

Intelligent Systems

Seminar Assignment 1 - Scheduling with Genetic Algorithms

October 2026

1 Introduction

In this seminar assignment, your goal is to use genetic algorithms to optimize a schedule of university lectures. You will be tasked with running suitable genetic algorithms, preparing the data to test your algorithms, and writing a report that will show your work and present the results in a clear and concise manner.

You can do the assignment in groups of 2 or individually. You are allowed to use artificial intelligence or agentic systems for help. If you do, you are required to submit a log of your dialogue with the system. You will also be required to justify and explain your choices during the seminar assignment presentations that will take place during the labs the week following the submission deadline.

2 Task 1 - 30 %

During the first task, you will use a genetic algorithm to create an optimal schedule based on the three input files:

- **Classes**, which contains all the classes that need to be present in the schedule. Each line of the file contains the name of the class, followed by a list describing the duration of each of its lectures. For example, the line `Programming [2, 3, 2]` means that the class named `Programming` has three lectures: the first takes 2 hours, the second takes 3 hours, and the third takes 2 hours. The order of the lectures is not important for scheduling.
- **Classrooms**, which contains the available classrooms and the hours during which the classroom is available. Each line contains the name of the classroom and the available times in the form of `[start time, end time]`. For example, the line `PR1 [8, 16]` means that the classroom `PR1` is available from 8:00 to 16:00. The schedule covers a single day of lectures.
- **Teachers**, which contains the teachers available to teach each class. Each line contains the teacher's name and the classes they can teach. For example, the line `Miha [programming, math]` means that Miha can teach programming and math.

For a schedule to be valid, it needs to satisfy the following conditions:

- Every lecture in the classes file must be present in the schedule.
- Every lecture needs to be assigned to a classroom at a time when that classroom is available. Lectures cannot overlap in the same classroom: a single classroom can hold only a single lecture in a given hour.
- Every lecture needs to be assigned a teacher. Teachers cannot teach more than one lecture at the same time.

To solve the task, you will first need to find a way to represent a schedule. We leave the details of the implementation to you: any representation that can be used to obtain a schedule is acceptable, whether that's arrays, matrices, or more complex python structures. Next, create a suitable fitness function that will reward schedules that satisfy the conditions and penalize those that do not. Finally, use a genetic algorithm to optimize the schedule. During this task, you do not have to return optimal schedule but the GA should still work.

Further, create a function to display the obtained schedule. This can be a simple text representation that shows how lectures are arranged in the schedule or graphical representation (not necessary).

3 Task 2 - 20 %

During this task, you will improve the genetic algorithm so that it produces better solutions. Possible ways to do this include:

- **Generating your own initial population.** By default, PyGAD generates the starting population completely randomly, leading to solutions that break many of the schedule constraints. You can create a better initial population (for example, using a greedy approach) and pass it to pygad using the `initial_population` parameter.
- **Using custom crossover and mutation operations.** Similarly, default PyGAD mutation and crossover operations are not aware of the schedule constraints. Create your own crossover and mutation functions that take the constraints into account. Instructions for user-defined crossover and mutation functions are available here:

https://pygad.readthedocs.io/en/latest/user_defined_operators.html#user-defined-crossover-mutation-and-parent-selection-operators.

Implement the improvements and run a test to determine whether they improve the performance. Use PyGAD lifecycle callbacks (`on_mutation`, `on_crossover`, etc.) to compare how quickly each step of the algorithm executes compared to the one in Task 1 and how quickly the fitness function increases. You will present the results of this comparison in Task 4.

4 Task 3 - 20 %

During this task, you will implement two additional constraints on the scheduling algorithm. Unlike the constraints in Task 1, these are soft constraints: the schedule does not need to satisfy them 100% to be valid, but satisfying them should increase the fitness score. The additional constraints are:

- **Teacher preferences.** Your scheduling program should now consider an additional file `preferences.txt`, which describes when teachers would like to teach their classes. Each line in the file contains the teacher's name followed by a dictionary of preferences in the form:

Miha {"likes" : [start_hour, end_hour], "dislikes" : [start_hour, end_hour]}.

Your algorithm should attempt to follow these preferences by placing each teacher's lectures during hours they like and outside the hours they dislike.

- **Teacher balance.** The schedule should balance the teaching hours between the teachers so that everyone teaches (approximately) the same number of hours. This may not always be possible, but the algorithm should balance the hours as much as possible.

Tackle these two constraints in two ways:

- Using a single fitness function that takes into account both constraints.
- Using multi objective optimization. PyGAD supports this with a few small tweaks to the algorithm described here:

https://pygad.readthedocs.io/en/latest/multi_objective.html.

As in Task 2, evaluate the approach and present the results as described in Task 4.

5 Task 4 - 30 %

In this task, you will evaluate your genetic algorithms and create a report (in the form of a jupyter notebook) that contains your code and the results. To perform a comprehensive evaluation, you should:

- **Create larger input files.** The example files provided on uclnca are too small for a comprehensive evaluation. To properly evaluate your approach, create larger files with more classes, classrooms, and teachers. Try to create files where the conditions are easy to meet as well as files with harder conditions.
- **Compare your approach on different sets of input files.** Compare your approach between small/large and easy/difficult inputs. How long does the algorithm take to find a suitable solution? How fast does the fitness function increase? Visualize the results using graphs.

- **Compare different configurations of genetic algorithms.** Do different parameters of the `pygad.GA` function affect the results? Do custom populations and crossover/mutation functions in Task 2 improve the performance of the algorithm? Visualize the results using graphs.
- **Find examples where the genetic algorithm fails.** Create input files where the algorithm does not find a good solution.

You will present a report during the seminar presentations that will take place during the labs the week after the submission deadline. Your report should include:

- The code of your approach separated by task. Highlight some key segments of the code that you think are especially interesting or important to your approach.
- The dialogue with an agentic system or AI you used to build the solution, obtain the results, and/or write the result.
- The results of each task.
- Comparisons of different algorithms.

Make sure to present these results in a clean and concise manner, using tables and graphs that are easy to read and understand. The presentations will last about 5 minutes, so make sure your report is concise enough to make that possible.