# Intelligent Time-Table Preparation


## Artificial Intelligence

INT (404)

Submitted By:- K18FG

Himansu Behera- A17

Aarya Raj- A18

Deepak Pandey- A19

Jampana Rajiv K Raju- A20

## Agenda

# CODE In Python

```
import prettytable as prettytable
import random as rnd
POPULATION_SIZE = 9
NUMB_OF_ELITE_SCHEDULES = 1
TOURNAMENT_SELECTION_SIZE = 3
MUTATION_RATE = 0.1


class Data:
    #Here we declare the classroom number and the maximum student capacity of the
students
    ROOMS = [["33-501",70],["34-203",35]]

    #Here we declare the meeting times, in which a class can hold, here MT1, MT2 means
meeting 1, meeting 2 and so on. MWF, TTH etc. means Monday, Wednesday, Friday
    MEETING_TIMES = [["MT1", "MWF 09:00 - 10:00"],
                     ["MT2", "MWF 10:00 - 11:00"],
                     ["MT3", "MWF 11:00 - 12:00"],
                     ["MT4", "MWF 13:00 - 14:00"],
                     ["MT5", "MWF 14:00 - 15:00"],
                     ["MT6", "MWF 15:00 - 16:00"],
                     ["MT7", "MWF 16:00 - 17:00"],
                     ["MT8", "TTH 09:00 - 10:00"],
                     ["MT9", "TTH 10:00 - 11:00"],
                     ["MT10", "TTH 11:00 - 12:00"],
                     ["MT11", "TTH 13:00 - 14:00"],
                     ["MT12", "TTH 14:00 - 15:00"],
                     ["MT13", "TTH 15:00 - 16:00"],
                     ["MT14", "TTH 16:00 - 17:00"]]

    #Here we declare the professors name
    INSTRUCTORS = [["AP1"],
                   ["AP2"],
                   ["AP3"],
                   ["AP4"],
                   ["AP5"],
                   ["AP6"],
                   ["AP7"],
                   ["AP8"]]

    # Defining Constructor
```

```
    def __init__(self):
        self._rooms = []; self._meetingTimes = []; self._instructors = []

        for i in range(0, len(self.ROOMS)):
            self._rooms.append(Room(self.ROOMS[i][0], self.ROOMS[i][1]))

        for i in range(0, len(self.MEETING_TIMES)):
            self._meetingTimes.append(MeetingTime(self.MEETING_TIMES[i][0],
self.MEETING_TIMES[i][1]))

        for i in range(0, len(self.INSTRUCTORS)):
            self._instructors.append(Instructor(self.INSTRUCTORS[i][0], self.INSTRUCTORS[i][1]))

        #Assigning instructors or professors to each course with params (courseid, coursename,
instructors inside an array, max. students allowed in course)
        course1 = Course("C1", "INT404", [self._instructors[0]], 70)
        course2 = Course("C2", "CSE408", [self._instructors[1]], 70)
        course3 = Course("C3", "CSE325", [self._instructors[2]], 35) #For Practical or Lab periods
assign students are 35 only
        course4 = Course("C4", "MTH302", [self._instructors[3]], 70)
        course5 = Course("C5", "CSE316", [self._instructors[2]], 70)
        course6 = Course("C6", "PEV106", [self._instructors[4]], 70)
        course7 = Course("C7", "CSE306", [self._instructors[5]], 70)
        course8 = Course("C8", "CSE310", [self._instructors[6]], 70)
        course9 = Course("C9", "CSE307", [self._instructors[7]], 35) #Here also for practical lab
        self._courses = [course1, course2, course3, course4, course5, course6, course7,
course8, course9]

        dept1 = Department("CSE", [course1, course2, course3, course4, course5, course8])
        dept2 = Department("CPE", [course6])
        dept3 = Department("ECE", [course7, course9])
        self._depts = [dept1, dept2, dept3]

        self._numberOfClasses = 0
        for i in range(0, len(self._depts)):
            self._numberOfClasses += len(self._depts[i].get_courses())

    def get_rooms(self): return self._rooms
    def get_instructors(self): return self._instructors
    def get_courses(self): return self._courses
    def get_depts(self): return self._depts
    def get_meetingTimes(self): return self._meetingTimes
    def get_numberOfClasses(self): return self._numberOfClasses
```

```
class Schedule:
    def __init__(self):
        self._data = data
        self._classes = []
        self._numbOfConflicts = 0
        self._fitness = -1
        self._classNumb = 0
        self._isFitnessChanged = True

    def get_classes(self):
        self._isFitnessChanged = True
        return self._classes

    def get_numbOfConflicts(self): return self._numbOfConflicts

    def get_fitness(self):
        if (self._isFitnessChanged == True):
            self._fitness = self.calculate_fitness()
            self._isFitnessChanged = False
        return self._fitness

    def initialize(self):
        depts = self._data.get_depts()
        for i in range(0, len(depts)):
            courses = depts[i].get_courses()
            for j in range(0, len(courses)):
                newClass = Class(self._classNumb, depts[i], courses[j])
                self._classNumb += 1
                newClass.set_meetingTime(data.get_meetingTimes()[rnd.randrange(0,
len(data.get_meetingTimes()))])
                newClass.set_room(data.get_rooms()[rnd.randrange(0, len(data.get_rooms()))])
                newClass.set_instructor(courses[j].get_instructors()[rnd.randrange(0,
len(courses[j].get_instructors()))])
                self._classes.append(newClass)
        return self

    def calculate_fitness(self):
        self._numbOfConflicts = 0
        classes = self.get_classes()
        for i in range(0, len(classes)):
            if (classes[i].get_room().get_seatingCapacity() <
classes[i].get_course().get_maxNumbOfStudents()):
                self._numbOfConflicts += 1
```

```
            for j in range(0, len(classes)):
                if (j >= i):
                    if (classes[i].get_meetingTime() == classes[j].get_meetingTime() and
                    classes[i].get_id() != classes[j].get_id()):
                        if (classes[i].get_room() == classes[j].get_room()): self._numbOfConflicts += 1
                        if (classes[i].get_instructor() == classes[j].get_instructor()):
self._numbOfConflicts += 1
        return 1 / ((1.0*self._numbOfConflicts + 1))

    def __str__(self):
        returnValue = ""
        for i in range(0, len(self._classes)-1):
            returnValue += str(self._classes[i]) + ", "
        returnValue += str(self._classes[len(self._classes)-1])
        return returnValue


class Population:
    def __init__(self, size):
        self._size = size
        self._data = data
        self._schedules = []
        for i in range(0, size): self._schedules.append(Schedule().initialize())

    def get_schedules(self): return self._schedules


class GeneticAlgorithm:
    def evolve(self, population): return
self._mutate_population(self._crossover_population(population))

    def _crossover_population(self, pop):
        crossover_pop = Population(0)
        for i in range(NUMB_OF_ELITE_SCHEDULES):
            crossover_pop.get_schedules().append(pop.get_schedules()[i])
        i = NUMB_OF_ELITE_SCHEDULES
        while i < POPULATION_SIZE:
            schedule1 = self._select_tournament_population(pop).get_schedules()[0]
            schedule2 = self._select_tournament_population(pop).get_schedules()[0]
            crossover_pop.get_schedules().append(self._crossover_schedule(schedule1,
schedule2))
            i += 1
        return crossover_pop
```

```
    def _mutate_population(self, population):
        for i in range(NUMB_OF_ELITE_SCHEDULES, POPULATION_SIZE):
            self._mutate_schedule(population.get_schedules()[i])
        return population

    def _crossover_schedule(self, schedule1, schedule2):
        crossoverSchedule = Schedule().initialize()
        for i in range(0, len(crossoverSchedule.get_classes())):
            if (rnd.random() > 0.5): crossoverSchedule.get_classes()[i] = schedule1.get_classes()[i]
            else: crossoverSchedule.get_classes()[i] = schedule2.get_classes()[i]
        return crossoverSchedule

    def _mutate_schedule(self, mutateSchedule):
        schedule = Schedule().initialize()
        for i in range(0, len(mutateSchedule.get_classes())):
            if(MUTATION_RATE > rnd.random()): mutateSchedule.get_classes()[i] =
schedule.get_classes()[i]
        return mutateSchedule

    def _select_tournament_population(self, pop):
        tournament_pop = Population(0)
        i = 0
        while i < TOURNAMENT_SELECTION_SIZE:
            tournament_pop.get_schedules().append(pop.get_schedules()[rnd.randrange(0,
POPULATION_SIZE)])
            i += 1
        tournament_pop.get_schedules().sort(key=lambda x: x.get_fitness(), reverse=True)
        return tournament_pop


class Course:
    def __init__(self, number, name, instructors, maxNumbOfStudents):
        self._number = number
        self._name = name
        self._maxNumbOfStudents = maxNumbOfStudents
        self._instructors = instructors

    def get_number(self): return self._number

    def get_name(self): return self._name

    def get_instructors(self): return self._instructors

    def get_maxNumbOfStudents(self): return self._maxNumbOfStudents
```

```
    def __str__(self): return self._name


class Instructor:
    def __init__(self, id, name):
        self._id = id
        self._name = name

    def get_id(self): return self._id

    def get_name(self): return self._name

    def __str__(self): return self._name


class Room:
    def __init__(self, number, seatingCapacity):
        self._number = number
        self._seatingCapacity = seatingCapacity

    def get_number(self): return self._number

    def get_seatingCapacity(self): return self._seatingCapacity


class MeetingTime:
    def __init__(self, id, time):
        self._id = id
        self._time = time

    def get_id(self): return self._id

    def get_time(self): return self._time


class Department:
    def __init__(self, name, courses):
        self._name = name
        self._courses = courses

    def get_name(self): return self._name

    def get_courses(self): return self._courses
```

```
class Class:
    def __init__(self, id, dept, course):
        self._id = id
        self._dept = dept
        self._course = course
        self._instructor = None
        self._meetingTime = None
        self._room = None

    def get_id(self): return self._id

    def get_dept(self): return self._dept

    def get_course(self): return self._course

    def get_instructor(self): return self._instructor

    def get_meetingTime(self): return self._meetingTime

    def get_room(self): return self._room

    def set_instructor(self, instructor): self._instructor = instructor

    def set_meetingTime(self, meetingTime): self._meetingTime = meetingTime

    def set_room(self, room): self._room = room

    def __str__(self):
        return str(self._dept.get_name()) + "," + str(self._course.get_number()) + "," + \
            str(self._room.get_number()) + "," + str(self._instructor.get_id()) + "," +
str(self._meetingTime.get_id())


class DisplayMgr:
    def print_available_data(self):
        print("> All Available Data")
        self.print_dept()
        self.print_course()
        self.print_room()
        self.print_instructor()
        self.print_meeting_times()
```

```
    def print_dept(self):
        depts = data.get_depts()
        availableDeptsTable = prettytable.PrettyTable(['Dept', 'Courses'])
        for i in range(0, len(depts)):
            courses = depts.__getitem__(i).get_courses()
            tempStr = "["
            for j in range(0, len(courses) - 1):
                tempStr += courses[j].__str__() + ", "
            tempStr += courses[len(courses) - 1].__str__() + "]"
            availableDeptsTable.add_row([depts.__getitem__(i).get_name(), tempStr])
        print(availableDeptsTable)

    def print_course(self):
        availableCoursesTable = prettytable.PrettyTable(['Id', 'Course code', 'Max no. of
students', 'Instructors'])
        courses = data.get_courses()
        for i in range(0, len(courses)):
            instructors = courses[i].get_instructors()
            tempStr = ""
            for j in range(0, len(instructors) - 1):
                tempStr += instructors[j].__str__() + ", "
            tempStr += instructors[len(instructors) - 1].__str__()
            availableCoursesTable.add_row(
                [courses[i].get_number(), courses[i].get_name(),
str(courses[i].get_maxNumbOfStudents()), tempStr])
        print(availableCoursesTable)

    def print_instructor(self):
        availableInstructorsTable = prettytable.PrettyTable(['Id', 'Instructor'])
        instructors = data.get_instructors()
        for i in range(0, len(instructors)):
            availableInstructorsTable.add_row([instructors[i].get_id(), instructors[i].get_name()])
        print(availableInstructorsTable)

    def print_room(self):
        availableRoomsTable = prettytable.PrettyTable(['Room no.', 'Max seating capacity'])
        rooms = data.get_rooms()
        for i in range(0, len(rooms)):
            availableRoomsTable.add_row([str(rooms[i].get_number()),
str(rooms[i].get_seatingCapacity())])
        print(availableRoomsTable)

    def print_meeting_times(self):
        availableMeetingTimeTable = prettytable.PrettyTable(['Id', 'Meeting Time'])
```

```
        meetingTimes = data.get_meetingTimes()
        for i in range(0, len(meetingTimes)):
            availableMeetingTimeTable.add_row([meetingTimes[i].get_id(),
meetingTimes[i].get_time()])
        print(availableMeetingTimeTable)

    def print_generation(self, population):
        table1 = prettytable.PrettyTable(['Schedule no.', 'Fitness', 'No. of conflicts', 'Classes
[dept,class,room,instructor,meeting-time]'])
        schedules = population.get_schedules()
        for i in range(0, len(schedules)):
            table1.add_row([str(i), round(schedules[i].get_fitness(),3),
schedules[i].get_numbOfConflicts(), schedules[i].__str__()])
        print(table1)

    def print_schedule_as_table(self, schedule):
        classes = schedule.get_classes()
        table = prettytable.PrettyTable(['Class no.', 'Dept', 'Course (number, max no. of
students)', 'Room (Capacity)', 'Instructor (Id)',  'Meeting Time (Id)'])
        for i in range(0, len(classes)):
            table.add_row([str(i), classes[i].get_dept().get_name(),
classes[i].get_course().get_name() + " (" +
                    classes[i].get_course().get_number() + ", " +
                    str(classes[i].get_course().get_maxNumbOfStudents()) +")",
                    classes[i].get_room().get_number() + " (" +
str(classes[i].get_room().get_seatingCapacity()) + ")",
                    classes[i].get_instructor().get_name() +" (" +
str(classes[i].get_instructor().get_id()) +")",
                    classes[i].get_meetingTime().get_time() +" (" +
str(classes[i].get_meetingTime().get_id()) +")"])
        print(table)

data = Data()
displayMgr = DisplayMgr()
displayMgr.print_available_data()
generationNumber = 0
print("\n> Generation # "+str(generationNumber))
population = Population(POPULATION_SIZE)
population.get_schedules().sort(key=lambda x: x.get_fitness(), reverse=True)
displayMgr.print_generation(population)
displayMgr.print_schedule_as_table(population.get_schedules()[0])
geneticAlgorithm = GeneticAlgorithm()
while (population.get_schedules()[0].get_fitness() != 1.0):
    generationNumber += 1
```

```
    print("\n> Generation # " + str(generationNumber))
    population = geneticAlgorithm.evolve(population)
    population.get_schedules().sort(key=lambda x: x.get_fitness(), reverse=True)
    displayMgr.print_generation(population)
    displayMgr.print_schedule_as_table(population.get_schedules()[0])
print("\n\n")
```

# 1. Introduction:-

This project implements one of possible solutions for generating university class schedule. The proposed solution is based on methods of evolutionary computing or genetic algorithm.

The success of solution is estimated on fulfilment of given constraints and criteria.

## 1.1 Objectives:-

The following are the objectives of the project Intelligent Timetable preparation.

- To find a generic solution that will facilitate generating schedule for University
- Timetable must be the best optimal solution
- Conflicts must be least possible

# 2. Project description:-

Project follows the genetic algorithm technique to generate the results.

## 2.1 About Genetic Algorithm:-

Genetic Algorithms (GAs) are adaptive heuristic search algorithms that belong to the larger part of evolutionary algorithms. Genetic algorithms are based on the ideas of natural selection and genetics.

## 2.2 Assumption of the project:-

Resources (classroom, teacher) cannot overlap timewise

- No teacher can hold two classes at the same time
- No group can listen for two classes at the same time
- No classroom can receive two classes at the same time
- Class should take place only in one of the allowed classrooms, which means a theory class which has a maximum student capacity of 70 can not sit in the lab which a maximum student capacity of 35 only.

## 2.3. Solution:-

The algorithm for the timetable is represented as table with its columns as class no., dept, Course, Room no., Instructor, Meeting time allotted to each course with days, where number of classrooms, allowed meeting time, details of courses, name of Instructor of each course with instructor Id are declared already in the file.

The representation of the class schedule is done in the following picture:-

| Class no. | Dept | Course (number, max no. of students) | Room (Capacity) |
|---|---|---|---|
| 0 | CSE | INT404 (C1, 70) | 33-501 (70) |
| 1 | CSE | CSE408 (C2, 70) | 33-501 (70) |
| 2 | CSE | CSE325 (C3, 35) | 33-501 (70) |
| 3 | CSE | MTH302 (C4, 70) | 33-501 (70) |
| 4 | CSE | CSE316 (C5, 70) | 33-501 (70) |
| 5 | CSE | CSE310 (C8, 70) | 33-501 (70) |
| 6 | CPE | PEV106 (C6, 70) | 33-501 (70) |
| 7 | ECE | CSE306 (C7, 70) | 33-501 (70) |
| 8 | ECE | CSE307 (C9, 35) | 34-203 (35) |

In the last column of the above image Meeting time is allotted to each subject with its days for example:-

INT404 is allotted 33-501 room no. with meeting time MWF 11:00 - 12:00 (MT3), where MWF means on Monday, Wednesday, Friday from 11:00 AM – 12:00 PM with meeting id MT3.

| Meeting Time (Id) |
|---|
| MWF 11:00 - 12:00 (MT3) |
| TTH 13:00 - 14:00 (MT11) |
| MWF 16:00 - 17:00 (MT7) |
| MWF 14:00 - 15:00 (MT5) |
| TTH 10:00 - 11:00 (MT9) |
| MWF 10:00 - 11:00 (MT2) |
| TTH 09:00 - 10:00 (MT8) |
| TTH 14:00 - 15:00 (MT12) |
| MWF 15:00 - 16:00 (MT6) |

Also, the generation of the time table which evolves according to it's fitness and no. of conflicts on the basis of genetic algorithm is displayed in the below picture.

> Generation # 3

| Schedule no. | Fitness | No. of conflicts | Classes [dept,class,room,instructor,meeting-time] |
|---|---|---|---|
| 0 | 1.0 | 0 | CSE,C1,33-501,AP1,MT3, CSE,C2,33-501,AP2,MT11, CSE,C3,33-501,AP3,MT7, CSE,C4,33-501,AP4,MT5, CSE,C5,33-501,AP3,MT9, CSE,C8,33-501,AP7,MT2, CPE,C6,33-501,AP5,MT8, ECE,C7,33-501,AP6,MT12, ECE,C9,34-203,AP8,MT6 |
| 1 | 0.5 | 1 | CSE,C1,33-501,AP1,MT3, CSE,C2,33-501,AP2,MT11, CSE,C3,33-501,AP3,MT7, CSE,C4,34-203,AP4,MT7, CSE,C5,33-501,AP3,MT9, CSE,C8,33-501,AP7,MT2, CPE,C6,33-501,AP5,MT8, ECE,C7,33-501,AP6,MT12, ECE,C9,34-203,AP8,MT6 |
| 2 | 0.5 | 1 | CSE,C1,33-501,AP1,MT14, CSE,C2,33-501,AP2,MT11, CSE,C3,34-203,AP3,MT14, CSE,C4,33-501,AP4,MT5, CSE,C5,33-501,AP3,MT9, CSE,C8,33-501,AP7,MT6, CPE,C6,33-501,AP5,MT13, ECE,C7,34-203,AP6,MT11, ECE,C9,34-203,AP8,MT6 |
| 3 | 0.333 | 2 | CSE,C1,33-501,AP1,MT14, CSE,C2,34-203,AP2,MT14, CSE,C3,33-501,AP3,MT7, CSE,C4,34-203,AP4,MT7, CSE,C5,33-501,AP3,MT9, CSE,C8,33-501,AP7,MT6, CPE,C6,33-501,AP5,MT8, ECE,C7,33-501,AP6,MT12, ECE,C9,34-203,AP8,MT5 |
| 4 | 0.333 | 2 | CSE,C1,33-501,AP1,MT3, CSE,C2,33-501,AP2,MT12, CSE,C3,33-501,AP3,MT13, CSE,C4,34-203,AP4,MT10, CSE,C5,33-501,AP3,MT9, CSE,C8,33-501,AP7,MT6, CPE,C6,33-501,AP5,MT8, ECE,C7,33-501,AP6,MT12, ECE,C9,34-203,AP8,MT4 |
| 5 | 0.333 | 2 | CSE,C1,33-501,AP1,MT14, CSE,C2,33-501,AP2,MT11, CSE,C3,33-501,AP3,MT7, CSE,C4,34-203,AP4,MT2, CSE,C5,33-501,AP3,MT9, CSE,C8,34-203,AP7,MT5, CPE,C6,33-501,AP5,MT13, ECE,C7,33-501,AP6,MT12, ECE,C9,34-203,AP8,MT6 |
| 6 | 0.333 | 2 | CSE,C1,33-501,AP1,MT3, CSE,C2,33-501,AP2,MT11, CSE,C3,33-501,AP3,MT7, CSE,C4,34-203,AP4,MT10, CSE,C5,34-203,AP3,MT1, CSE,C8,33-501,AP7,MT6, CPE,C6,33-501,AP5,MT8, ECE,C7,33-501,AP6,MT12, ECE,C9,34-203,AP8,MT6 |
| 7 | 0.25 | 3 | CSE,C1,33-501,AP1,MT3, CSE,C2,33-501,AP2,MT11, CSE,C3,33-501,AP3,MT7, CSE,C4,34-203,AP4,MT7, CSE,C5,33-501,AP3,MT9, CSE,C8,34-203,AP7,MT6, CPE,C6,33-501,AP5,MT8, ECE,C7,33-501,AP6,MT12, ECE,C9,34-203,AP8,MT6 |
| 8 | 0.2 | 4 | CSE,C1,33-501,AP1,MT6, CSE,C2,33-501,AP2,MT12, CSE,C3,33-501,AP3,MT7, CSE,C4,33-501,AP4,MT12, CSE,C5,33-501,AP3,MT5, CSE,C8,34-203,AP7,MT2, CPE,C6,33-501,AP5,MT13, ECE,C7,34-203,AP6,MT11, ECE,C9,33-501,AP8,MT5 |

As, we can see in the above picture we have four columns Schedule no., Fitness, No. of conflicts, Classes. Here it started from the 8$^{th}$ row and in the 8$^{th}$ row the fitness was 0.2, No. of conflicts were 4 and schedule which was generated is shown in the classes column with comma seperated values.

For example CSE,C1,33-501,AP1,MT6 means dept is CSE, course id is C1 which is declared already in the file for the INT404, and room no. is 33-501, instructor id is AP1 and MT6 means meeting time 6 which is Monday, Wednesday, Friday from 15:00hrs – 16:00hrs.

After that evolved and finally reached to the $0^{th}$ schedule where we can see the No. of conflicts are zero and the Fitness is 1.

So it is the best optimal solution and it is then printed in the final format which is shown earlier.

It is not always true that no confilcts will arise, it totally depends how much hard the constraints which had been provided are.

However, in that case also it will always provide the best optimal soultion.

### 2.4. Modules which are used:-

**PrettyTable:-** Only one external module has been used, i.e. PrettyTable. PrettyTable is a module which is used to log the matrix or table output in a prettier format. It is a open source Library under the pip repository.

## 3. Usage:-

To run this code first open the terminal in the directory and run the command:-

***pip intall PrettyTable***

If there is no error then, PrettyTable will be installed in the system after that we can run the command

***python ai.py***

**Note:- It is necessary that python must be installed in the system to run the code.**