National University of Computer and Emerging Sciences, Islamabad

FAST School of Computer Science

CS2002-Artificial Intelligence Project

***Submitted by*:**

1. Syed Saifullah Rizwan (21I-0830)

Submitted to: Ma'am Labiba Fahad

DATA SET GENERATION:

For starters we needed a data set to generate well a timetable or a chromosome of a timetable: For this purpose I made Panda DataFrames for each { professors, courses, sections, rooms, timeslots, days}

Each dataframe has specific information relevant to them. However, each of them have a column name of 'Enc' which is used for binary encoding of the chromosome later on.

Here is the Complete Data Set :

```
Professors:
   ProfessorID           Name  Enc
0            1    Kashif Munir  000
1            2  Mukhtaar Ullah  001
2            3    Naveed Ahmed  010
3            4   Farukh Bashir  011
4            5    Amina Ashfaq  100


Courses:
   CourseID              Name  IsLab  ProfessorID  Enc
0         1                OS  False            1  000
1         2            OS Lab   True            1  001
2         3                AI  False            3  010
3         4            AI Lab   True            3  011
4         5              CNET  False            5  100
5         6  Network Security  False            5  101
```

```
Rooms:
    RoomID  Capacity  Floor  Enc
0        1        60      1  000
1        2       120      1  001
2        3        60      2  010
3        4        60      2  011
4        5        60      3  100
5        6       120      3  101
6        7        60      0  110
7        8       120      0  111
8        9        60      1  000
9       10        60      1  001
10      11        60      0  010
11      12       120      0  011
12      13        60      1  100
13      14        60      1  101
14      15       120      2  110

Sections:
   SectionID  CourseID  Capacity  Enc
0          1         1        50  000
1          1         6        50  001
2          2         2        25  010
3          3         3        55  011
4          4         4        20  100
5          5         5        60  101
6          5         1        60  110
```

```
TimeSlots:
   TimeSlotID       Timings  Enc
0           1    (8.3, 9.5)  000
1           2    (10, 11.2)  001
2           3  (11.3, 12.5)  010
3           4    (13, 14.2)  011
4           5  (14.3, 15.5)  100
5           6    (16, 17.2)  101
```

```
Days:
         Day  Enc
0     Monday  000
1    Tuesday  001
2  Wednesday  010
3   Thursday  011
4     Friday  100
```

CHROMOSOME GENERATION:

To generate Chromosomes for the timetable I defined a function:

```
def generate_chromosome(courses, sections, professors, rooms, timeslots, days):
```

which takes all the dataframes of input data and makes a chromosome tuple :

`detailed_chromosome = []` (Chromosome Containing Meaningful Data)

`encoded_chromosome = []` (Chromosome Containing Binary Encoded Data)

it iterates through the section dataset and for every row (needed for multiple courses for same section) calculates and stores the data for the required course, professor, room. Picks a random day and timeslot (checking not pick the last possible one (for lab as it needs 2)).

Then it converts the Tuple of Timeslots to string by:

```
 # Convert timeslot tuple to string format for detailed chromosome
first_timeslot_str= f"{first_timeslot['Timings'][0]:.2f},
{first_timeslot['Timings'][1]:.2f}"
```

So that it is easier to manipulate later on when we make separate timetables for separate days.

After this Both the encoded_gene and the detailed_gene are constructed:

```
# Detailed gene construction
            detailed_gene = {
               'Course': course['Name'],
                'Type': 'Lab' if course['IsLab'] else 'Theory',
                'Section': row['SectionID'],
                'Capacity': row['Capacity'],
                'Professor': professor['Name'],
                'First Lecture Day': first_day['Day'],
                'First Lecture Timeslot': first_timeslot_str,
                'First Lecture Room': room['RoomID'],
                'First Lecture Room Capacity': room['Capacity']
            }
            # Encoded gene construction using binary strings
            encoded_gene = {
                'Course': course['Enc'],
                'Type': '1' if course['IsLab'] else '0',
                'Section': row['Enc'],
                'Capacity': format(row['Capacity'], '07b'),
                'Professor': professor['Enc'],
                'First Lecture Day': first_day['Enc'],
                'First Lecture Timeslot': first_timeslot['Enc'],
                'First Lecture Room': room['Enc'],
                'First Lecture Room Capacity': '1' if room['Capacity'] == 120
else '0'
            }
```

As I forgot and didn't make a new dataframe for the Section Strength I added it as a field in Section named Capacity, For Binary encoding I added

```
'Capacity': format(row['Capacity'], '07b')
```

Which just converts the Capacity to Binary and pads it.

After this I check if the Course is a Lab, if so, I select the second timeslot to the next timeslot of the first one, then convert it to string and then accordingly update both the genes:

```
          if course['IsLab']:
                # For lab courses, ensure two consecutive time slots
                second_timeslot = timeslots.iloc[first_timeslot_index + 1]
                second_timeslot_str=f"{second_timeslot['Timings'][0]:.2f},
                                      {second_timeslot['Timings'][1]:.2f}"
                detailed_gene.update({
                    'Second Lecture Day': first_day['Day'],
                    'Second Lecture Timeslot': second_timeslot_str,
                    'Second Lecture Room': room['RoomID'],
                    'Second Lecture Room Capacity': room['Capacity']
                })
                encoded_gene.update({
                    'Second Lecture Day': first_day['Enc'],
                    'Second Lecture Timeslot': second_timeslot['Enc'],
                    'Second Lecture Room': room['Enc'],
                    'Second Lecture Room Capacity': '1' if room['Capacity'] ==
120 else '0'
                })
```

Then if it is not a Lab, I check the Soft Constraint of same or alternate days (will explain later on) by

```
# Ensure second lecture day is not the same or adjacent to the first
valid_days = days[~days['Day'].isin([first_day['Day'],
day_before(first_day['Day'], days), day_after(first_day['Day'], days)])]
```

This line of code filters out rows from the DataFrame days where the 'Day' column is not the same as the first lecture day or its adjacent days.

2 helper functions are used to calculate the valid days

```
def day_before(day, days_df):
def day_after(day, days_df):
```

day_before(day, days_df): Retrieves the day before the given **day** from the DataFrame **days_df** by finding its index and returning the corresponding value if it exists, otherwise returns **None**.

day_after(day, days_df): Retrieves the day after the given **day** from the DataFrame **days_df** by finding its index and returning the corresponding value if it exists, otherwise returns **None**.

Then using the valid days, I select a second day and choose a random timeslot and room where the capacity of the room is bigger than the capacity of the course_section (Section studying that course).

```
second_day = valid_days.sample().iloc[0]
second_timeslot = timeslots.sample().iloc[0]
second_room = rooms[rooms['Capacity'] >= row['Capacity']].sample().iloc[0]
```

Then I update the both the Genes accordingly by adding details of Second Lecture. (Day, Timeslot, Room, Capacity)

Fitness Calculation:

To calculate the Fitness, I defined a function:

```
def calculate_fitness(detailed_chromosome, encoded_chromosome, timeslots, rooms,
days):
```

in which, I Constructs a dictionary **timeslot_dict** from the DataFrame **timeslots**, mapping each **'Enc'** value to its corresponding **'Timings'** value for quick lookup.

```
    # Construct a dictionary from timeslot Enc to Timings for quick lookup
timeslot_dict = {ts['Enc']: ts['Timings'] for idx, ts in timeslots.iterrows()}
```

I will now be talking here mostly about how the chromosomes are constructed and which Hard and Soft Constraints I have applied and check here.

As for the Chromosomes, each chromosome whether detailed or encoded has n genes where n is the course_section meaning sum of courses for each section.

to check for classes a nested loop is applied to get 2 genes (1 course_section).

```
for i in range(len(encoded_chromosome)):
    gene1 = encoded_chromosome[i]
    prof11, day11, slot11, room11 = gene1['Professor'],
                                    gene1['First LectureDay'],
                                    gene1['First Lecture Timeslot'],
                                    gene1['First Lecture Room']
  day12, slot12, room12 = gene1['Second Lecture Day'],
                          gene1['SecondLecture Timeslot'],
                          gene1['Second Lecture Room']
        for j in range(i + 1, len(encoded_chromosome)):
            gene2 = encoded_chromosome[j]
            prof21, day21, slot21, room21 = gene2['Professor'],
                                            gene2['FirstLecture Day'],
                                            gene2['First Lecture Timeslot'],
                                            gene2['First Lecture Room']
            day22, slot22, room22 =  gene2['Second Lecture Day'],
                                     gene2['SecondLecture Timeslot'],
                                     gene2['Second Lecture Room']
```

values of both genes are compared if there is a conflict a penalty of +10 is applied if timeslot clash or room clash (Hard Constraint)

and a +1 penalty is applied if a Lab is scheduled in Morning or A Theory class scheduled in evening.

Then the negative value of penalty is returned -> `return -penalty`

3 helper functions are defined to check for timeslot overlap and to calculate if the timeslot is in morning or afternoon:

```
def timeslot_overlap(timeslot1, timeslot2):

def in_morning(timeslot):

def in_afternoon(timeslot):
```

timeslot_overlap(timeslot1, timeslot2): Checks if two timeslots overlap by comparing their start and end times, returning True if there is an overlap, False otherwise.

in_morning(timeslot): Determines if the given timeslot falls within the morning session, assuming the morning session ends at 12:50.

in_afternoon(timeslot): Determines if the given timeslot falls within the afternoon session, assuming the afternoon session starts at 13:00.

HARD AND SOFT CONSTRAINTS:

- Classes can only be scheduled in free classrooms. ✓
- A classroom should be big enough to accommodate the section. There should be two categories of classrooms: classroom (60) and large hall (120). ✓
- A professor should not be assigned two different lectures at the same time. ✓
- The same section cannot be assigned to two different rooms at the same time. ✓
- A room cannot be assigned for two different sections at the same time. ✓

- No professor can teach more than 3 courses. ✔
- No section can have more than 5 courses in a semester. ✔
- Each course would have two lectures per week not on the same or adjacent days. ✔
- Lab lectures should be conducted in two consecutive slots. ✔
- 15 mins breaks allowed between consecutive classes to ensure that there is sufficient time for transitions between classes. ✔ (10 mins)

Soft Constraints:

- All the theory classes should be taught in the morning session and all the lab sessions should be done in the afternoon session. ✔
- Teachers/students may be facilitated by minimizing the number of floors they have to traverse. That is, as much as possible, scheduled classes should be on the same floor for either party. ✗
- A class should be held in the same classroom across the whole week. ✗
- Teachers may prefer longer blocks of continuous teaching time to minimize interruptions and maximize productivity except when the courses are different. ✗

GENETIC ALGORITHM:

To generate the population, I defined a function:

```
def generate_population(pop_size, courses, sections, professors, rooms,
timeslots, days):
    return [generate_chromosome(courses, sections, professors, rooms, timeslots,
days) for _ in range(pop_size)]
```

Which just calls the `generate_chromosome(…)` function `pop_size` times.

Then for selection I defined:

```
def select(population, fitness_scores, num_parents):
    # Tournament selection
    selected_indices = sorted(range(len(fitness_scores)),
    key=lambda i:   fitness_scores[i], reverse=True)[:num_parents]
    return [population[i] for i in selected_indices]
```

In `select(…)` a specified number of parents from the population using tournament selection based on their fitness scores, returning their indices sorted in descending order of fitness scores.

Then For Crossover and Mutation I defined:

```
def crossover(parent1, parent2):
def mutate(child, mutation_rate, timeslots):
```

`crossover(…)` does a one point crossover and produces 2 child Chromosomes (encoded) then to generate the detailed part of the chromosome I defined a function called :

```
def regenerate_detailed_from_encoded(encoded_chromosome, courses, professors,
rooms, timeslots, days):
```

Which basically takes the Encoded part of the chromosome and maps it to the original dataset to get the Detailed part. It checks for the alternate or same day constraint for Theory Courses as well, if the Second day is either the same or alternate then the First Day then it is fixed. It also checks if the course changed was a lab, then if its timing (2 consecutive slots) is correct. If they are not correct then it fixes the timing and then updates the changes in the Encoded Part of the chromosome as well so every change is recorded.

1 helper function `def get_adjacent_days(current_day, days_df):`

get_adjacent_days(current_day, days_df): Returns a list of adjacent days to the **current_day** from the DataFrame **days_df** to prevent scheduling on consecutive days for theory courses.

In `mutate(…)` if the `random.random()` returns a bigger value than the mutation rate then it will change the timeslot then there is a chance if another `random.random()` is bigger than 0.5 to change the room. After this `regenerate_detailed_from_encoded(…)`

Is called to generate the detailed part of the chromosome.

For the Actual function for genetic algorithm I defined:

```
def genetic_algorithm(courses, sections, professors, rooms, timeslots, days,
pop_size, num_generations, mutation_rate):
```

It generates a population, calculates the fitness and checks if the best fitness of the population is better than the best_fitness than its index is stored and the actual chromosome is saved to `best_solution`.

If Perfect Fitness has been reached (which is 0) indicating 0 conflicts with the Timetable then the loop breaks to print the output.

Else, number of parents are selected through selection. And then Crossover and Mutation is applied.

After which a new population is made from

```
 # Create the new generation

  population = parents + children
```

As The output was so lengthy, It overlapped in the output window. So I store it into a text file.

```
# Define the file path to store the table

output_file_path = "output_table.txt"

# Write the table to the text file

with open(output_file_path, "w") as file:

    file.write(str(table))
```

Then to store in a separate file and print separate timetables for each day

```
["Monday", "Tuesday", "Wednesday", "Thursday", "Friday"]
```

I defined the following functions :

```
def generate_daily_timetables(detailed_chromosome):

def get_sort_key(time_slot):
```

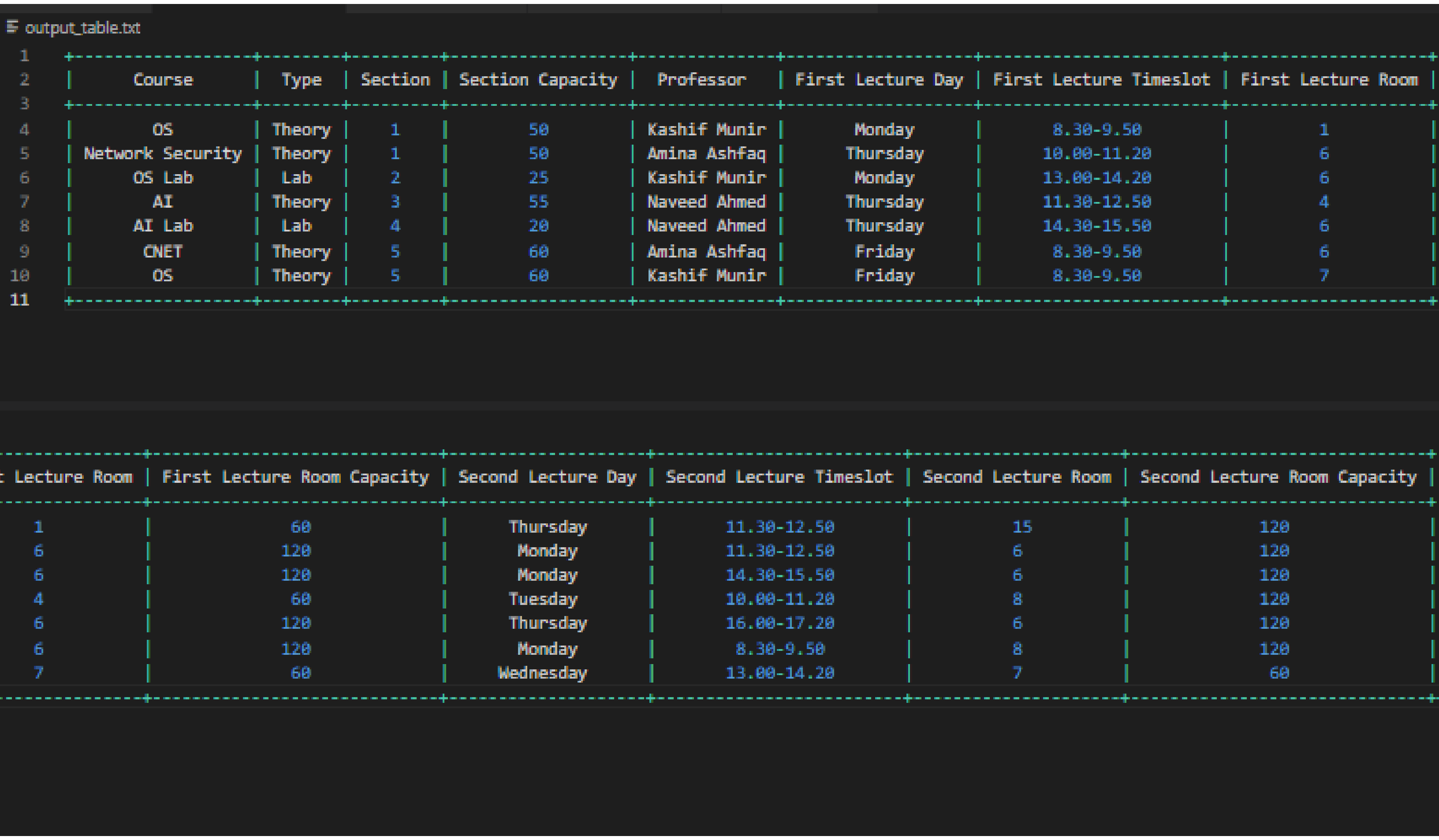
```python
def convert_to_twelve_hour(time_str):

def print_timetables(timetables):

def write_timetables_to_file(timetables, file_path):
```

generate_daily_timetables(detailed_chromosome): Generates daily timetables from the detailed chromosome, sorting lectures by time and formatting them into a dictionary of DataFrames for each day.

print_timetables(timetables): Prints timetables for each day in the **timetables** dictionary using PrettyTable for better visualization.

write_timetables_to_file(timetables, file_path): Writes timetables for each day in the **timetables** dictionary to a text file specified by **file_path**, using PrettyTable for formatting.

Best_chromosome(example) (fitness 0):

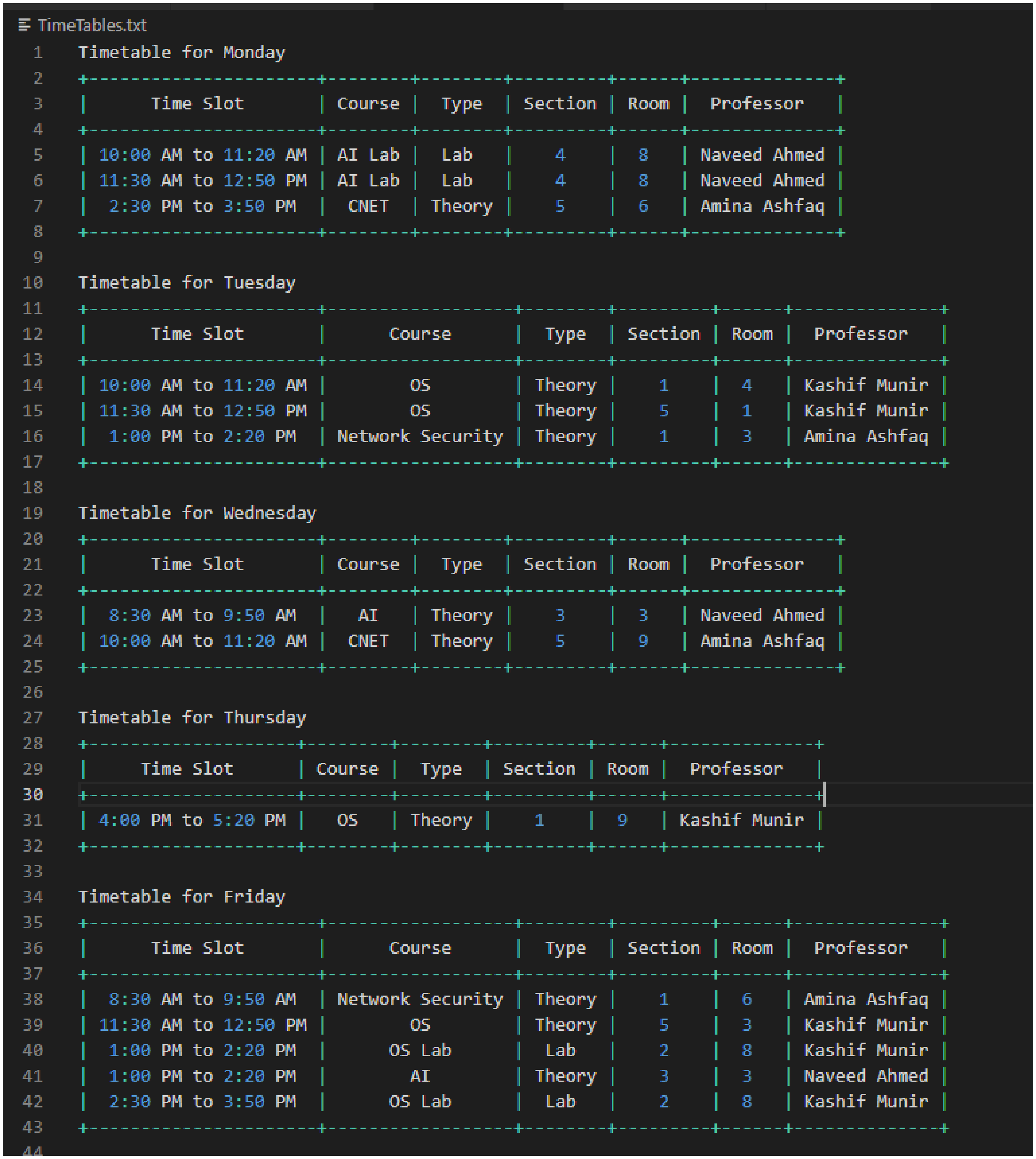


output_table.txt

```
+------------------+--------+---------+------------------+--------------+------------------+----------------------+-------------------+
|      Course      |  Type  | Section | Section Capacity |  Professor   | First Lecture Day | First Lecture Timeslot | First Lecture Room |
+------------------+--------+---------+------------------+--------------+------------------+----------------------+-------------------+
|        OS        | Theory |    1    |        50        | Kashif Munir |      Monday      |       8.30-9.50      |         1         |
| Network Security | Theory |    1    |        50        | Amina Ashfaq |     Thursday     |      10.00-11.20     |         6         |
|      OS Lab      |  Lab   |    2    |        25        | Kashif Munir |      Monday      |      13.00-14.20     |         6         |
|        AI        | Theory |    3    |        55        | Naveed Ahmed |     Thursday     |      11.30-12.50     |         4         |
|      AI Lab      |  Lab   |    4    |        20        | Naveed Ahmed |     Thursday     |      14.30-15.50     |         6         |
|       CNET       | Theory |    5    |        60        | Amina Ashfaq |      Friday      |       8.30-9.50      |         6         |
|        OS        | Theory |    5    |        60        | Kashif Munir |      Friday      |       8.30-9.50      |         7         |
+------------------+--------+---------+------------------+--------------+------------------+----------------------+-------------------+
```

```
-------------------+----------------------------+--------------------+-------------------------+---------------------+------------------------------+
t Lecture Room | First Lecture Room Capacity | Second Lecture Day | Second Lecture Timeslot | Second Lecture Room | Second Lecture Room Capacity |
-------------------+----------------------------+--------------------+-------------------------+---------------------+------------------------------+
  1            |             60             |      Thursday      |       11.30-12.50       |          15         |             120              |
  6            |            120             |       Monday       |       11.30-12.50       |          6          |             120              |
  6            |            120             |       Monday       |       14.30-15.50       |          6          |             120              |
  4            |             60             |      Tuesday       |       10.00-11.20       |          8          |             120              |
  6            |            120             |      Thursday      |       16.00-17.20       |          6          |             120              |
  6            |            120             |       Monday       |        8.30-9.50        |          8          |             120              |
  7            |             60             |     Wednesday      |       13.00-14.20       |          7          |              60              |
-------------------+----------------------------+--------------------+-------------------------+---------------------+------------------------------+
```

Timetable of the best chromosome:



TimeTables.txt

```
Timetable for Monday
+----------------------+--------+--------+---------+------+---------------+
|       Time Slot      | Course |  Type  | Section | Room |   Professor   |
+----------------------+--------+--------+---------+------+---------------+
| 10:00 AM to 11:20 AM | AI Lab |  Lab   |    4    |  8   | Naveed Ahmed  |
| 11:30 AM to 12:50 PM | AI Lab |  Lab   |    4    |  8   | Naveed Ahmed  |
|  2:30 PM to 3:50 PM  |  CNET  | Theory |    5    |  6   | Amina Ashfaq  |
+----------------------+--------+--------+---------+------+---------------+

Timetable for Tuesday
+----------------------+------------------+--------+---------+------+---------------+
|       Time Slot      |      Course      |  Type  | Section | Room |   Professor   |
+----------------------+------------------+--------+---------+------+---------------+
| 10:00 AM to 11:20 AM |        OS        | Theory |    1    |  4   | Kashif Munir  |
| 11:30 AM to 12:50 PM |        OS        | Theory |    5    |  1   | Kashif Munir  |
|  1:00 PM to 2:20 PM  | Network Security | Theory |    1    |  3   | Amina Ashfaq  |
+----------------------+------------------+--------+---------+------+---------------+

Timetable for Wednesday
+----------------------+--------+--------+---------+------+---------------+
|       Time Slot      | Course |  Type  | Section | Room |   Professor   |
+----------------------+--------+--------+---------+------+---------------+
|  8:30 AM to 9:50 AM  |   AI   | Theory |    3    |  3   | Naveed Ahmed  |
| 10:00 AM to 11:20 AM |  CNET  | Theory |    5    |  9   | Amina Ashfaq  |
+----------------------+--------+--------+---------+------+---------------+

Timetable for Thursday
+--------------------+--------+--------+---------+------+---------------+
|      Time Slot     | Course |  Type  | Section | Room |   Professor   |
+--------------------+--------+--------+---------+------+---------------+
| 4:00 PM to 5:20 PM |   OS   | Theory |    1    |  9   | Kashif Munir  |
+--------------------+--------+--------+---------+------+---------------+

Timetable for Friday
+----------------------+------------------+--------+---------+------+---------------+
|       Time Slot      |      Course      |  Type  | Section | Room |   Professor   |
+----------------------+------------------+--------+---------+------+---------------+
|  8:30 AM to 9:50 AM  | Network Security | Theory |    1    |  6   | Amina Ashfaq  |
| 11:30 AM to 12:50 PM |        OS        | Theory |    5    |  3   | Kashif Munir  |
|  1:00 PM to 2:20 PM  |      OS Lab      |  Lab   |    2    |  8   | Kashif Munir  |
|  1:00 PM to 2:20 PM  |        AI        | Theory |    3    |  3   | Naveed Ahmed  |
|  2:30 PM to 3:50 PM  |      OS Lab      |  Lab   |    2    |  8   | Kashif Munir  |
+----------------------+------------------+--------+---------+------+---------------+
```