Information Management School

Computational Intelligence for Optimization Project

MASTER PROGRAM IN DATA SCIENCE AND ADVANCED ANALYTICS

SOLVING A UNIVERSITY SCHEDULING PROBLEM WITH GENETIC ALGORITHMS

Group: ***Pastéis de Data***

https://github.com/MajdAlAjlaniiii/timetable-scheduling-ga

Diogo Pires, 20230534

Majd Al Ajlani, 20230767

Manuel Gonçalves, 20230466

Maria Batrakova, 20230739

June, 2024

NOVA Information Management School
Instituto Superior de Estatística e Gestão de Informação
Universidade Nova de Lisboa

INDEX

1. GitHub Code Repository

Link to the Repository: https://github.com/MajdAlAjlaniiii/timetable-scheduling-ga

Division of labor: In this project, all team members contributed equally to both the coding and the report writing. Although specific tasks were assigned to each member, such as sections of the code or parts of the report, we maintained a collaborative approach throughout the process. Every team member reviewed each other's work, provided feedback, and assisted whenever necessary.

2. Timetable/Scheduling Problem

2.1. Objective

In this Computational Intelligence for Optimization project, we'll implement Genetic Algorithms (GAs) to solve a University Course Scheduling problem. GAs are search heuristics that mimic the process of natural evolution to generate high-quality solutions for optimization and search problems. By encoding possible schedules (solutions), GAs will evolve these solutions through selection, crossover, and mutation operations. The goal of this project is to develop a timetable that satisfies all hard constraints, such as preventing overlaps, and that optimizes soft constraints, like accounting for Professor's preferences. Through iterative improvement, the GA aims to find the most efficient and practical scheduling arrangement for our university courses.

2.2. Data

We created the data used for the design of the schedules by creating lists of multiple classrooms (e.g. "A144", "A120"), possible timeslots (e.g. Monday, Wednesday, and Friday at 09:00 - 10:00), professors, departments, and courses. Each classroom has its own seating capacity, to use as a constraint later on, and each timeslot, professor and course has their own number or ID to facilitate debugging. Every course has specific professors, their preferences, and a maximum number of students, and each department is composed of specific courses.

2.3. Hard and Soft Constraints

Our fitness score calculations are based on the number of conflicts arising from adherence or violation of hard and soft constraints. As the names imply, hard constraints incur a heavier penalty while soft constraints incur a smaller penalty. After testing multiple combinations, we opted for a hard constraint penalty of 1 and a soft constraint penalty of 0.01, this allowed us to clearly distinguish which solutions complied with the model's constraints when evaluating fitness scores. Our model is designed so that certain key requirements are mandatory, for example, one fundamental requirement is that all courses must be scheduled. Also, every course must be assigned to a timeslot, a professor, and a classroom.

Starting with the **hard constraints**, firstly, each classroom's capacity must be sufficient to accommodate all students enrolled in the course. A violation occurs if the room's capacity is less than the number of students. Secondly, no overlaps are allowed in the schedule. This includes room overlaps, where two different classes cannot be scheduled in the same room at the same time, and professor overlaps, where a professor cannot be scheduled to teach more than one class at the same time.

Soft constraints, while less critical, are still important for creating an optimal schedule. Some professors have preferred timeslots for teaching, and if a class is scheduled outside these preferred timeslots, it incurs a soft penalty. Additionally, courses may have preferred rooms based on factors such as proximity to department offices or availability of specific equipment. If a class is not scheduled in one of its preferred rooms, it also incurs a soft penalty.

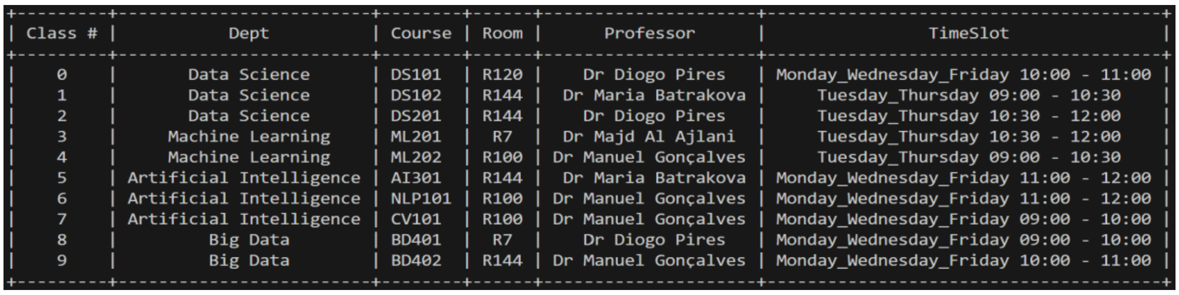
By incorporating these hard and soft constraints into our GA, we aim to generate schedules that are both feasible and optimized for the preferences and capacities of the university's resources.

3. Genetic Algorithm Overview

We incorporate elitism in our genetic algorithm with all selection techniques to ensure that the best solutions are carried over to the next generation. This practice helps maintain the quality of solutions and accelerates the convergence of the algorithm towards an optimal or near-optimal solution.

3.1. Representation

Here is the example of one individual, i.e. schedule:



```
+---------+------------------------+--------+------+--------------------+---------------------------------------+
| Class # |          Dept          | Course | Room |     Professor      |               TimeSlot                |
+---------+------------------------+--------+------+--------------------+---------------------------------------+
|    0    |      Data Science      | DS101  | R120 |   Dr Diogo Pires   | Monday_Wednesday_Friday 10:00 - 11:00 |
|    1    |      Data Science      | DS102  | R144 | Dr Maria Batrakova |     Tuesday_Thursday 09:00 - 10:30    |
|    2    |      Data Science      | DS201  | R144 |   Dr Diogo Pires   |     Tuesday_Thursday 10:30 - 12:00    |
|    3    |    Machine Learning    | ML201  |  R7  | Dr Majd Al Ajlani  |     Tuesday_Thursday 10:30 - 12:00    |
|    4    |    Machine Learning    | ML202  | R100 | Dr Manuel Gonçalves |     Tuesday_Thursday 09:00 - 10:30    |
|    5    | Artificial Intelligence | AI301  | R144 | Dr Maria Batrakova | Monday_Wednesday_Friday 11:00 - 12:00 |
|    6    | Artificial Intelligence | NLP101 | R100 | Dr Manuel Gonçalves | Monday_Wednesday_Friday 11:00 - 12:00 |
|    7    | Artificial Intelligence | CV101  | R100 | Dr Manuel Gonçalves | Monday_Wednesday_Friday 09:00 - 10:00 |
|    8    |        Big Data        | BD401  |  R7  |   Dr Diogo Pires   | Monday_Wednesday_Friday 09:00 - 10:00 |
|    9    |        Big Data        | BD402  | R144 | Dr Manuel Gonçalves | Monday_Wednesday_Friday 10:00 - 11:00 |
+---------+------------------------+--------+------+--------------------+---------------------------------------+
```

We chose this representation because it provides a clear and structured way to include all necessary details of a schedule, making it straightforward to manipulate and evaluate within the GA. Each class in the schedule is characterized by its department, course, room, professor, and timeslot (timeslot part consists of specific weekdays plus exact time of either 1 or 1.5 hours). Departments and courses are 'fixed' to ensure they will stay the same from generation to generation, and if we want to scale up the data with more classes, it won't be a problem.

The representation is handled by the *Schedule class*, which encapsulates the details of the schedule, including the list of classes, the number of conflicts, fitness value, and a flag to track changes in fitness. This kind of representation ensures that all relevant information is included, allowing the GA to perform accurate fitness evaluations and apply genetic operators effectively.

3.2. Fitness Function

We began with a simple version of fitness function that considered only hard constraints.

$$\text{Fitness Score} = \frac{1}{\#\ \text{hard\_constraint\_violations} \times \text{HARD\_CONSTRAINT\_PENALTY} + 1}$$

The number of conflicts is multiplied by the hard constraint penalty, ensuring the fitness score is inversely proportional to the number of hard constraint violations. The optimal solution, with no hard constraint violations, achieves the maximum fitness score of 1.

Later, we came up with another variant of fitness function that considers not only hard constraints, but also soft constraints violations:

$$\text{Fitness Score} = 0 \; - (\#\ \text{hard\_constraint\_violations} \times \text{HARD\_CONSTRAINT\_PENALTY}) + \\ + (\#\ \text{soft\_constraint\_violations} \times \text{SOFT\_CONSTRAINT\_PENALTY})$$

Hard constraints are more critical and incur higher penalties, while soft constraints are desirable but less critical, incurring lower penalties, so *HARD_CONSTRAINT_PENALTY* and *SOFT_CONSTRAINT_PENALTY* weights were adjusted in the way to promote explainability to values of fitness we get. The optimal solution in this case with no hard constrain violations and satisfying soft constraints requirements achieves the maximum fitness score of 0 making this a maximization task where the goal is to maximize the fitness score by minimizing violations. From now on we use the second variant of fitness function.

3.3. Selection methods

To test the impact of different selection strategies on the performance of our GA, we evaluated three different selection methods: **tournament selection** (of size 3) and **ranking selection**. In the tournament selection, the algorithm selects the best schedule from a random subset of a predefined size, while in the ranking selection, it assigns a selection probability for each schedule in the population based on the ranking.

3.4. Crossover

Multiple crossover strategies were tested, namely: **single point crossover**, **cycle crossover,** and **uniform crossover.** While the first two were covered at the classes, uniform crossover is worth describing briefly: it is a crossover technique where each gene (or element) in the offspring's chromosome is independently chosen from one of the parents with a fixed probability, typically 0.5. This means that each gene in the offspring has an equal chance of being inherited from either parent. Note that all genetic operations are performed on rooms, professors, and timeslots. The crossover rate used in our genetic algorithm is 0.9, indicating that crossover is applied in 90% of the mating operations.

3.5. Mutation

Mutation was used to introduce variations into the population by randomly altering the genes and maintaining genetic diversity. Although, akin to real life, mutations are very rare, with a low probability of affecting the population, we still explored two different methods: **binary mutation** and **swap mutation**. Although binary mutation is commonly used with binary representations, nothing can prohibit one from using it on other types of representations if the rest is coherent. In swap mutation, we mutate only rooms to diversify the approach. The mutation rate used in our genetic algorithm is 0.1, indicating that mutation is applied in 10% of the operations.

4. Experimental Setup

We designed an experiment to compare and evaluate the performance of different unique combinations of genetic operators. The main objective was to identify the most effective combinations of crossover, mutation, and selection methods in terms of their impact on the obtained fitness values.

Each unique combination of these operators was tested for 30 runs, with each run consisting of 150 generations. This setup resulted in a total of 2 selection methods * 2 mutation methods * 2 crossover methods = 8 different unique configurations. The performance of each configuration was measured by the average fitness of the population over each equivalent generation for the different runs. We chose not to consider Cycle crossover at this stage, since its application is not as appropriate for our problem, and it generally produced on average worse results when tested as seen in Appendix A.

For the final experimental setup exemplified in Appendix B, the configurations that showed the best performance stabilizing at higher levels of fitness, were ***single_binary_tournament*** and ***uniform_binary_tournament*.** They demonstrated fast convergence and maintained the highest average fitness levels across generations. These configurations reached a near-optimal fitness level quickly and sustained it throughout the generations suggesting that the combination of binary mutation and tournament selection is highly effective.

For the impact of different operators, Single Point Crossover was slightly more effective than Uniform Crossover. Binary mutation generally provided better results compared to Swap mutation and Tournament selection showed faintly better performance compared to Ranking selection in most cases.

The statistical significance of the results is supported by the confidence intervals displayed in Appendix C. The analysis indicates that certain configurations, such as ***single_binary_tournament*** and ***uniform_binary_tournament***, are statistically superior due to higher average fitness levels and narrower confidence intervals. If the confidence intervals of two configurations do not overlap, it indicates a statistically significant difference in their performance. For instance, since the confidence interval of ***single_binary_tournament*** does not overlap with ***single_swap_ranking***, we can infer with confidence that ***single_binary_tournament*** outperforms ***single_swap_ranking*** and the other configurations its confidence interval does not overlap***.***

Additionally, using the same established framework, we tested the impact of different hard and soft constraint penalty values combinations on each unique configuration as evidenced in Appendix D. The lines in each subplot are colored in different shades of green, with darker shades representing higher penalty values and lighter shades representing lower penalty values. For this experiment we considered all combinations of values using ***hard_constraint_penalties*** = [1, 2, 5, 10] and ***soft_constraint_penalties*** = [0.5, 1, 2, 5]. This visualization demonstrates that updating hard and soft penalty values, either to higher or lower values, doesn't have a discernible impact on the fitness for the unique configurations.

5. Results and Discussion

The results obtained from our experimentation aimed at leveraging GAs to optimize the Time Scheduling problem are promising. The best-performing configurations demonstrated rapid convergence to high average fitness levels, indicating effective optimization. Specifically, configurations utilizing Single Point Crossover, Binary Mutation, and Tournament Selection outperformed others, achieving near-optimal fitness quickly and maintaining it across generations.

Additional potential experimentation for our project includes parameter tuning, such as experimenting with different population sizes, numbers of generations, and fine-tuning mutation and crossover rates.

6. Conclusion

The implementation of Genetic Algorithms in solving the University Course Scheduling problem proved to be highly effective. Our experiments revealed that configurations using Single Point Crossover, Binary Mutation, and Tournament Selection achieved the highest fitness levels, indicating efficient optimization. The use of both hard and soft constraints ensured practical and feasible schedules, accommodating all key requirements. Future work can focus on parameter tuning and exploring additional genetic operators to further enhance the optimization process. Overall, the project demonstrates the significant potential of GAs in complex scheduling tasks.

7. Appendix

7.1. Appendix A

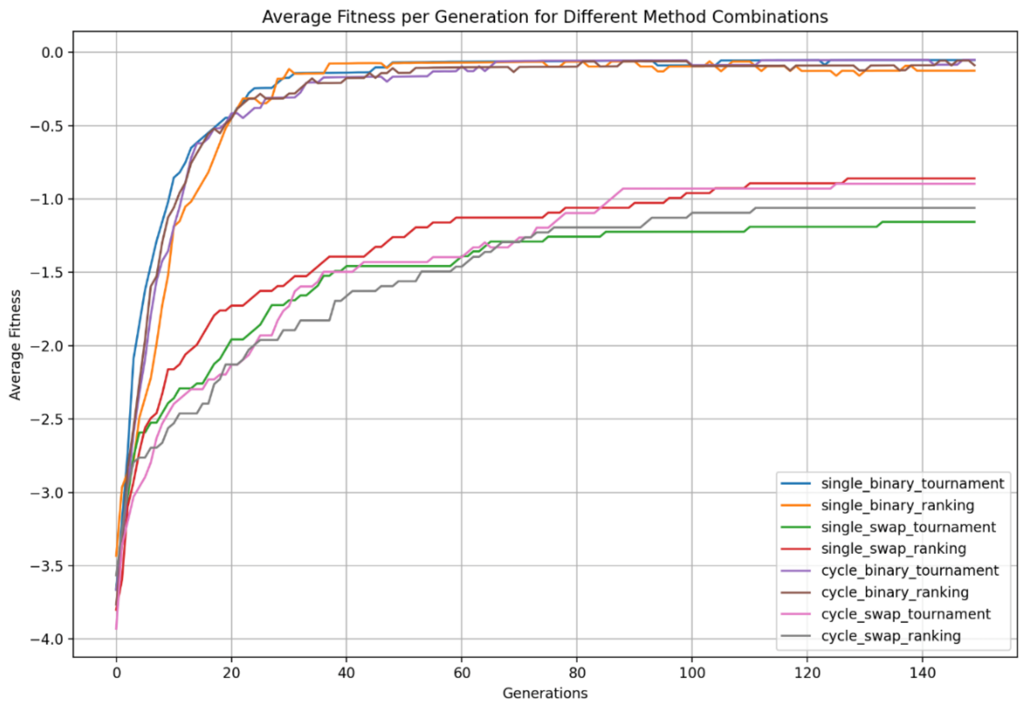


Figure 1 – Average Fitness per Generation for single point and cycle crossover; binary and swap mutations; ranking and tournament selection combinations.

7.2. Appendix B

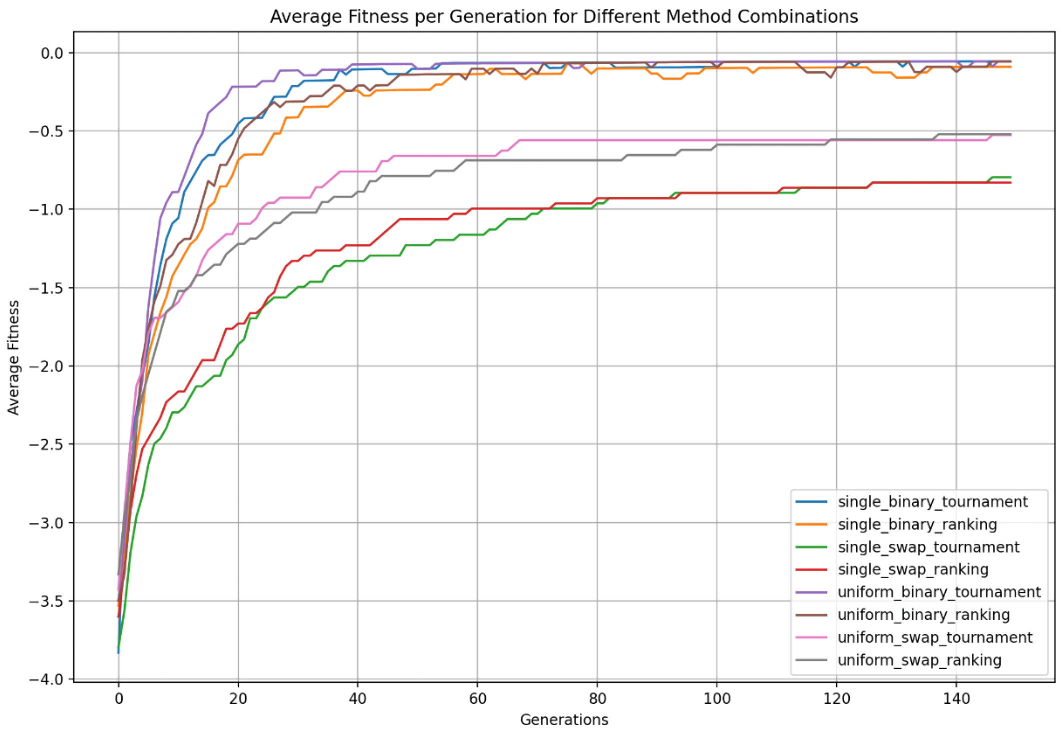


Figure 2 – Average Fitness per Generation for different single point and uniform crossover; binary and swap mutations; ranking and tournament selection combinations.

7.3. Appendix C

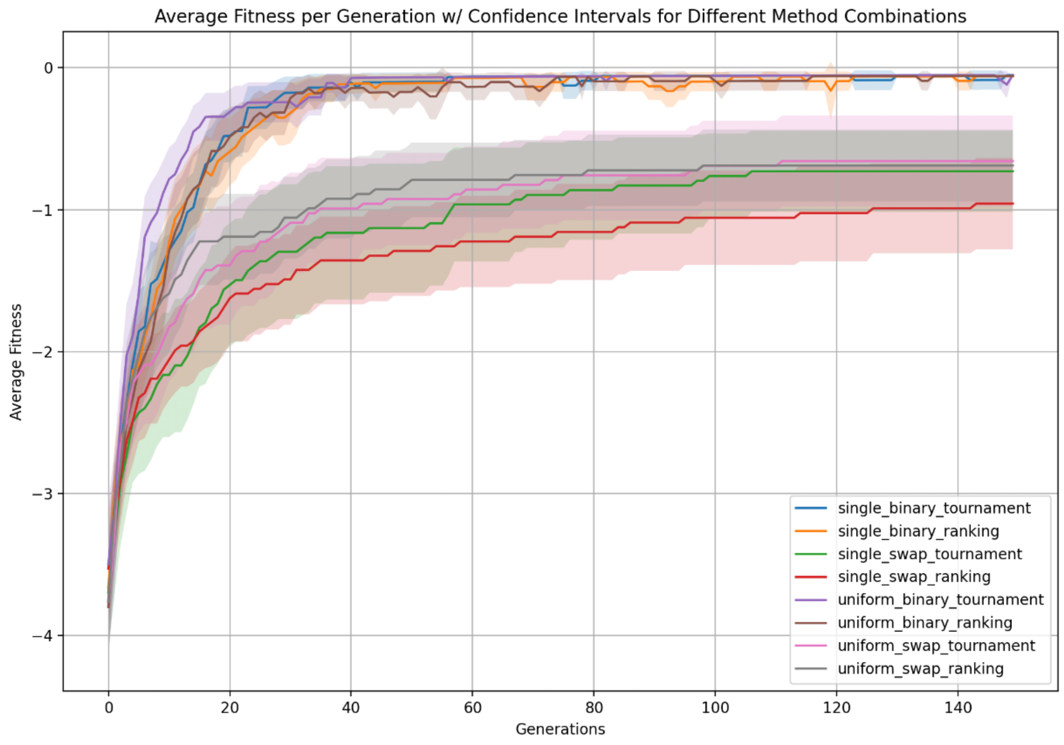


Figure 3 – Average Fitness per Generation for different single point and uniform crossover; binary and swap mutations; ranking and tournament selection combinations with confidence intervals.

7.4. Appendix D

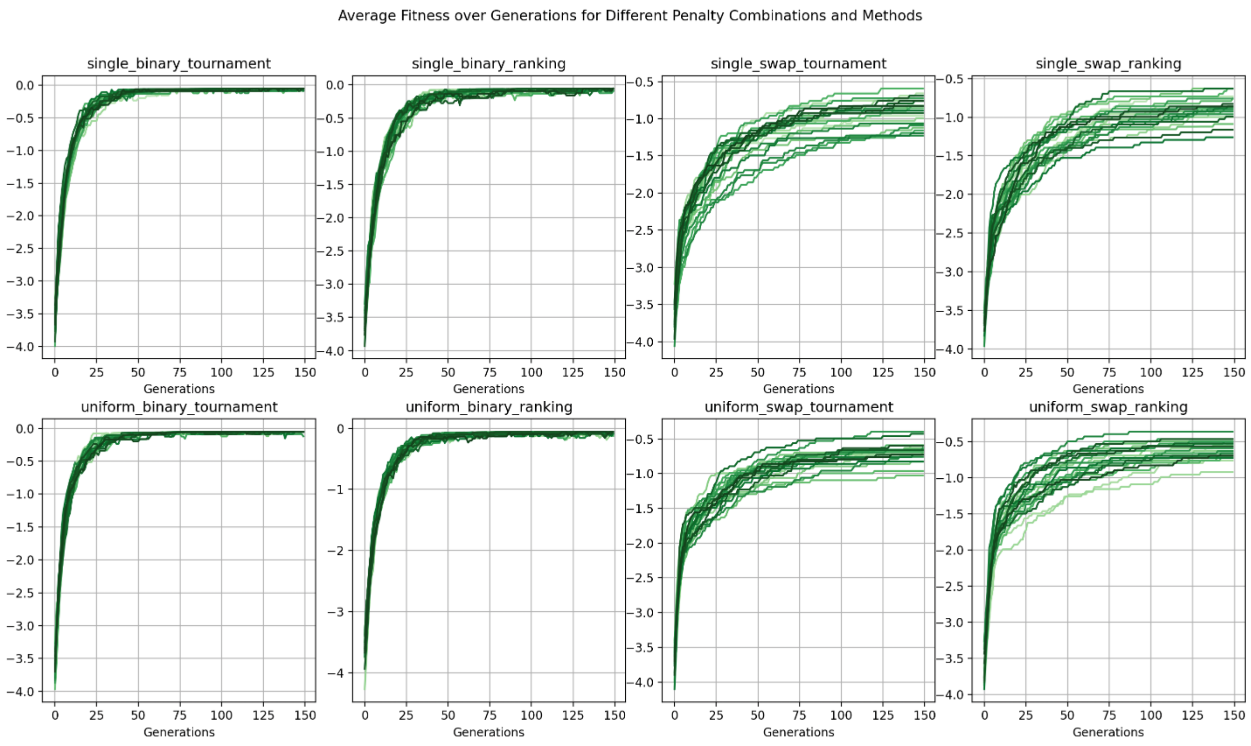


Figure 4 – Average Fitness per Generation for different single point and uniform crossover; binary and swap mutations; ranking and tournament selection combinations with different constraints' penalties (1, 2, 5 and 10 for hard penalties; and 0.5, 1, 2 and 5 for soft penalties).