Lazy Student

EARIN Miniproject 1 Final Report

Jakub Duchniewicz, Maciej Świderski

April 28, 2020

1 Introduction

The premise of the task is to choose a subset of subjects that will both satisfy the total ECTS requirements and fit in the amount of days the student is willing to study each semester. There are N semesters and k ECTS per semester, student is willing to spend a maximum of m days each semester on studying. Additionally some courses have prerequisites which impact their availability.

2 Algorithm

While the first approach stated in the initial report was a solution to the problem at hand, it did not make use of any metaheuristic whatsoever. That is why we have decided to change our approach after finishing this particular one. It is attached in its complete form as well as an example of 'pure' algorithmic approach to the problem. The proper algorithm is a genetic one, where the population comprises distinct individuals, mating every generation and producing offspring gradually better in quality. One individual is a list of semesters, a semester being a sequence of '0's and '1's which state whether a subject was taken this particular semester or not. The first generation is generated until it satisfies constraints, then nascent generations are created taking into account parameters: elitism (whether it is enabled and its percent) and percent of current population crossing over. These parameters can be tweaked to user's preference. Moreover it sometimes happens that the result vacilitates, therefore several latest best results are recorded and compared to assert convergence. The fitness of our algorithm is determined as the difference between maximal number of days our student wants to study each day, and actual achieved, it is negative for valid results, and the lower it is the more 'lazy' given choice of subjects is.

2.1 Optimization parameters

2.2 Elitism on/off

For this section, following parameters were set up: subject set size $sz = 30$, number of semesters $N = 5$, $k = 15$, $m = 30$. As discussed previously, whether we enable elitism or not, is a deciding factor in convergence of our algorithm. Hence first we ran the tests for elitism enabled or not. In figure 1 and 2 can be seen respectively: time to evaluate and score of 20 runs with eltisim enabled and disabled. As it can be observed elitism is responsible for decreasing the time it takes for obtaining the solution, because it promotes the choices of best-fit individuals. However ultimately elitism does not have such an impact on the converged score - it is just a shortcut.

2.3 Elitism percent tweaking

When tweaking this parameter, three values were investigated: 5, 20 and 50 percent. From the graph it can be deduced that the highest elitism seems to provide the best calculation time while still not influencing the

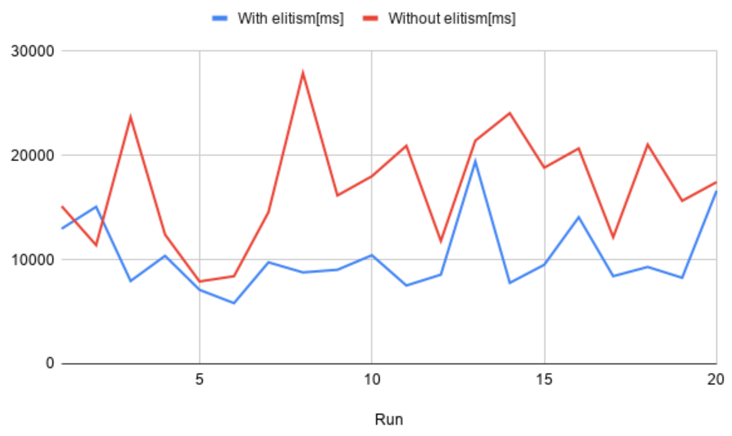

Figure 1: Tests for elitism on and off time comparison.

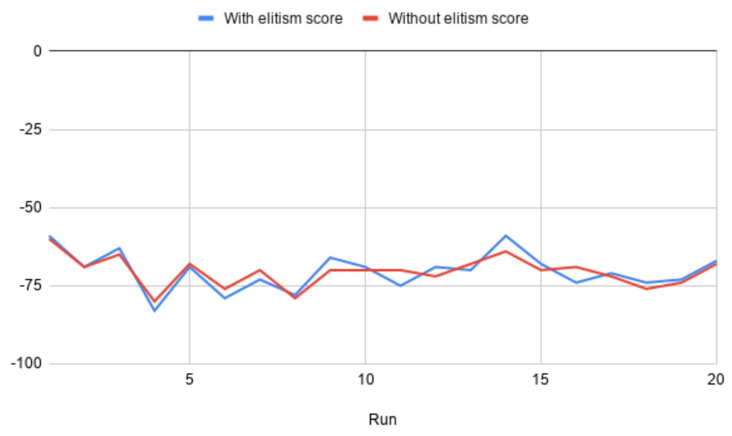

Figure 2: Tests for elitism on and off score comparison.

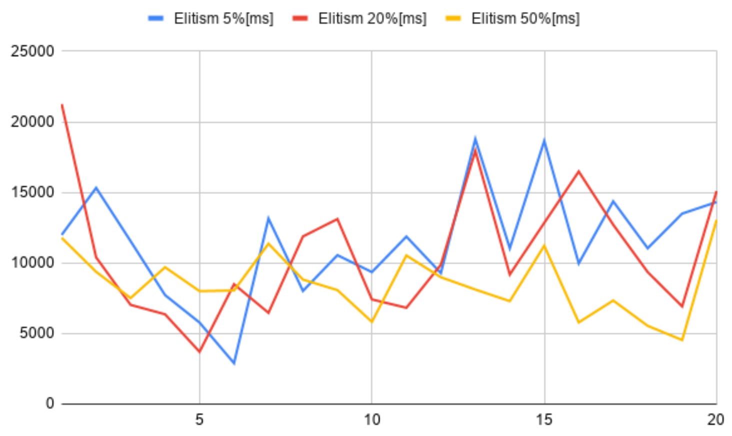

Figure 3: Elitism percent tweaking time comparison.

score. Figures 3 and 4 show this tweaking.

2.4 Crossover percent tweaking

Another parameter that we tweaked was crossover percent. This particular parameter was most surprising to tweak, because with the reduction of crossover percent, the convergence rate (time to solve) grew, without influencing greatly the resultant score. Figures 5 and 6 present this.

2.5 Convergence iterations tweaking

Additional comparison can be made for the convergence parameter, which specifies when the algorithm should stop seeking for a solution. By default it is set to a value of 20 which was set empirically. This can be seen in figures 7 and 8.

3 Sample algorithm run

Below is presented an exemplary run of the algorithm:

```
Input:
3 10 20
4 4
3 2
2 3
4 2
3 4
3 3
3 2
4 2
4 2
4 4
3 4
4 3
2 2
4 3
4 3
#
8 5
10 10

Output:
################################################
Converged - no change after 20 iterations.
################################################

Semester: 0 | 1 7 11
Semester: 1 | 2 6 8 12
Semester: 2 | 3 5 15

Algorithm ran 168 miliseconds.
```

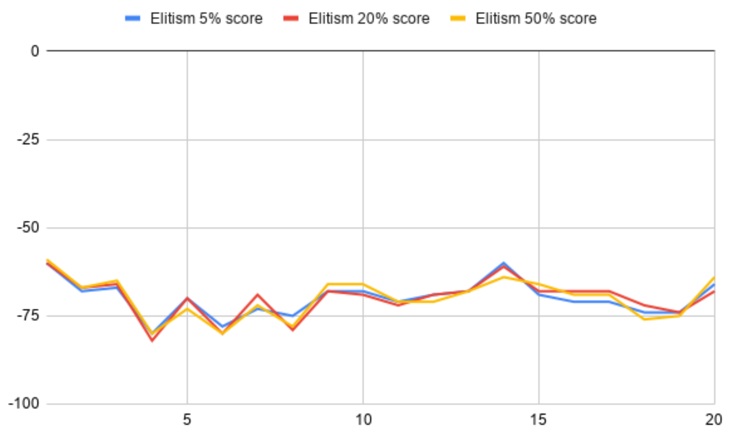

Figure 4: Elitism percent tweaking score comparison.

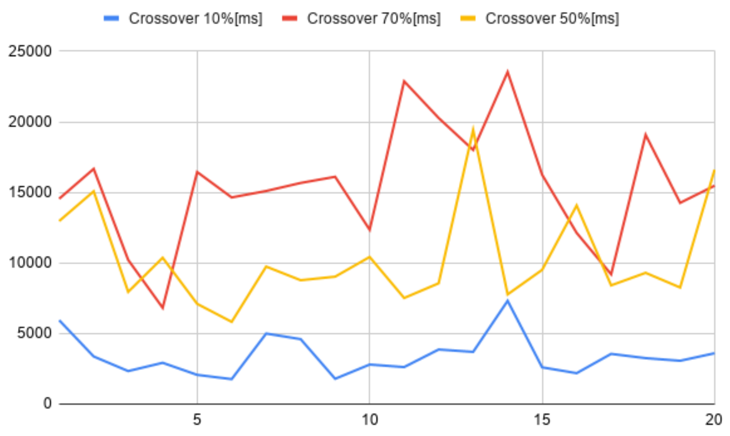

Figure 5: Crossover percent tweaking time comparison.

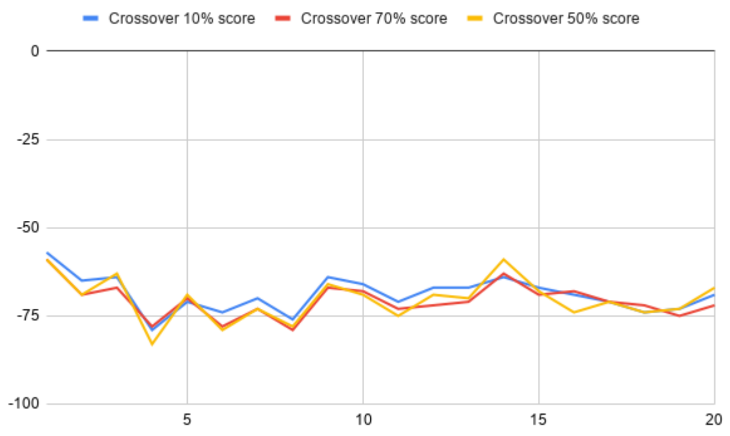

Figure 6: Crossover percent tweaking score comparison.

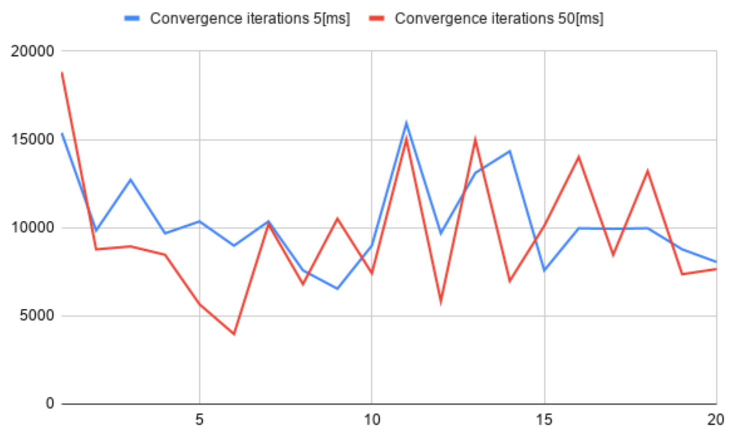

Figure 7: Convergence tweaking time comparison.

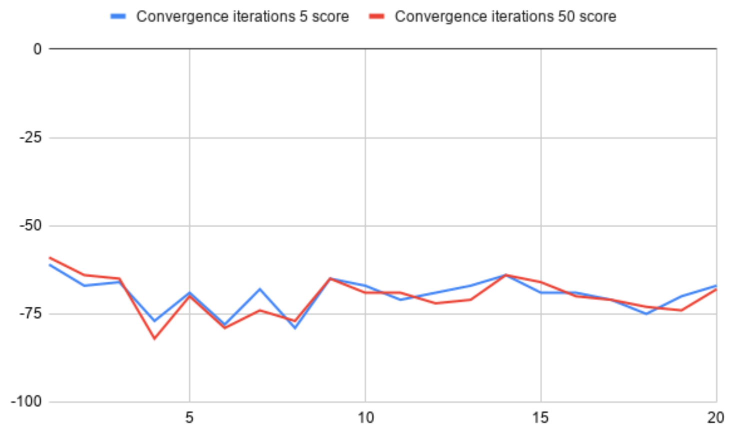

Figure 8: Convergence tweaking score comparison.

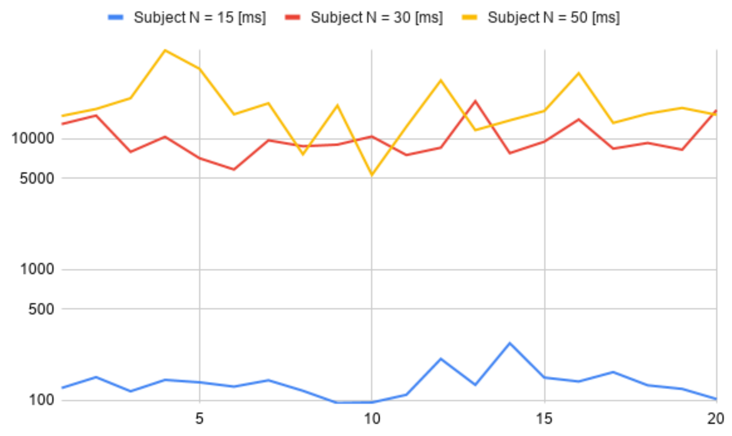

Figure 9: Problem size time comparison.

4 Problem size comparison

Three sizes were compared with adjusted parameters for each one of them. For subject set size sz = 15 - N = 3, k = 8, m = 15, for sz = 30 - N = 5, k = 15, m = 30, and sz = 50 - N = 5, k = 10, m = 30. With higher number of semesters, and 50 subjects, this program could not find any reasonable starting combinations under 10^6 combinations. These figures present these comparisons 9, 10. It can be clearly seen that with higher sz and N the problem grows much more complex in terms of time.

5 Discussion

Several parameters heavily influence the result of this algorithm and these are: elitism, crossover percentage and problem size. Deducing from our experiments we believe elitism could be removed and the algorithm would still converge quite quickly. However, crossover is much more crucial to the genetic algorithm's performance. When it is too big, too many individuals mate and introduce too much viable candidates which ultimately penalize this algorithm's convergence. Conversely, with reduction of crossover percent, the convergence rate grows, as the population has fewer 'optimal' individuals to mate from. What could be investigated more, is the problem complexity correlation with number of semesters and possible subjects set. When they grow, problem quickly becomes infeasible to solve. Two more parameters remain and these are: population size and mutation algorithm. Mutation algorithm chosen, trades only a single bit between two individuals' genes, hence being just slightly potent. There is probably a better one for this kind of problems, however we did not manage to invent it.

6 Conclusion

Genetic algorithm proved to be a noteworthy solution to this kind of problem. While the initial approach with A* was not inherently bad, it just proved infeasible after writing some preliminary code for it. This kind of algorithms can give the developer some joy, because of their randomness and variety of parameters to tweak.

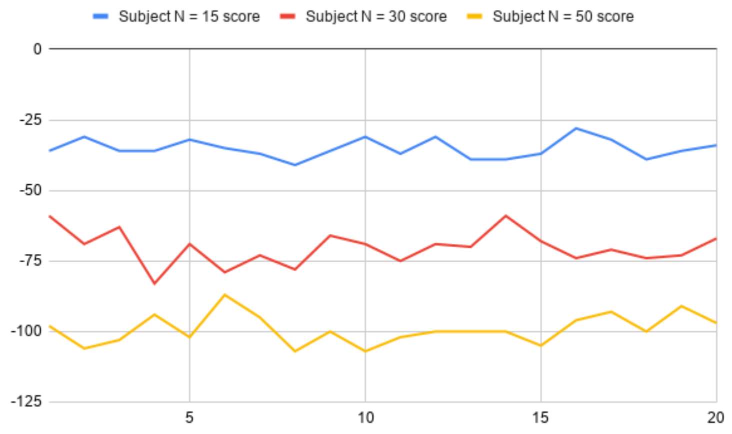

Figure 10: Problem size score comparison.