

Program Structure & Algorithms
INFO 6205 Spring 2019- Section 03
Project- Class Scheduling Problem

[GitHub:](#)

https://github.com/janhavibhalerao/INFO6205_511

Team Details: Janhavi Bhalerao(001494855), Ankit Awadhiya(001449209)

Problem Description:

- We have developed a genetic algorithm that schedules classes for a university timetable.
- This is a variant of constraint satisfaction problem. In these type of problems, we have a set of variables that need to be assigned in a way that they avoid violating a set of constraints.
- While scheduling a timetable we have to consider many constraints. Like the classes should not clash or one professor cannot be in two places at the same time.

Implementation Details:

Genetic code:

One chromosome is a collection of genes. Each class has 1 gene for timeslot, 1 gene for professor and 1 gene for classroom. These genes are represented by integers ids. The population consists of all individuals which are candidate solutions. Most randomization is on genes to ensure maximum solution space coverage.

Structure of chromosome:

Class 1	2	Timeslot Id
	4	Classroom Id
	1	Professor Id
Class 2	3	Timeslot Id
	5	Classroom Id
	2	Professor Id

Helper Classes:

We initialize the collection of candidate solutions which form the population in GeneticAlgorithm class. Have used dummy values for implementation purposes and following classes:

- Class
- Classroom
- Course
- Professor
- Timeslot

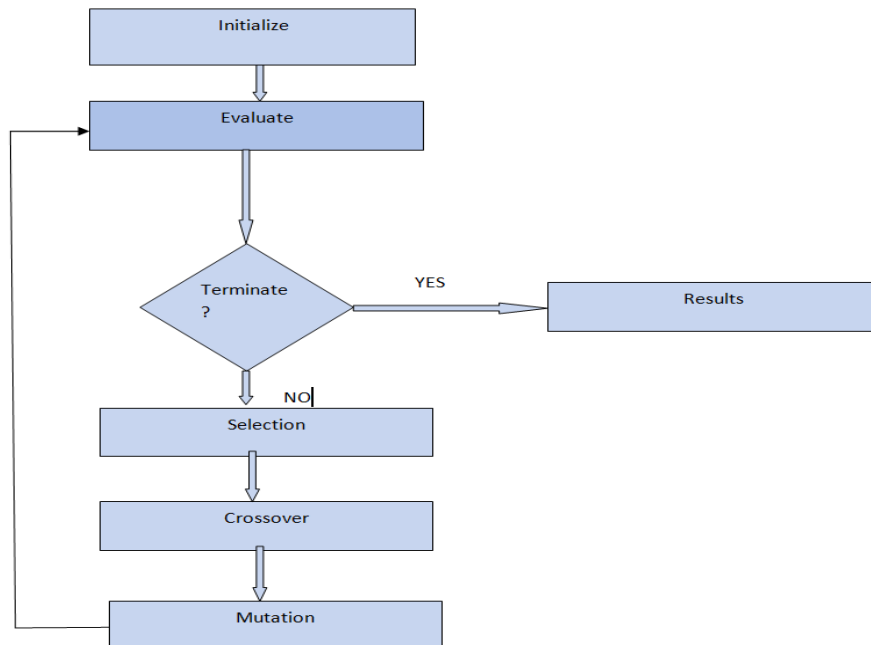
Primary constraints:

- Professors can only be in one classroom at any given time.
- Classrooms can only host one class at a time and the class must be in a free classroom.
- Classrooms need to have a suitable capacity to host the class.

Evaluation Goal:

Optimize our timetable in such a way we satisfy as many constraints.

Genetic Algorithm:



Key Details:

In Schedule class:

- This class knows all of the available rooms, timeslots, etc
- It also reads a chromosome and evaluate fitness.

createClasses()- accepts individual, checks courses and students groups to be give timetables, creates Class object for them. After this,t assigns timeslot, room and professors to them.

It uses genetic algorithm and evaluates different combinations of timeslot, room and professors.

calcClashes()- checks each class and returns the number of clashes. All constraint violations are counted as a clash. If any constraints are violated then there is a clash and a count is kept of it. Eg- If there is a conflict while scheduling classes or capacity of room is too small.

In ScheduleAlgo Class:

We have initialized the population in this class with the population size, mutation rate, crossover rate and tournament size.

Fitness Evaluation:

- After population has been initialized we evaluate population and assign fitness to each individuals. We try to create a schedule ensuring that there are 0 clashes.
- We use createClass() and calcClashes() here. The schedule object is cloned here, createClasses() is called and no. of clashes are calculate via calcClashes().

0 clashes = fitness value of 1

- calcPopulation() iterates through population and calculates fitness for each.

Terminating conditions:

- Maximum number of generations reached
- If we find individuals with good fitness

Selection:

- We have used tournament selection which works by pre-decided individuals from population_randomly. We then compare chosen individuals fitness to find the best.
- Individual with highest fitness values wins tournament and is returned as selected individuals.
- We have picked tournament size as 5. Individual with high fitness has better rate of survival.

Crossover:

We have done uniform crossover, each gene has 50 % chance of coming from parent 1 or parent 2.

Elitism:

- We have tried to improve Genetic Algorithm performance by using the concept of elitism. This will ensure we don't end up losing best individuals and will transfer best individuals of current generation to the next generation.
- We have given an elitism count of 2. If this value is kept too high, it will slow down genetic algorithms search process due to preserving too many individuals(This causes lack of genetic diversity).
- Crossover is only applied if both crossover conditions are met and individual is not elite.

Mutation:

- Our genotype consists of classroom, professor id and timeslot, we cannot choose a random number for mutation. We create a new random individual and run uniform crossover to implement mutation.
- Using constructor of Individual class to create a new random individual and select gene from this individual to copy into individual to be mutated. This process ensures we are always selecting valid genes.

Parallel Processing:

- We have improved the performance of this genetic algorithm by distributing the processing requirement as the fitness function can be computationally expensive.
- Multiple fitness functions can be processed simultaneously via parallel processing. This will have a huge impact when there are hundreds of individuals to be evaluated for each population.
- We have used IntStream to implement parallel processing. It creates optimal number of threads.
- In our genetic algorithm, calFitness() clones Timetable object before using it to ensure that objects in one thread will not affect objects in another thread during parallel processing. Hence, since we have cloned the object, changes in one Timetable object will not affect other threads.

Observations & Conclusions:

With the scheduling problem, there may not be a solution always unlike other problems. Eg- If there are clashes in classes then it is not an acceptable solution.

Fitness Function:

- The first generation results in very low fitness. The timetable is not right at this point.
- However, with increasing number of generations the fitness gets better. The algorithm starts giving constant results around 100 generations.

Output:

Generation No.62 Best fitness: -29.1

Generation No.67 Best fitness: 3.61

Generation No.73 Best fitness: 4.19

Generation No.78 Best fitness: 5.06

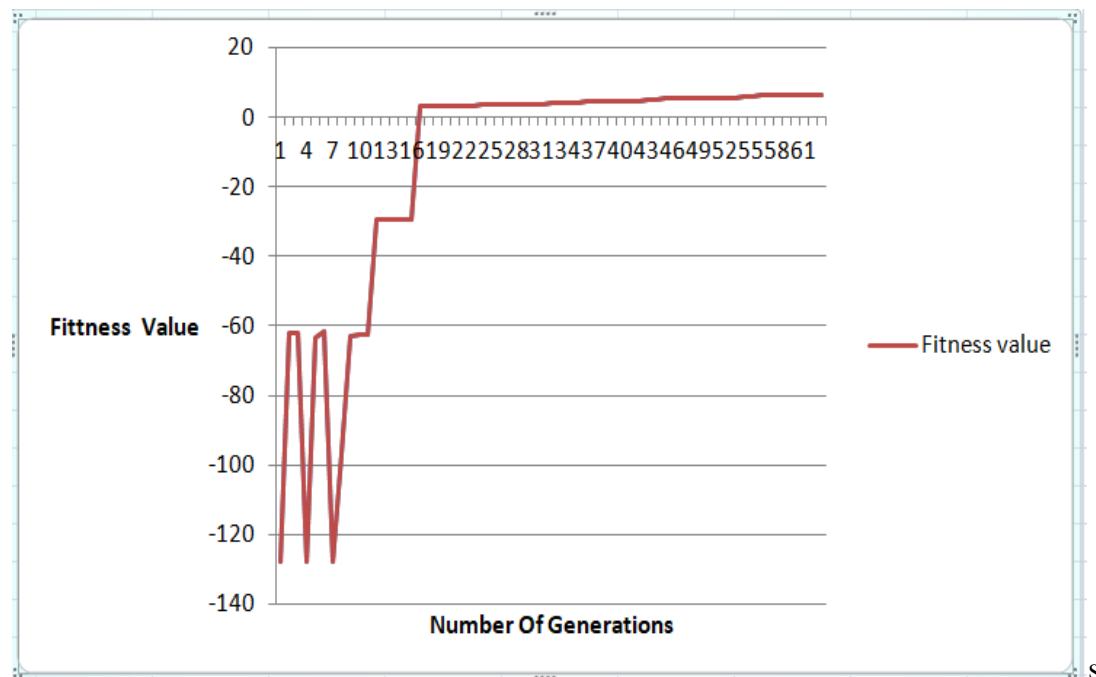
Generation No.88 Best fitness: 5.64

Generation No.91 Best fitness: 5.93

Generation No.100 Best fitness: 6.22

Graph:

Y-axis: Fitness Value; X-axis: Generations



Rate of mutation:

- High rate of mutation allows genetic diversity and may help avoid local optimums.
- But, if the rate is too high then there maybe too much genetic variation. This may result in loss of good solutions found in previous population.
- Hence, we choose a mutation rate of 0.01 which is enough diversity and wont cause algorithm to loose genetic information from previous population.

Population size:

- If the population is too small , the computation is faster but we get less desirable solutions.
- With a large population size we get a bigger search space and algorithm can sample more area. This results in an accurate and optimal solution.
- Hence we have chosen a population size of 1000.

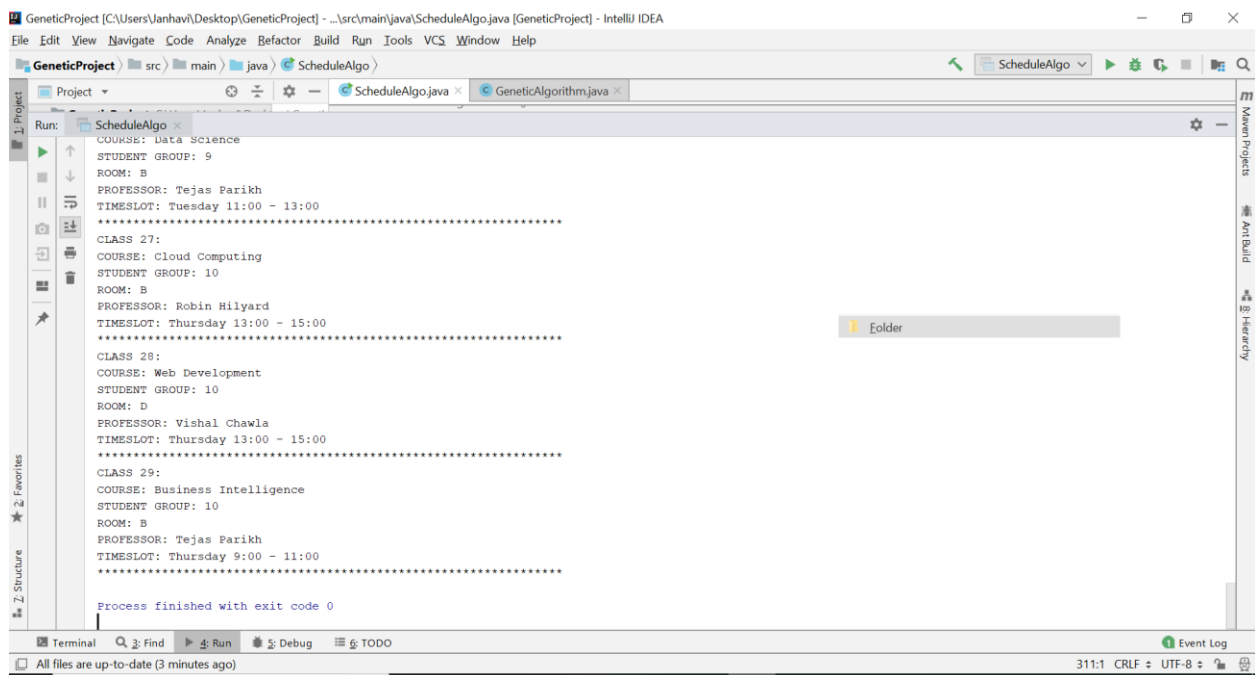
Crossover rate:

With a high crossover rate , we get completely new results altogether in population. If the crossover rate is too low, we may not reach a fit solution. So we have chosen a crossover rate of 0.9

Evidence of running:



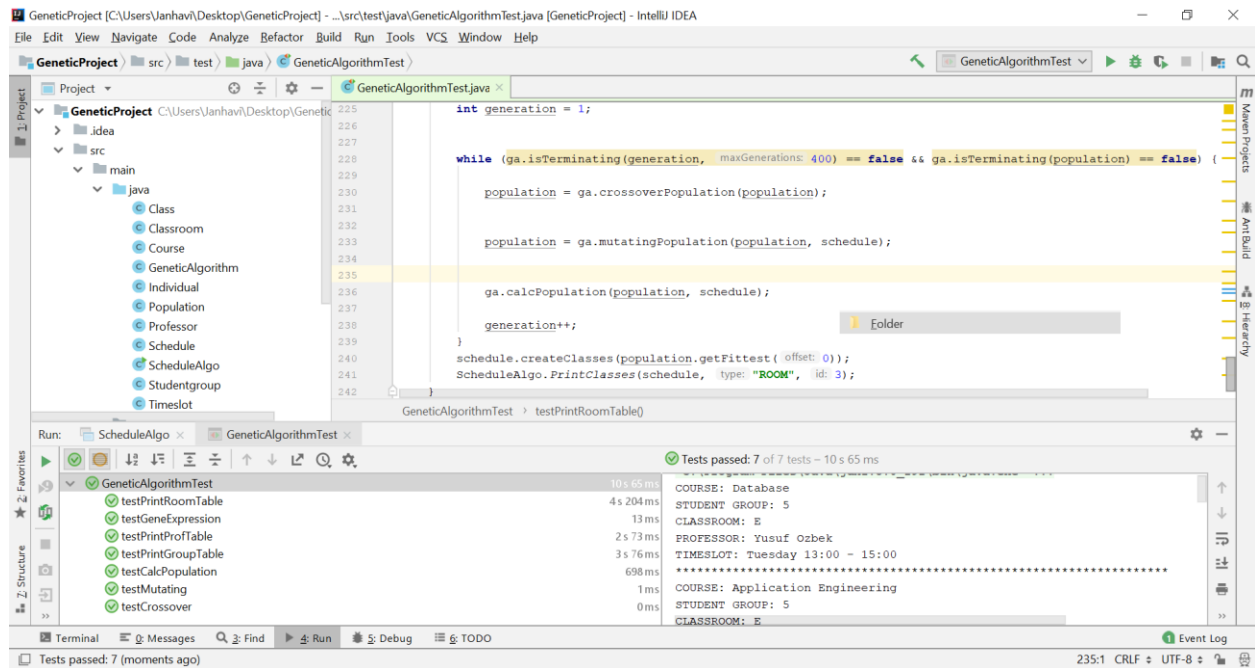
Output.txt



```
GeneticProject [C:\Users\Vanhav\Desktop\GeneticProject] - ...src\main\java\ScheduleAlgo.java [GeneticProject] - IntelliJ IDEA
File Edit View Navigate Code Analyze Refactor Build Run Tools VCS Window Help
GeneticProject src main java ScheduleAlgo GeneticAlgorithm.java
Run: ScheduleAlgo
COURSE: Data Science
STUDENT GROUP: 9
ROOM: B
PROFESSOR: Tejas Parikh
TIMESLOT: Tuesday 11:00 - 13:00
*****
CLASS 27:
COURSE: Cloud Computing
STUDENT GROUP: 10
ROOM: B
PROFESSOR: Robin Hilyard
TIMESLOT: Thursday 13:00 - 15:00
*****
CLASS 28:
COURSE: Web Development
STUDENT GROUP: 10
ROOM: D
PROFESSOR: Vishal Chawla
TIMESLOT: Thursday 13:00 - 15:00
*****
CLASS 29:
COURSE: Business Intelligence
STUDENT GROUP: 10
ROOM: B
PROFESSOR: Tejas Parikh
TIMESLOT: Thursday 9:00 - 11:00
*****
Process finished with exit code 0
Terminal Find Run Debug TODO
All files are up-to-date (3 minutes ago) 311:1 CRLF UTF-8
```

Unit Tests:

We have used maven and Junit Test for unit testing and running the test cases.



References:

<https://towardsdatascience.com/introduction-to-genetic-algorithms-including-example-code-e396e98d8bf3>

<http://www.theprojectspot.com/tutorial-post/creating-a-genetic-algorithm-for-beginners/3>