



Department of Electrical & Computer Engineering  
ENCS3340 - Artificial Intelligence

## Department Course Distribution

**Prepared by:** Mohammad Abu-Shelbaia

**Instructor:** Dr. Aziz Qaroush

**Date:** January 13, 2023

# **Abstract**

The aim of this project is to practice using local search methods in real-life scenarios, as for the outcome of this project is to find an optimized solution for department course distribution, as for the search method its going to be a genetic algorithm to optimize the courses for our Department of Electrical & Computer Engineering.

# Contents

<b>1</b>	<b>Genetic Algorithm</b>	<b>1</b>
1.1	Introduction and Formalization . . . . .	1
1.2	Fitness . . . . .	2
1.3	Selection . . . . .	3
1.4	Crossover . . . . .	3
1.5	Mutation . . . . .	4
1.6	Running . . . . .	4
<b>2</b>	<b>Results</b>	<b>6</b>
<b>3</b>	<b>Conclusion</b>	<b>7</b>

## List of Figures

1.1	Chromosome Representation . . . . .	1
1.2	Crossover . . . . .	3
1.3	Algorithm Run . . . . .	5
2.1	Sample Run . . . . .	6

# 1 Genetic Algorithm

## 1.1 Introduction and Formalization

The genetic algorithm being proposed is designed to solve a scheduling problem for a set of 56 courses, with each course represented as a gene in the population. Each gene's value is a list of sections, and each section has a lecturer and a time slot. The problem is formally defined as: find the optimal schedule for the 56 courses such that each course is assigned a lecturer and a time slot, subject to constraints such as lecturer availability and time slot availability. The algorithm will be implemented with the following parameters: a population size of  $n$ , a number of generations of  $k$ , a crossover rate of 1, and a mutation rate of 0.3. The initial population will be randomly generated, with each gene representing a course and its value being a list of sections with randomly assigned lecturers and time slots. The fitness of each individual will be evaluated according to a predetermined fitness function that takes into account the constraints and optimize for certain criteria. Individuals will be selected for mating based on their fitness scores, with higher-scoring individuals being more likely to be chosen. Crossover and mutation will be applied to create new offspring, and these offspring will be evaluated for fitness. The weakest individuals in the population will be replaced by the new offspring, and this process will be repeated for a specified number of generations. The best individual(s) will be returned as the solution to the problem.

Each course is either lab or lecture, labs are taken once a week, while lectures are taken twice a week. the lectures days are taken on pairs either (M, W), (S, W), (S, M), or (T, R). labs slots are available three times a day at 8:30, 11:25, or 2:15, and each of them has a duration of 2:40 hours, lecture slots are available four times a day at 8:30, 10:00, 11:25, or 12:40 with a duration of 1:15 hours.

The reason we choose this approach instated of having the time slots as the genes, and the sections of each course is a value is that this approach makes working with data easier and more accessible since the sections of the same course are listed together instated of on a separate genes, hence evaluating the fitness would be much easier and faster.

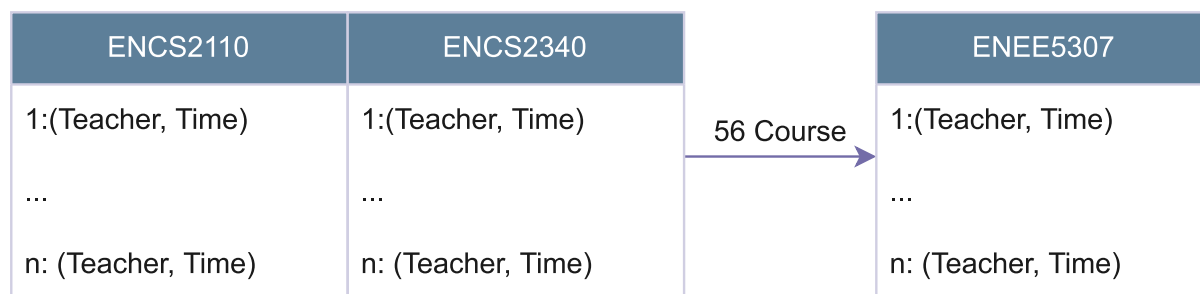


Figure 1.1: Chromosome Representation

## 1.2 Fitness

As for the most important part of the genetic algorithm that defines the whole process, we first have to define what a diverse schedule is. first, we are going to define our preferences, as a student, we generally don't like coming on Saturdays, late labs, and early labs/lectures. And I rather have a late late lab rather than coming on Saturday. and I'd rather have an early lab/lecture than have a late lab. and according to this information we set the weight of each plenty, and we can change these weights to change the direction of the solution according to our preferences.

As for the course fitness, it is represented as the number of days the sections of the course covers, the number of time slots it covers, and how even the sections are distributed between (M, W) and (T, R). And, as for the fitness of the whole schedule, it takes into account the number of courses taken in the morning, the number of late labs, the number of courses taken on Saturdays, and the most important thing which is conflicts between the same year class courses.

Additionally there is two criteria that fail the schedule, having two or more lab section on the same time since labs are limited to one room, and having the same teacher teaching more than one section at the same time slot.

Further explanation on how we can drive the algorithm in the direction we want, if we want to have no lectures on Saturday we can by giving Saturday labs respectively higher plenty, in the other hand, the number of conflicts will rise instated of falling since all the courses are cramped on four days instated of five, which might not give the students the space required to sign-up for their courses.

Finally, our most important variable is the number of conflicts, and by testing we found that this value won't go under 33. so we are going to optimize our values to have a conflict range of around 33, and have as less as possible Saturdays slots, early slots, and late labs.

### 1.3 Selection

As for selection method, we had many options but the roulette wheel and the ranked selection method seemed more reasonable in our case, but we settled for the ranked selection since our fitness values are close to each other which forces the roulette wheel to give an even fitness proportion across the different chromosome while the ranked selection will always give the highest fitness a highest priority.

Further explanation, our implementation of the ranked method give each chromosome a sequential rank based on their fitness, meaning that the lowest fitness will get a rank of 1, while the highest fitness will get a rank of  $n$ , the probability evaluation will be the rank divided by the sum of all ranks.

$$P_i = \frac{R_i}{\sum_{k=1}^n k} \quad (1)$$

According to the equation above, if we have a population of size five, with fitness's accordingly are 100, 101, 102, 103, and 104, with roulette wheel the probability's accordingly is 0.196, 0.198, 0.2, 0.201, and 0.203. But if we use our method the probability's accordingly are 0.066, 0.133, 0.2, 0.266, and 0.333, which as we can see favors the chromosome's with higher priority and its evenly distributed when the fitness values are close.

according to the probability defined above we select two chromosome we will call them parents, and then apply crossover and mutation to generate a new offspring.

### 1.4 Crossover

As for crossover, after we choose the parents, we implemented double point crossover method, which takes 2 random points of the chromosome and exchange the genes between the two parents between these two points.

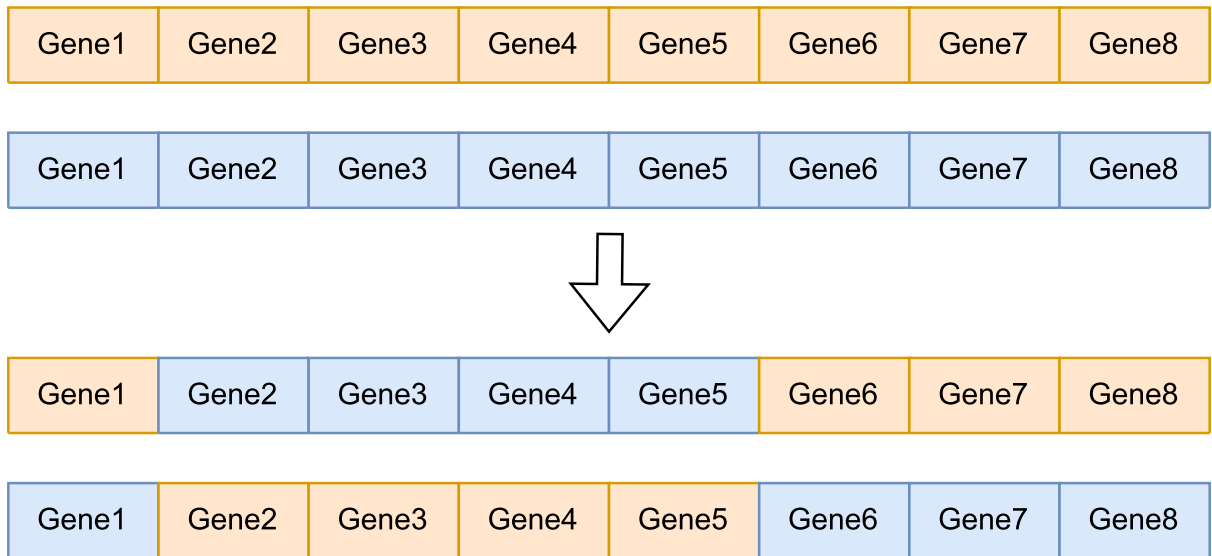


Figure 1.2: Crossover

As we can see that the double crossover method is better than the single crossover method since it takes two points, these 2 points can be at random points which gives more options, e.g., if we had first point at  $n/2$  and the last at  $n$  it would be the same as implementing a single crossover at  $n/2$ , but it could expand more and have for example the two points at 3 and  $n-3$  which will exchange more genes at random intervals.

Additionally, if our crossover is not valid we consider the parents as the new offspring and continue to mutate the parents.

## 1.5 Mutation

After testing with our project, we can safely assume that the mutation is one of the most important parts in the genetic algorithm, since it expands the versatility the algorithm looks in by doing some random changes, but in the other hand, mutation might as well fail the schedule.

Hence, we gave the mutation a rate of 0.3, and gave it 5 tries to mutate without failing the schedule. As for the method we selected at least 10 random courses, in each course we selected random sections and for each section we assigned a random time from the available time slots. By testing we noticed that the algorithm had a big jump in finding the optimal solution after the implementation of mutation. Additionally after the 5 tries if they still return an invalid schedule we return the original chromosome before mutation.

## 1.6 Running

To run our algorithm we initialize the population with  $k$  random chromosomes, then take the best  $n$  chromosomes, select two parents from the new population, perform crossover; if the crossover is valid we mutate the offspring else we mutate the parents, then for mutation as its mentioned earlier if after 5 tries it returns invalid schedule we return the original schedule without mutation and add it to the population. we repeat the process of selection, crossover, and mutation for  $(\text{len}(\text{population})//2)$  for each generation, and we run for  $k$  generations.



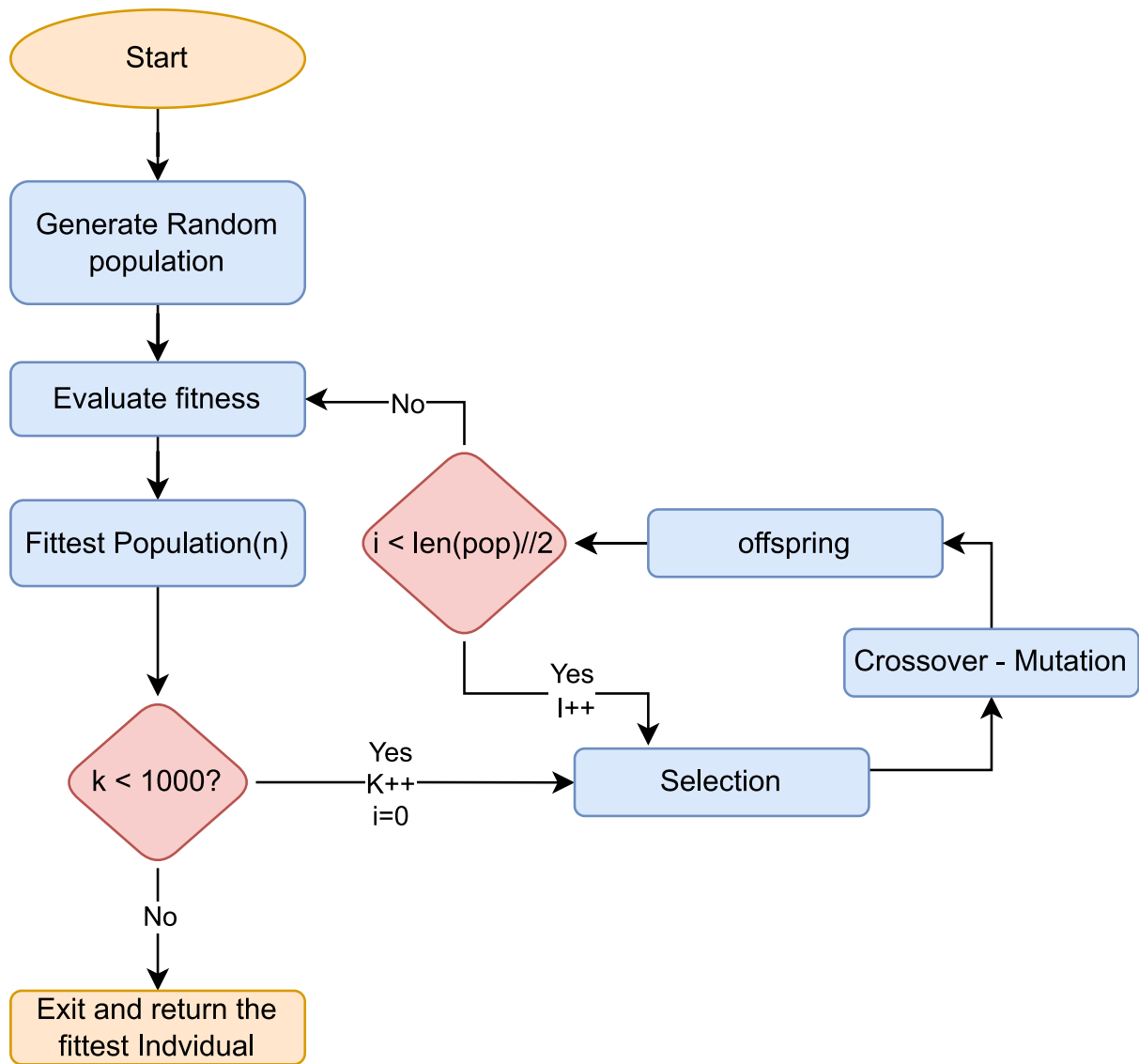
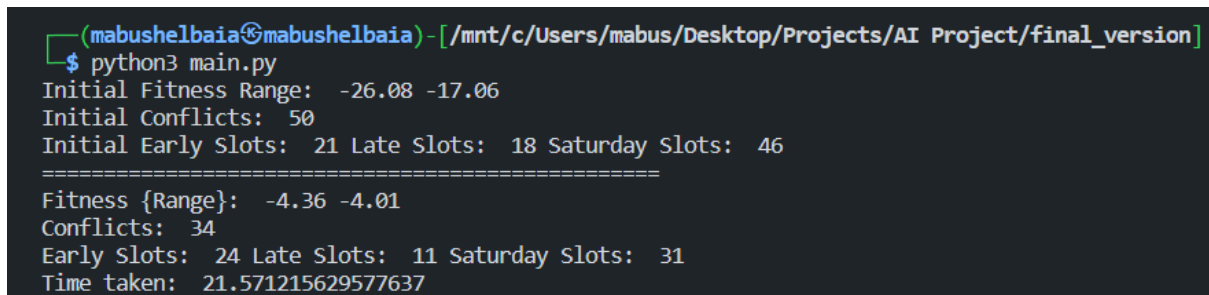


Figure 1.3: Algorithm Run

## 2 Results

Based on our preferences discussed on the fitness part, we set a high penalty for Saturdays, high penalty for conflicts, medium penalty for late labs, and low penalty for early slots, further more we set a condition that the number of courses taken in pair is even between sections, and we got an improvement from 50 conflicts to 34, reduced late labs from 18 to 11, and reduced Saturday slots from 46 to 31, which improved the fitness from -17.06 to -4.01 all that in just 20 seconds, as shown in the figure below.



```
(mabushelbaia@mabushelbaia) - [ /mnt/c/Users/mabus/Desktop/Projects/AI Project/final_version ]
$ python3 main.py
Initial Fitness Range: -26.08 -17.06
Initial Conflicts: 50
Initial Early Slots: 21 Late Slots: 18 Saturday Slots: 46
=====
Fitness {Range}: -4.36 -4.01
Conflicts: 34
Early Slots: 24 Late Slots: 11 Saturday Slots: 31
Time taken: 21.571215629577637
```

Figure 2.1: Sample Run

PS: This is not a demo its just to demonstrate the results we got the full demo and codes are listed on GitHub.

### 3 Conclusion

In conclusion, genetic algorithms are a powerful optimization technique that can be applied to a wide range of problems, including scheduling. Genetic algorithms are able to efficiently explore the solution space of a problem and find near-optimal solutions. One key aspect of genetic algorithms is the use of mutation, which introduces random changes to the solution candidates. By carefully tuning the mutation rate and method, the genetic algorithm can effectively balance exploration and exploitation, leading to faster convergence to the optimal solution. Overall, the genetic algorithm approach is an efficient method for solving scheduling problems and other optimization challenges.