

Project Report

Group NO.: 20

Group Member: Qing Mu, Wen Sun

Topic: Genetic Algorithm Course Scheduling

Problem Statement:

When scheduling for huge number of courses, it could be a problem since we need to take many requirements into consideration. For instance, the classroom need to have enough seats for students, and two course classes cannot take place in the same room at the same time.

Here we decided to use genetic algorithm to solve the problem. Under certain model, we only need to supplement the method of measuring the fitness of an individual when a new requirement appears, with which we can generate schedule in right direction.

Implementation:

Objects:

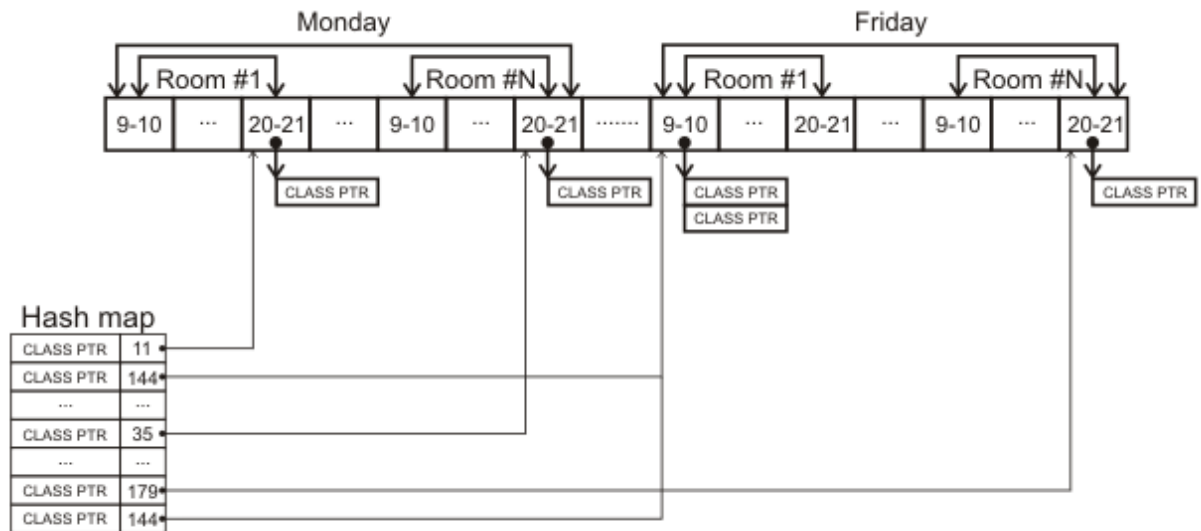
Course Class: Each course class is attended by a certain number of students. It contains a class ID, the duration in hours, and the number of the students.

Classroom: A classroom is a place where a course class takes place. It contains a room ID, and the total seats of the room.

Genes:

Timeslot id, which represent when and where the course would take place.

Chromosome(all possible gene type):



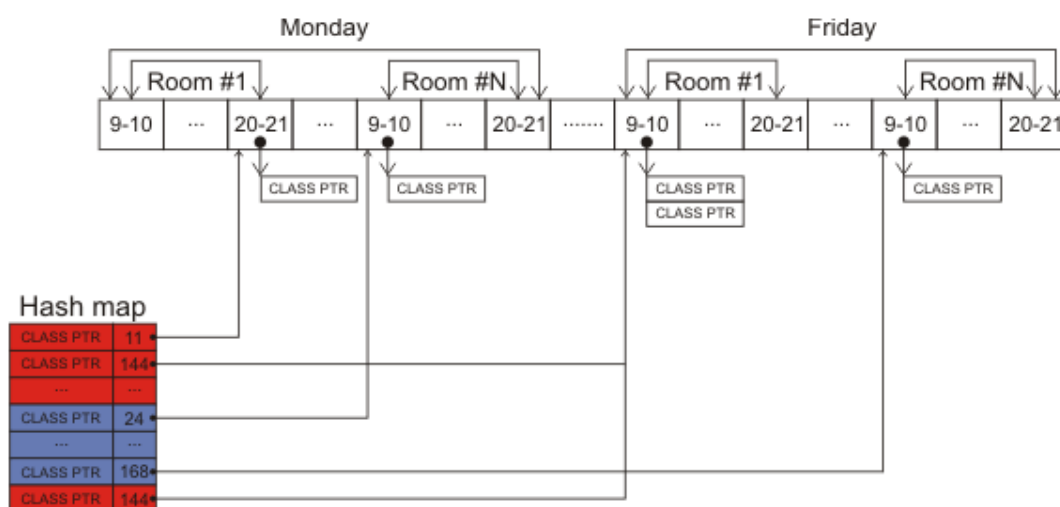
It is a list which contains all the time slots (splitted down to hours) of all the classrooms sequentially. Each element of the list contains one class and one classroom. Assume that courses are only arranged between 8 am and 9 pm, from Monday to Friday.

A hashmap is used to record each course class in a schedule. The key is class ID, and the value is the first time slot that this course class begins. It is used to calculate the fitness score.

A chromosome would contain the number of the crossover points, which is used in the crossover process.

A chromosome also contains a number that indicates the mutation probability. A mutation is randomly occurred after the offspring is generated driven by the mutation probability.

Individual:



We use a hashmap to represent an individual. The key is the course ID while the value (or say gene type) is the timeslot in which it would start.

Fitness:

A Fitness Method gives every schedule candidate a score according to the constraints we add to the problem, to find the candidates with highest scores. In each generation, only 10% percent of candidates will be remained as the parents of the next generation.

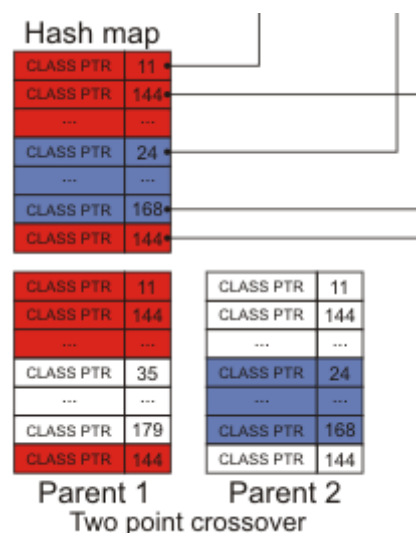
Each course class has a fitness point from 0 to 3:

- If a course class uses a spare classroom, we add 1.
- If a course class uses a classroom with enough seats, we add 1.
- If a course class is in the right time(not cross day), we add 1.

Then we accumulate the class points of the schedule candidate.

- Fitness score = schedule score / the highest possible score

Crossover:



In each generation, only 10% percent of candidates with highest fitness scores will be remained as the parents of the next generation.

When crossing over, we design three sources of randomness. 1)randomly chosen parents from the best 10% of the last generation.2) randomly generated crossover points when crossing over. 3) Mutation (under certain mutation probability).

Mutation:

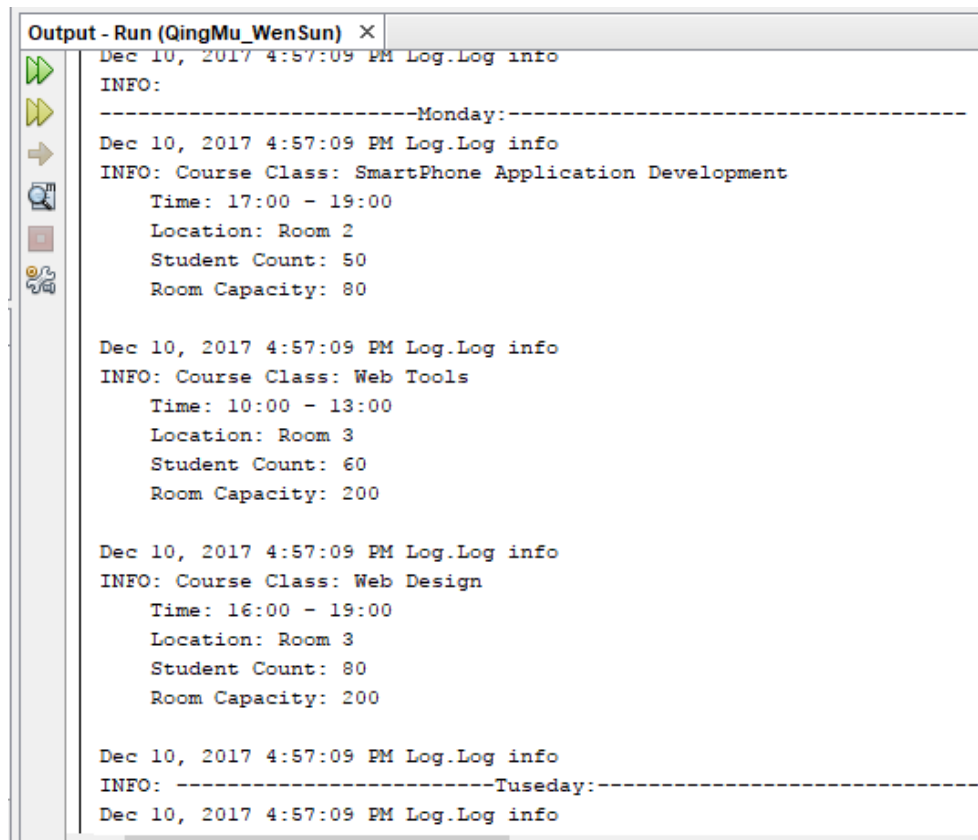
After each offspring is generated by crossover, we randomly mutate the offspring by randomly pick one course classes in its schedule and put it to another random time slot of a random classroom.

Result:

Unit test:

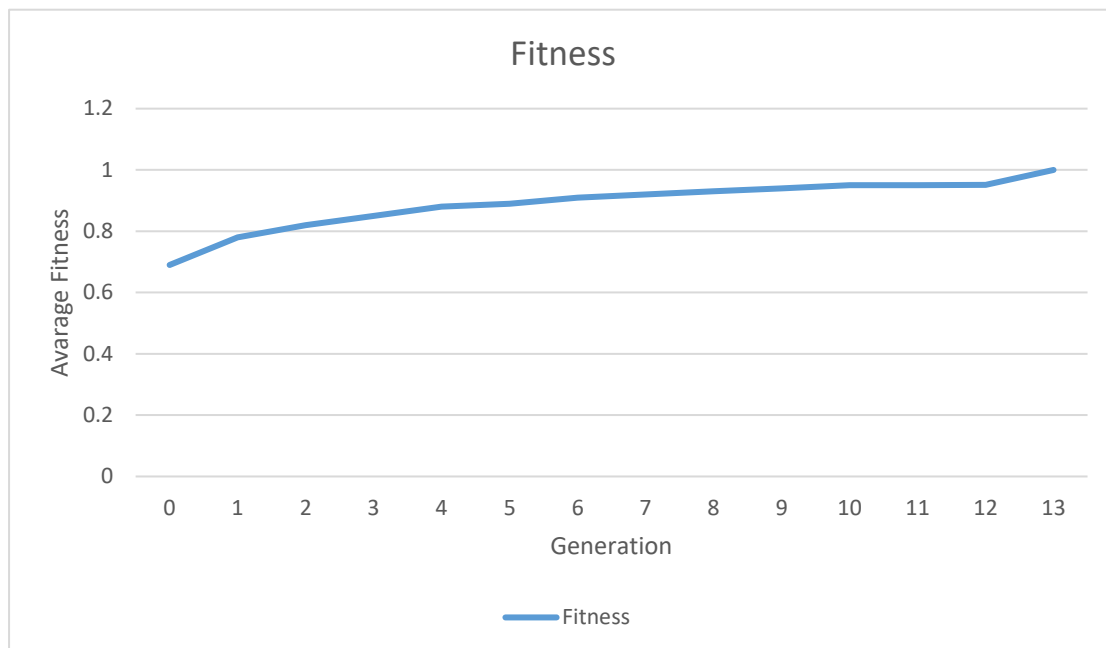
```
-----  
T E S T S  
-----  
Running GeneticAlgorithm.AppTest  
Tests run: 1, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.063 sec  
Running GeneticAlgorithm.FitnessTest  
Tests run: 4, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.031 sec  
Running GeneticAlgorithm.GenerationTest  
Tests run: 1, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0 sec  
  
Results :  
  
Tests run: 6, Failures: 0, Errors: 0, Skipped: 0
```

Application output: (more detail to be seen in log4j file)



```
Output - Run (QingMu_WenSun) X  
Dec 10, 2017 4:57:09 PM Log.Log info  
INFO:  
-----Monday:-----  
Dec 10, 2017 4:57:09 PM Log.Log info  
INFO: Course Class: SmartPhone Application Development  
Time: 17:00 - 19:00  
Location: Room 2  
Student Count: 50  
Room Capacity: 80  
  
Dec 10, 2017 4:57:09 PM Log.Log info  
INFO: Course Class: Web Tools  
Time: 10:00 - 13:00  
Location: Room 3  
Student Count: 60  
Room Capacity: 200  
  
Dec 10, 2017 4:57:09 PM Log.Log info  
INFO: Course Class: Web Design  
Time: 16:00 - 19:00  
Location: Room 3  
Student Count: 80  
Room Capacity: 200  
  
Dec 10, 2017 4:57:09 PM Log.Log info  
INFO: -----Tuesday:-----  
Dec 10, 2017 4:57:09 PM Log.Log info
```

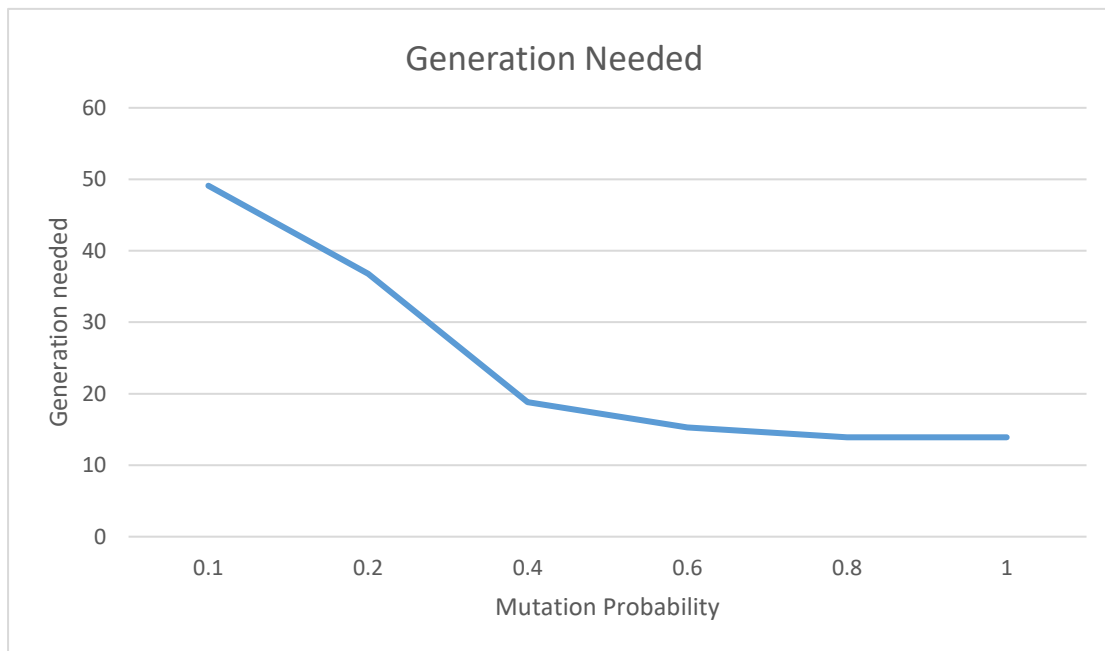
When there are 20 courses to be scheduled in 3 rooms, the average fitness of each generation developed gradually.



Conclusion:

The main source of a feasible answer is randomness when generating new generation since we can hardly get the right schedule from the randomly generated schedule.

The main source of improvement is mutation, so we run 10 tests over some mutation probability and take the average of the test results. In the test, we set number of classroom to be 3 and there are 20 courses to be scheduled in the weekdays.



When we don't mutate at all, we will run into dead loop and never get a right answer. As has been shown above, between 0.1 and 0.4, when we greater the chance to mutate, we are able to get our answer in a quicker way. However, after 0.4, the mutation probability do not make much difference.

We conclude that when the mutation probability is great than 0.4, we can acquire genes needed for a right answer in a short time, but it takes few more generations to acquire the advantageous combination of the genes.

After some discussion, we think there are two other variables that might limit the performance of the algorithm.

1. Crossover points: more crossover points may help us to crossover two well-performed parents in a better way or say to let the new generation to acquire the advantages combinations in a quicker way.
2. Generation number: Create more individuals in the generation basically means more independent tests so that we would have bigger chance to acquire genotype we needed in one generation.

Reference:

<https://www.codeproject.com/Articles/23111/Making-a-Class-Schedule-Using-a-Genetic-Algorithm#Introduction0>