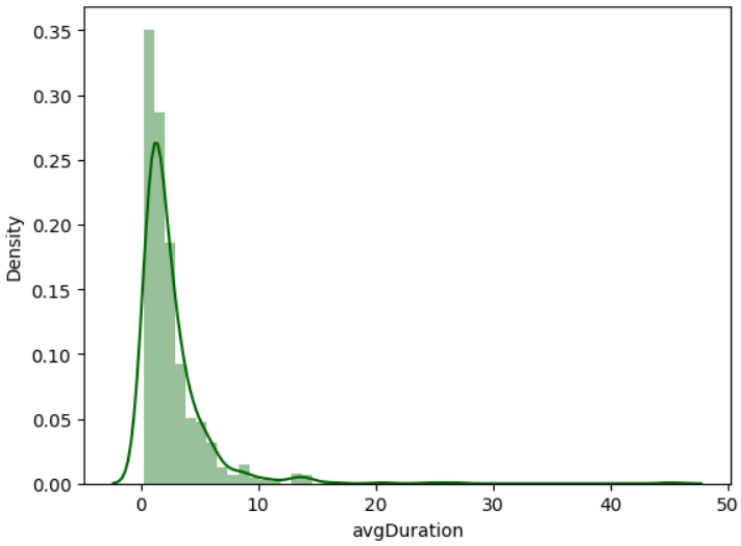

Fig. 3: Average duration for solution finding

In the next step I checked what influence **mutation rate** has on the final outcomes. As we can see from **Fig. 4** minimum mutation rate with both values produces approximately the same number of solved problems. But as regards maximum mutation rate we may notice that maximum 30% of mutation helps significantly to produce better results. Genetic algorithm without mutation increasing (maximum mutation rate is 0.0) produces more unsolved results in comparison to the changing mutation rate, so it may not be used in the final solution.

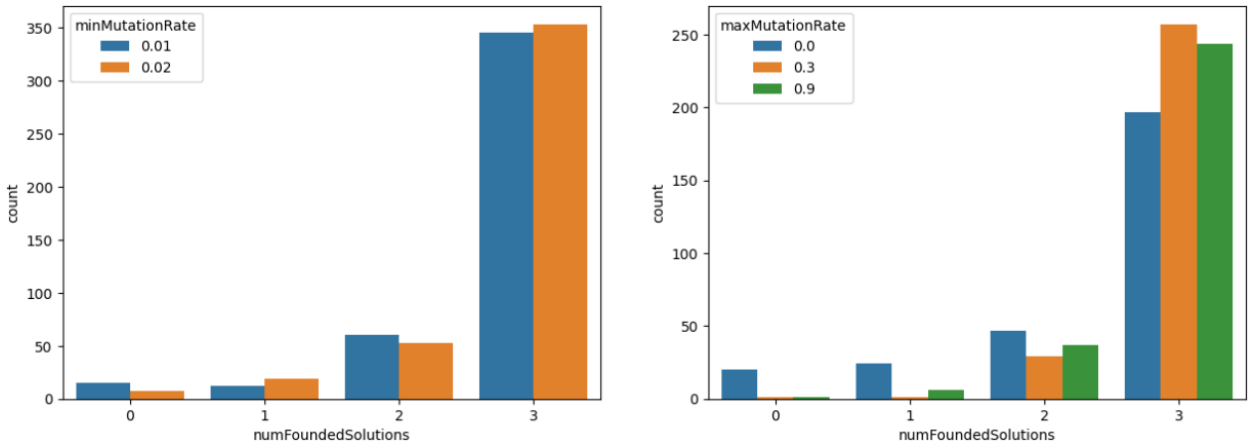

Fig. 4: Mutation rate influence on the algorithm performance

As for the **crossover rate**, we may also notice that it does not have a big influence on the performance: both values produce approximately the same number of solved and unsolved problems.

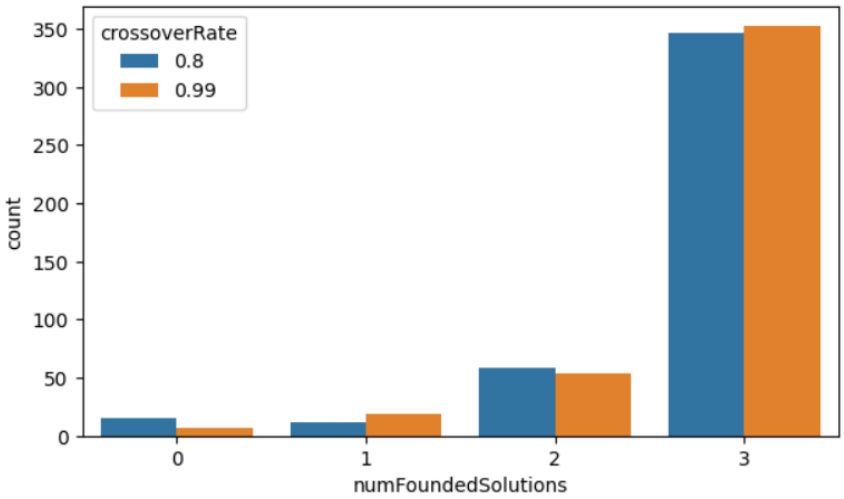

Fig. 5: Crossover rate influence on the algorithm performance

On the other hand we may notice that **population size** influences the performance. The smaller it is, the less successful solutions we may find (see **Fig. 6**). So, in the final test we have to use the bigger value even if it takes more time for calculations.

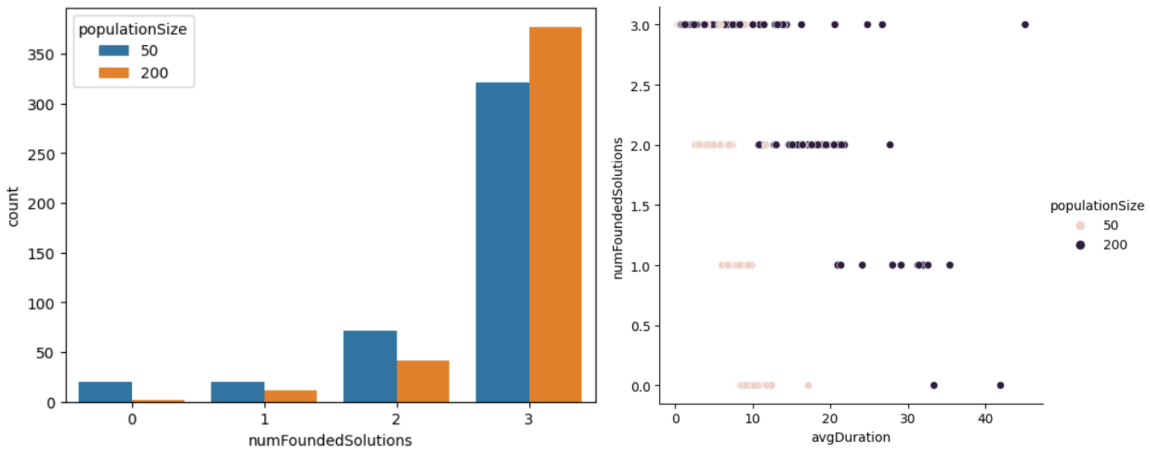

Fig. 6: Population size influence on the algorithm performance

Last but not least, I will compare the combinations of different **parent selection and crossover methods**. As we may notice from **Fig. 7** the lowest probability for finding a solution produces Roulette Wheel Selection (its bars are lower than the others). At the same time we may notice a pattern that the left 4 bars are a bit lower than the right ones. It means that the Single Crossover Method is less effective than the Uniform, so it may not be chosen for the final test too.

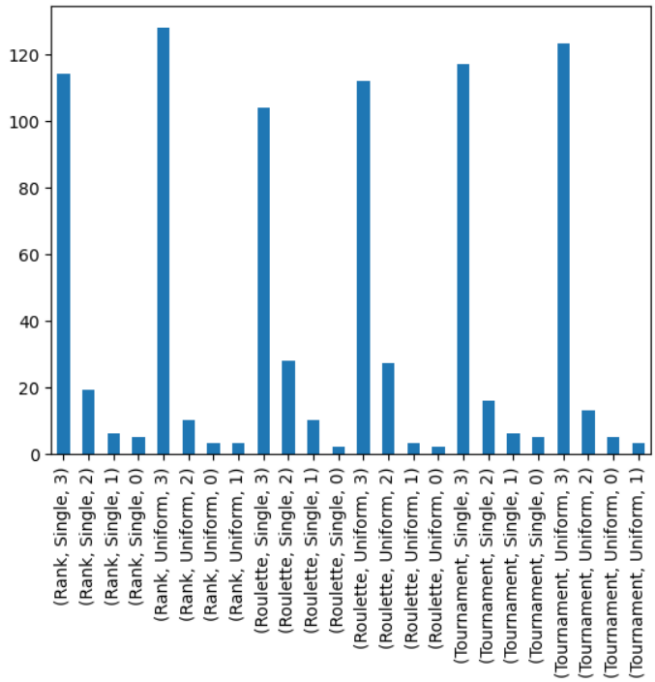

Fig. 7: Parent selection and crossover methods influence on the algorithm performance

Final Parameters Setting

All in all, I sorted the whole dataset of our results by number of found solutions (in descending order) and by average end generation and duration (ascendingly) - these criteria are the most crucial ones. I found out that top 3 parameter combinations are:

1. population size (200)
parent selection method ('Tournament')
crossover method ('Uniform')
crossover rate (0.8)
minimum mutation rate (0.02)
maximum mutation rate (0.3)

2. population size (200)
parent selection method ('Tournament')
crossover method ('Uniform')
crossover rate (0.8)
minimum mutation rate (0.01)
maximum mutation rate (0.3)

3. population size (200)
parent selection method ('Rank')
crossover method ('Single Point')
crossover rate (0.99)
minimum mutation rate (0.02)
maximum mutation rate (0.3)

So, in my final **Black Box** test I decided to use **combination 1** as it produces the best results (3 founded solutions, end generation is 13.6 and duration is about 0.76 sec on average). It also corresponds to the restrictions and conclusions from the **Experiments** part.

BLACK BOX

To perform an experiment at this part I used all data: wuf20-71, wuf20-91, wuf50-218 and wuf75-325 (M, N, Q and R). I tried to solve every problem in a folder, but some of the calculations lasted too long, so I was made to stop my experiment earlier due to my computer limitations - that is why some folders are not fully worked out. Still there are a lot of results of different SAT problems that are going to be compared in this section. Used input parameters have been mentioned in the section above. The whole experiment lasted about 3 days and I was trying to decide **if selected strategy is good enough for solving MWSAT problems**.

So, the final results consist of 11201 entries, where each entry marks the solution to an exact problem. As I have already mentioned - I did not manage to run a genetic algorithm for all of the problems, so as we may see on **Fig. 8** there are a smaller number of problems with 75 variables and 325 clauses. Nevertheless, for every solution I calculated its duration, number of necessary generations, maximum mutation rate and the best reached weight.

Fig. 8: Calculated problems distribution

At about 24.6% of M, 33.3% of N, 46.6% of Q and 52.7% of R-complex problems reached 500 generations. Still, we may see from **Fig. 9** that lots of problems of any complexity level were solved till 100 generation.

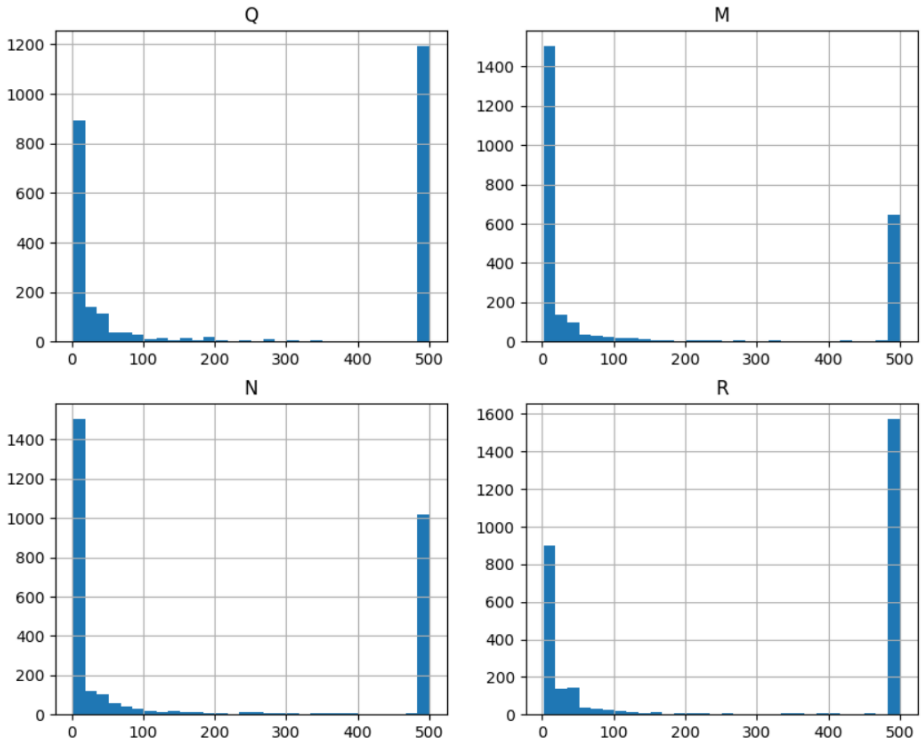

Fig. 9: Histogram of generations needed for finding a solution

In case an optimal solution was found (~60% of cases) we may check the next statistics (see **Fig. 10**):

	endGeneration	Duration	maxMutation
count	6780.000000	6780.000000	6780.000000
mean	37.000885	6.987228	0.090015
std	73.843453	42.935288	0.101461
min	2.000000	0.188271	0.020000
25%	8.000000	0.890276	0.020000
50%	11.000000	1.185249	0.040000
75%	25.000000	2.682356	0.090000
max	499.000000	3141.805944	0.300000

Fig. 10: Descriptive statistics for the successful solutions of MWSAT problems

On the opposite side, on **Fig.11** we may see the distribution of the problems, the right evaluation of which was not found at all (it means that even after 500 generations there are some clauses that are not fulfilled). There are 2256 of such problems (~20% of cases), most of which are bigger MWSAT problems (50 variables with 218 clauses). I suppose the statistics may not be even worse for the 75-325 dataset in case we solve equal amounts of problems as for the 50-218.

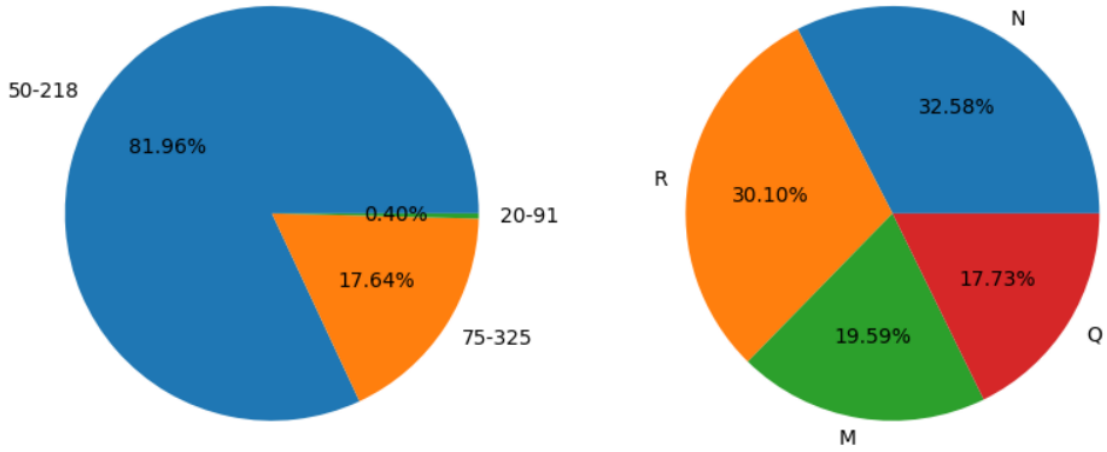

Fig. 11: Unsolved problems distribution

However there are a number of outcomes that did not reach optimal weight results, but still found some variable evaluation that satisfies all clauses (~20% of cases). As we may see from **Fig. 12** (left graph) the more complex the problem is, the less successful final solutions (with an optimal weight) we may find. On the other hand, as we may notice from the right graph, we may find some variable evaluation that satisfies all of the clauses in any complex case with equal probability, however, this solution is not the best one for our optimization problem.

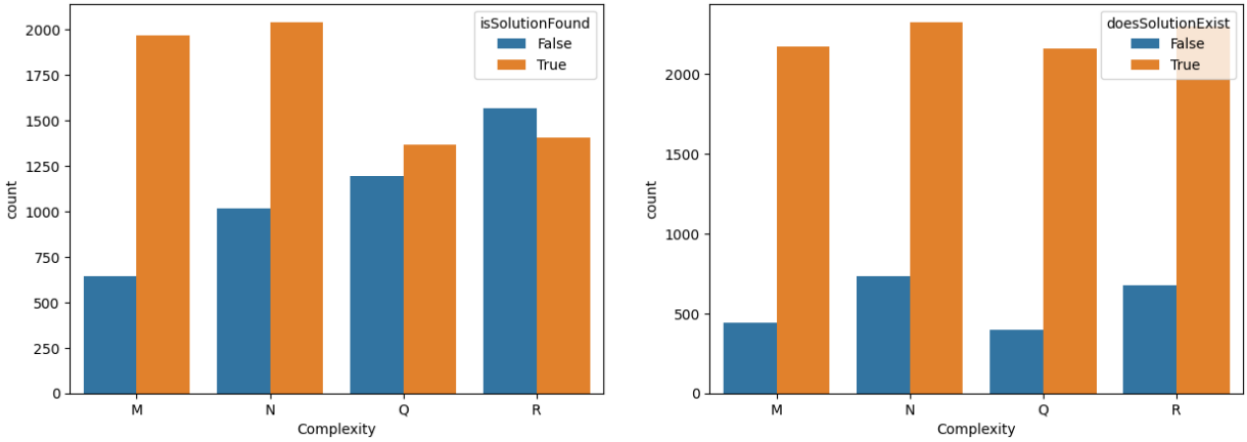

Fig. 12: Successful (left) and unsuccessful (right) solution distribution depending on the problem complexity

I also decided to compare some existing solutions that are not good enough for considering them as optimal ones. I counted the value of error for each problem ((optimal weight - reached weight) / reached weight * 100%) and plotted its distribution on **Fig. 13**. We may see that most of the errors are smaller than 50%, so they have a big probability of being solved in case more generations are allowed.

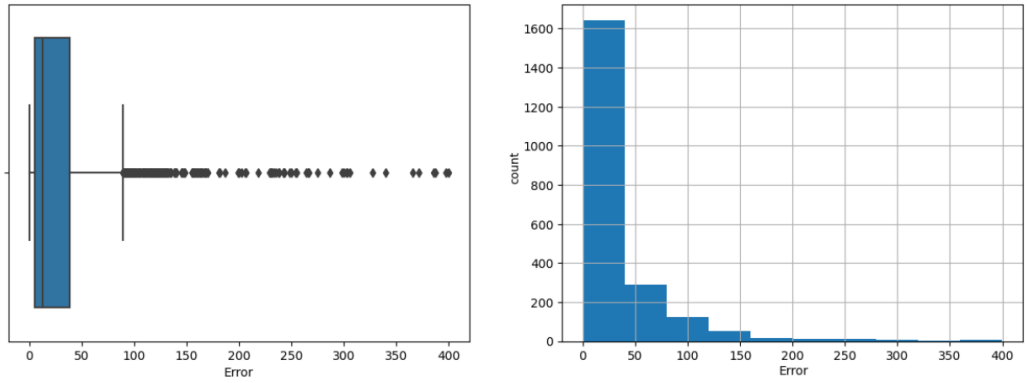

Fig. 13: Error value analysis

Last but not least, I decided to check how mutation rate was influencing our outcomes. As we may see from **Fig. 14** mostly it remains really low, however there were cases when it reached 30%. This way a huge number of chromosomes were changed (or even population reset was performed).

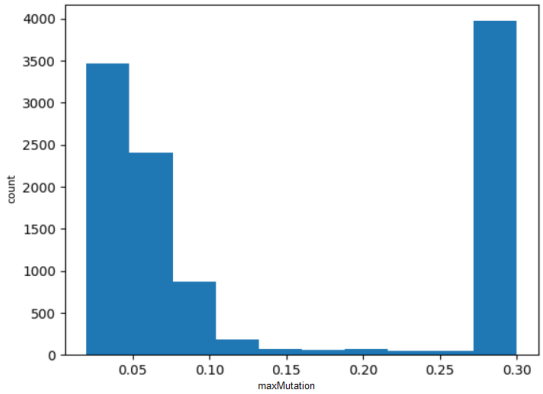

Fig. 14: Maximum reached mutation rate distribution

That is why I decided to check the solutions where mutation rate reached its limits. As we may see from **Fig. 15** still there are a dominant amount of unsolved problems, but the successful ones appear too. That means we get out our solution from the local maximum and improve our general performance.

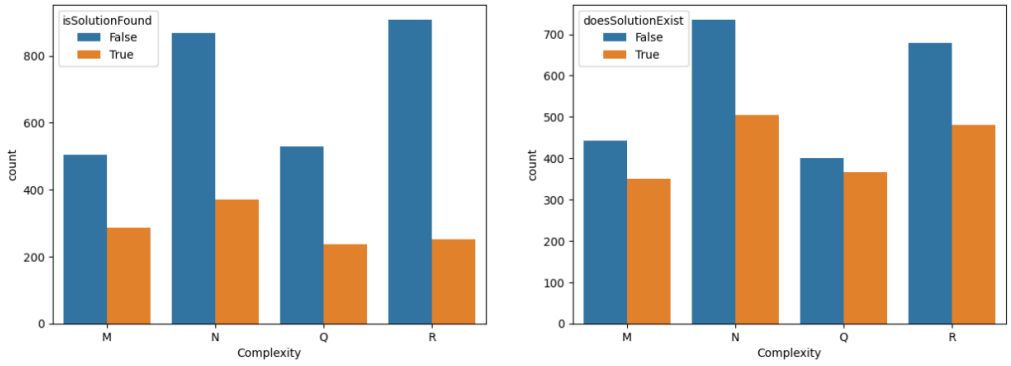

Fig. 15: Successful (left) and unsuccessful (right) solution distribution depending on the problem complexity

DISCUSSION

Due to the previous conclusions and carried experiments we may clearly say that our genetic algorithm model with selected input parameters produces good enough results for solving MWSAT problems. 60% of all tasks reached optimal results and the next 20% were solved (but not optimally).

For the future examinations it will be great to check other input parameter values and perform their tuning on a bigger variety of data. It will make our algorithm even faster and more suitable for different problems.