Patryk Stroński 242564

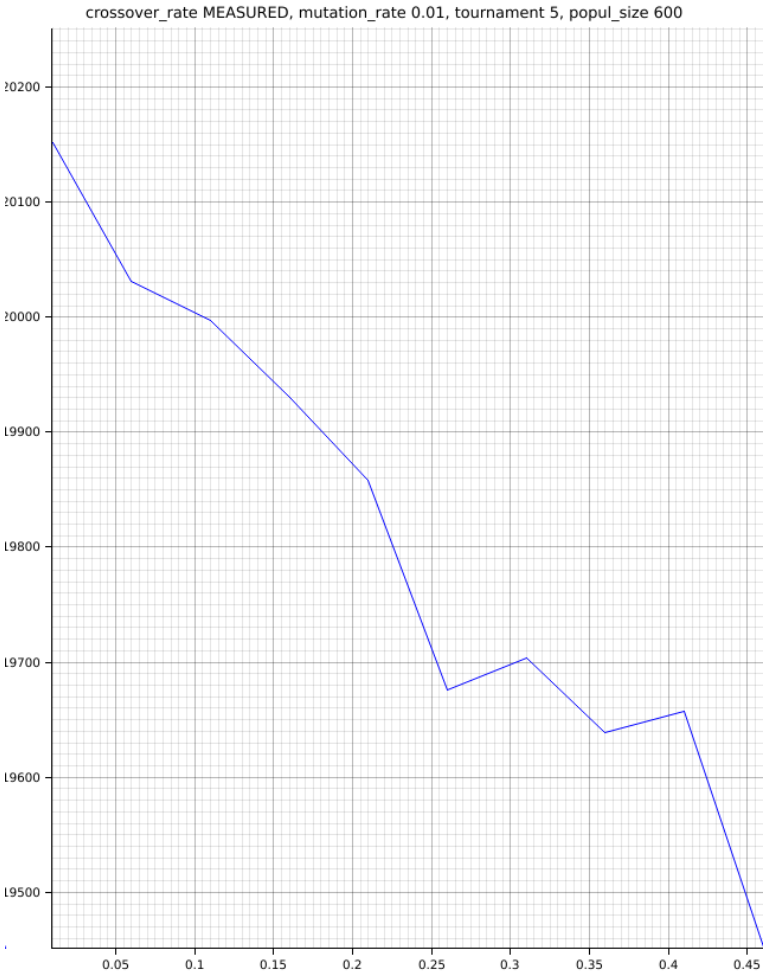
Assignment 1 – genetic algorithm

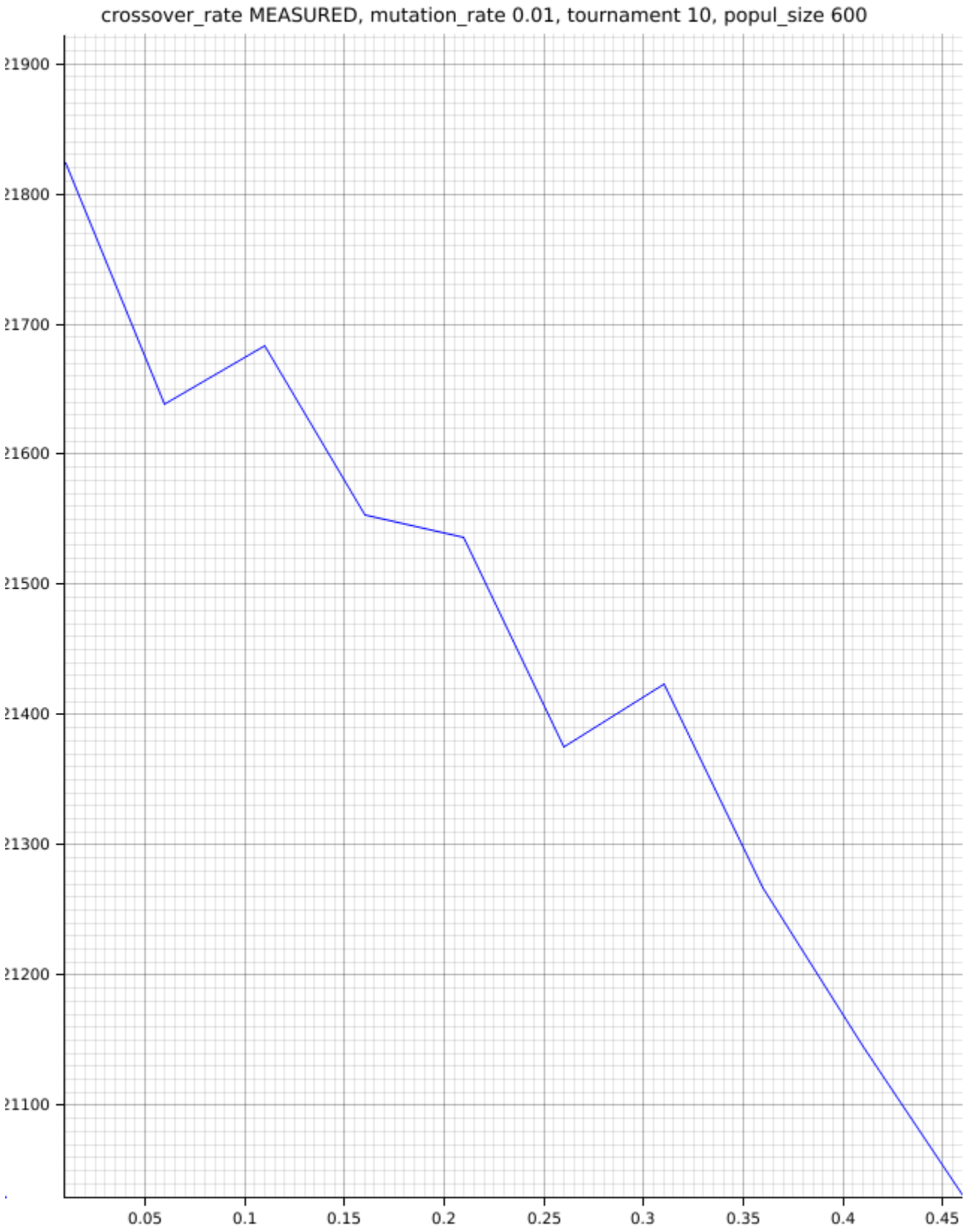
Introduction

The genetic algorithms have been implemented using Rust programming language. The Knapsack list set has been implemented using struct Knapsack, whereas the individuals, as the Vectors of boolean values.
All the result files are included in the zip.
The number of iterations (generations) is equal 50. After those 50 generations pass, the winners are evaluated based on evaluate function which evaluates the quality of the individual or returns 1 in case the individual is overburdened (by size or weight). The other, measured values, are all written on the graphs. Each graph is the result of 5 subsequent computations and taking average of those. This is good especially if we have individuals generated randomly each time the population is initialized.

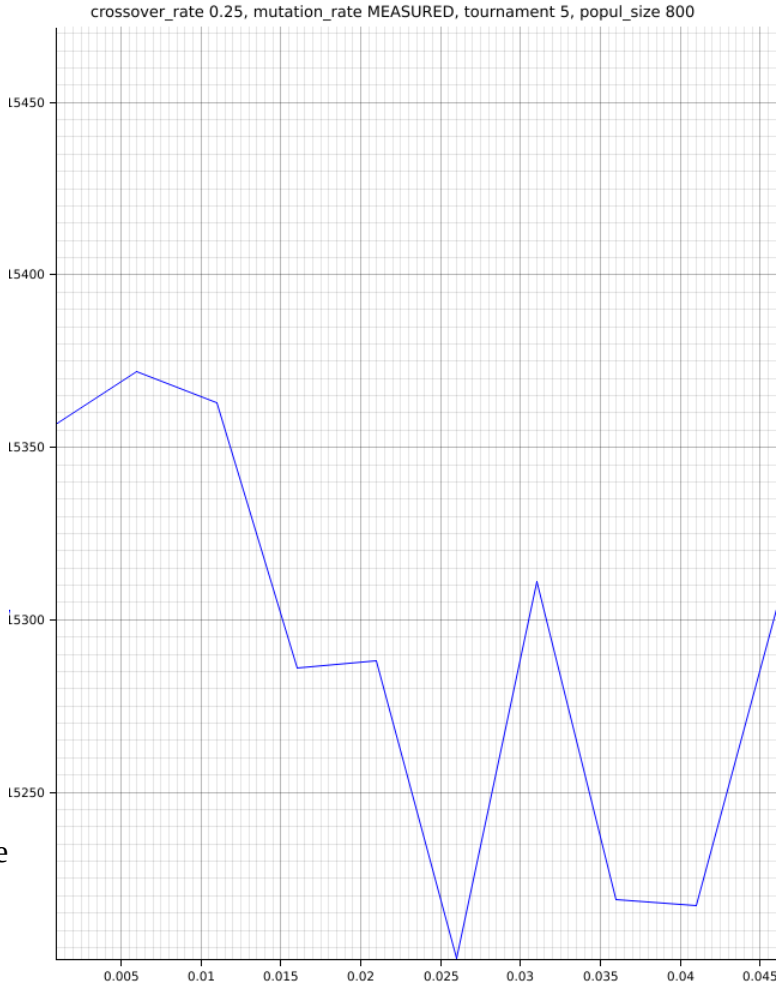
Analysis of the crossover:

The crossover in such big set occurred to be non-effective. The crossover is implemented in this way. Crossover_rate is the probability parents do crossover. The outcome is computed using random function. Here we see that the crossover does dist the order with which the populations have been created. The bigger is the crossover, the worse the result. Even though, for crossover_rates we see close to inverse proportion – with the increase of crossover_rate, the decrease of fitness goes. That might be also because of small (comparing to population size) tournament value. For 10, on the other hand, we achieve such diagram .



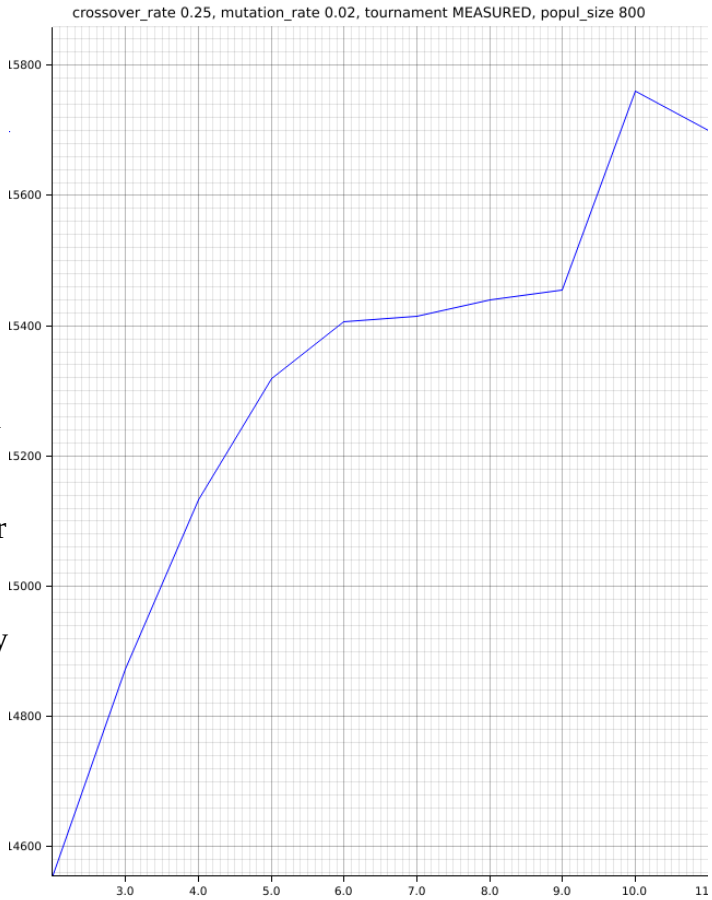

crossover_rate MEASURED, mutation_rate 0.01, tournament 10, popul_size 600
21900
21800
21700
21600
21500
21400
21300
21200
21100
0.05
0.1
0.15
0.2
0.25
0.3
0.35
0.4
0.45

Analysis fo the mutation rate



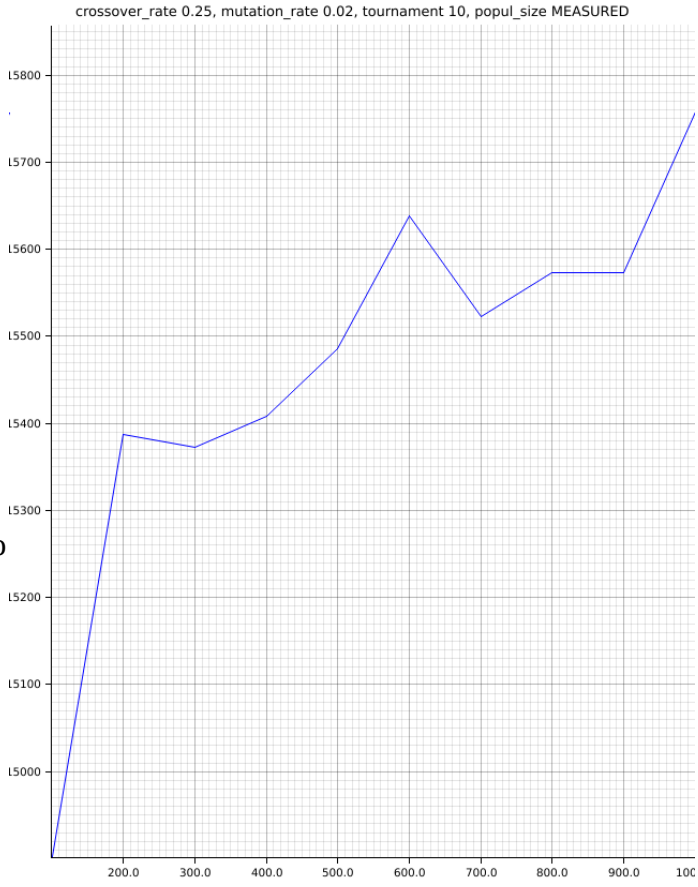
Here we see that small but possible mutation is generally beneficial for our generation. The best values are achieved for mutation_rate between 0 and 0.014. This is because this mutation has a chance to enhance the individuals, but also the chance to deteriorate them is quite low. The rest of the diagram show clearly that for big mutation rates, the values are much more uncontrolled.
The number of genes to mutate is calculated based on this equation:
*let nmb_genes_to_mutate = ((mutation_rate * 1000.0) as u32;*
The fractional part that might last here is cut off. Therefore if we mutate around 6 genes for 1001 allel-long chromosome, the mutation is effective. Later on, the effectiveness decreases and shows uncontrolled behavior.

Analysis of the tournament size



Tournament is the amount of individuals taken in consideration to choose best individuals from. It is the range. Each population generation individuals are shuffled in order to achieve more entropy and randomness. The tournament is best for value 10. This might be cause by big population – in big populations it is better to choose from big amount of individuals, as there is simply more choices and always the best are chosen and mixed with each other. Of course too big tournament size leads to very narrow amount of individuals to be taken. For instance, in tournament equal 50 far more individuals would be omitted than for tournament=10. And for tournament 3 nearly every individual is taken into account. This causes mixing bad traits with good traits and it creates mediocre traits.

Analysis of the population size



Here the amount of people in population shows another quite clear ration: the bigger population, the better score. Here, even though, it is important to note the tournament size: 10. It is, in my opinion, medium-sized tournament which gives many opportunities for bigger sets. It uses the big variety of individuals and chooses a considerate part of them. Based on this and previous diagrams also we can clearly see the correlation between population size and torunament size. Big tournament size works good for big population size. For smaller populations, the tournament size needs to be also considerably smaller as the variety and diversity of individuals is shrinking.

Based on the above diagrams we can derive the best values:

```
crossover_rate: 0.001
mutation_rate 0.006
population_size 1000
tournament_size 10
value: 22450
```

Ant colony algorithm

The ant colony algorithm gave the following value: `16026.` We can see that it was very effective in this case. Only in case of mutations and crossovers the values were higher. The ant colony algorithm utilized graph evaluation. It was basing on tournament size and was using different tournament sizes each evaluation. For each evaluation new ones were created therefore the result is taken based on different permutations of tournament size. Crossover was stable for all permutations and it was 0.05. The ant colony algorithm worked basing on finding best values amongst 5 generations. The available tournament sizes were from 3 to 7. Population size was 1000.

Final thoughts:

The evolutionary method shows very well how the distribution fo genes occur. The genetic algorithms also show, as the above charts, that many factors are often controlled randomly therefore it is hard to predict some effects, even though It is easy to see some tendency in each behavior. Based on that we can derive:

1. Often crossovers can distort the good populations
2. Mutation is good if kept at small amount
3. Big torunament size works good with big amount of individuals.
4. Many individuals means more possibilites of having good data
5. Non-evolutionary methods can also be efficient