Python based Automated Class Routine Generation System Using Teacher Preferences and Ranking

Abstract:

Class routine generation in academic institutions is a complex and time-intensive process, often cause to human error and conflicts in scheduling. This project proposes an automated class routine generation system that integrates teacher preferences, course requirements, and priority rankings to produce an efficient and fair schedule.

Developed in Python with a Tkinter-based GUI, the system reads structured data inputs such as teacher list with teacher name and ranking, courses with course title and course code and credit and year and term, teacher preferences then applies a priority-based allocation algorithm like higher priority (lower rank) teacher get class first based on their preference avoiding conflicts, and exports the final routine in a readable Excel format.

This system reduces administrative workload, minimizes scheduling conflicts, and improves overall satisfaction among teachers.

1. Introduction:

What is Class Routine Generation?

Class routine generation is the process of assigning courses, teachers, and years into specific time slots in a structured and conflict-free manner. It must consider various constraints such as teacher availability, course credit hour, overlapping classes, and lab class and theory class. It ensures am efficient class routine with consider all constraints and priorities effectively.

Why Automate Class Routine Generation is needed?

Manual routine preparation is more complex and time-consuming and may have errors and conflicts. It becomes increasingly difficult as the number of teachers, and courses grows. Teachers often have preferences based on personal schedules or expertise, which are hard to accommodate manually. An automated solution:

- Saves time and labor,
- Minimizes human error,
- Ensures fair and preference-based allocation,
- Allows easy modification and scalability,
- Provides outputs in universally readable formats (e.g., Excel).

This project addresses these challenges by creating a Python-based system that considers teacher preferences, prioritizes based on ranks, and uses an intelligent reassignment algorithm to generate optimized class routines.

2. Related Work:

Among various approaches we use greedy algorithm for generating class routine. Greedy Approach:

- For each day and each year it assigns course to available slots if teacher is available on this slot based on his preference
- It ensures that in one day same teacher does not exists in same slot for different course in different year strictly maintain this constraint.
- If conflict occurs then it try to reassign course in different available time slot moving previous course if slot is available for this. In this way it avoids overlapping.
- It try to assign courses for higher rank teacher first and ensures most of the teacher satisfaction.

3. System Requirement Analysis:

User Requirements:

The user requirement for this system is to make the system fast, flexible, less prone to error, reduce expenses and save time. Time can be saved by scheduling the routine in an automatic manner. And also implement this system in greedy algorithm using proper constraints.

Functional Requirements:

- Input: Take all inputs such as teacher list with rank, course with assigned teacher and his preference then save this inputs to JSON file to implement the system.
- Conflict-Handling: This system ensures that no overlapping arises for same day in same slot for any teacher or between two teachers. That means the system able to resolve all kind of conflicts.
- Routine Generation: Generate automatically a weekly class routine with given data.
- Output: Show routine clearly in Excel format for readability and editing.

Non-Functional Requirements:

Non-functional requirements specify the criteria that can be used to judge the operation of a system rather than specific behaviors. They are contrasted with functional requirements that define specific behavior or functions. Non-functional requirements of the developed system as below:

- Time slots are fixed but user will able to edit, add, and delete different time slots as needed.
- Every semester and session are changeable.
- All information is addable/ delete able/ updateable.

4. System Design Methodology:

The proposed automated routine management system offers a different segment that helps to manage class schedules in an interactive and efficient way. In this section, a proper description of design methodologies and approaches used for implementing this system is given.

Using Greedy approach:

Greedy Teacher Priority: Higher ranked teachera are scheduled first.

First Fill Available Slot: Assign the empty slot first according to teachers Preference.

Conflict Handling: Handle conflict both for same teacher and different teachers in case of any year and day efficiently using backtracking.

Constraints:

Course Credit Hour: For theory course must be scheduled for a number of hours equal to its credit and for sessional must be scheduled for 3 hours. **Slot:** Theory courses are scheduled between 9 AM and 1 PM (first half) and Lab courses are scheduled between 2 PM and 5 PM (second half).

Teacher Ranking: Higher ranked teacher get priority in slot selection.

Available Slots: Classes are assigned if slots are available for this course teacher.

Avoid Conflict: No overlapping classes are allowed either for same teacher or different course teacher.

Algorithm:

- **Input Data**

  Load the teacher list: IDs, names.

  Load the course list: course ID, year, term, assigned teacher.

  Load the teacher preferences: preferred time slots, rankings.

- **Sort Preferences**

  Sort the teacher preferences for each teacher by their rank

- **Initialize Empty Routine**

  Create an empty routine matrix with time slots and courses.

- **Assign Teachers to Courses**

`

For each course:

  Retrieve the sorted preferences of the teacher.

  Check for conflicts (overlapping assignments).

  If there is no conflict, assign the teacher to the course.

  If there is a conflict, reassign the teacher to another available slot.

- **Conflict Resolve**

  To ensure a valid and practical routine, the system implements a conflict resolution mechanism that intelligently manages scheduling clashes. Here's how it works:

  Conflict Detection:
  During the assignment process, the algorithm checks whether the teacher assigned to a course already has a class scheduled in the same time slot. If a conflict is found (i.e., the teacher is already assigned elsewhere during that period), the system marks it as a scheduling conflict.

  Resolution with Backtracking:
  Instead of rejecting the course outright, the system attempts to resolve the conflict by using a **backtracking approach**:
  1. It looks into the teacher's list of preferred slots.
  2. It searches for alternative time slots where the teacher is available and no conflicts exist.
  3. If such a slot is found, the existing class occupying the desired slot may be **moved to another valid slot** (if possible).

4. This "swap-and-retry" mechanism ensures minimal disruption and preserves teacher preferences.

Fair Assignment & Continuity: The resolution algorithm gives **priority to higher-ranked teachers** when resolving overlapping slots. It ensures the distribution remains **fair and continuous**, avoiding isolated gaps in the schedule while ensuring maximum teacher satisfaction.

Fallback Warning: If no conflict-free slot can be found or reassignments are impossible due to constraints, the system issues a warning, allowing manual review or later intervention.

- **Balanced Distribution**

  Ensure that each teacher has a fair load of courses, avoiding overassigning to any one teacher.

- **Export Routine**

  Generate the final routine based on assignments.
  Export the routine to an Excel file for easy viewing and sharing.

- **Output**

  Display the routine visually on the Tkinter GUI.

Allow the user to export the routine to Excel.

➢ Pseudo code:

```
// Initialize data
courses ← load "courses.json"
teacher_rank ← load "teacher_rank.json"
teacher_prefs ← load "teacher_preferences.json"
days ← ["Sunday", "Monday", "Tuesday", "Wednesday", "Thursday"]

// Initialize empty structures
routine ← [4 years × 5 days × 7 slots] filled with None
teacher_schedule ← defaultdict(set)  // Each teacher has a set of slots per day

// Sort courses by teacher rank (ascending)
sort courses by teacher_rank.get(course.teacher, ∞)


// Assignment functions
```

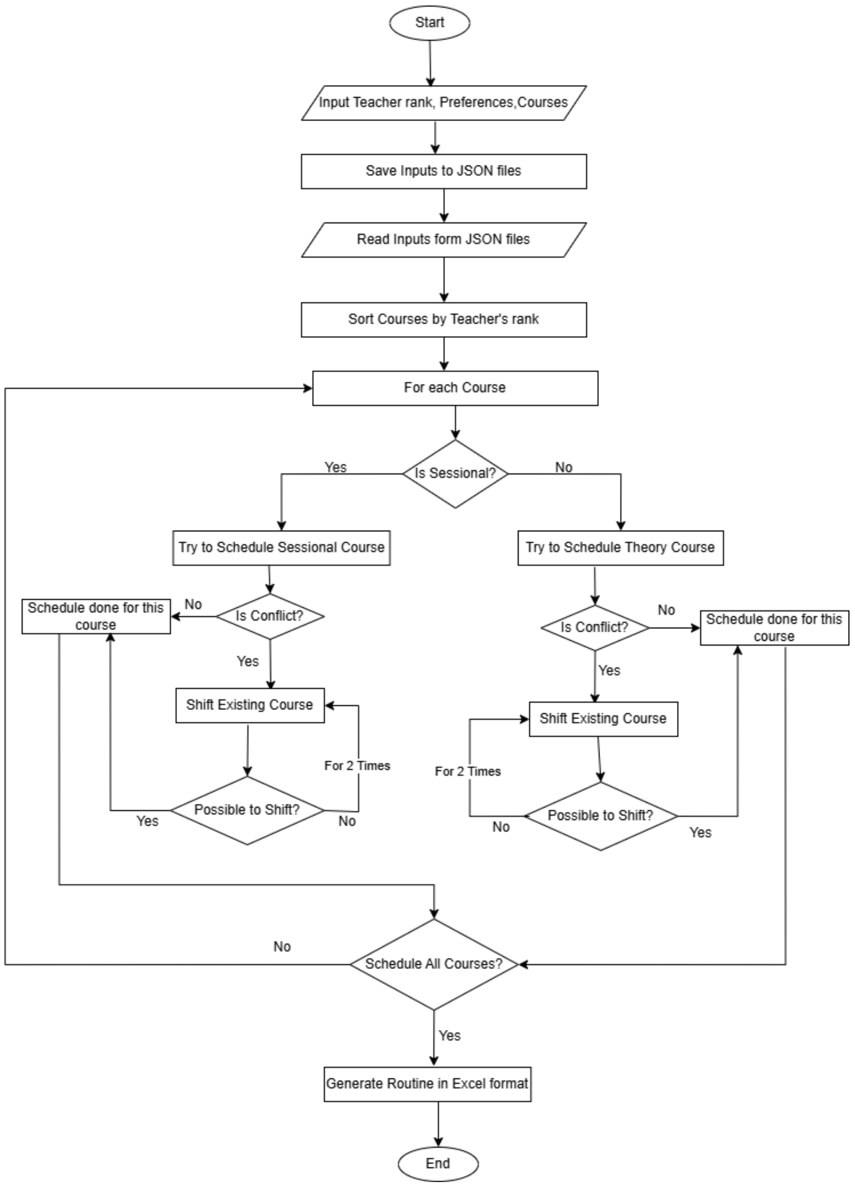
```
function can_assign(teacher, year, day, slot):
    return routine[year][day][slot] = None ∧ slot ∉ teacher_schedule[teacher][day]

function assign_slot(year, day, slot, course, teacher):
    routine[year][day][slot] ⟵ (course, teacher)
    teacher_schedule[teacher][day].add(slot)

    if course is 3-hour:
        routine[year][day][slot+1] ⟵ (course, teacher)
        routine[year][day][slot+2] ⟵ (course, teacher)
        teacher_schedule[teacher][day].add(slot+1)
        teacher_schedule[teacher][day].add(slot+2)

function reassign_slot(year, day, slot, requester):
    current_course, current_teacher ⟵ routine[year][day][slot]

    if current_teacher = None ∨ current_teacher = requester:
        return true

    for (new_day, new_slot) in teacher_prefs[current_teacher]:
        if (new_day, new_slot) = (day, slot):
            continue

        if can_assign(current_teacher, year, new_day, new_slot):
            assign_slot(year, new_day, new_slot, current_course, current_teacher)
            return true

        if reassign_slot(year, new_day, new_slot, current_teacher):
            assign_slot(year, new_day, new_slot, current_course, current_teacher)
            return true

    return false



// Main course assignment
for each course in courses:
    year ⟵ course.year
    code ⟵ course.code
    teacher ⟵ course.teacher
    credits_needed ⟵ course.credit
    assigned ⟵ 0

    // Try preferred slots first
    for (day, slot) in teacher_prefs.get(teacher, []):
```

```
  if assigned ≥ credits_needed:
    break

  if can_assign(teacher, year, day, slot):
    assign_slot(year, day, slot, code, teacher)
    assigned ← assigned + (1 if course is 1-hour else 3)

// Try reassigning if needed
for (day, slot) in teacher_prefs.get(teacher, []):
  if assigned ≥ credits_needed:
    break

  if reassign_slot(year, day, slot, teacher):
    assign_slot(year, day, slot, code, teacher)
    assigned ← assigned + (1 if course is 1-hour else 3)
```

➢ Explanation:

- **Input**: Teachers' information and their preferences for teaching slots and assigned courses are taken as input.
- **Priority Assignment**: Teachers with higher priority (lower rank value) are scheduled first.
- **Sorting Preferences**: Teachers' preferences are ranked. Sorting preferences ensures that teachers are assigned to their most preferred slots, but it is prioritized by availability.
- **Slot Checking**: The algorithm first checks preferred slots for availability.
- **Assign Teachers**: Each course is assigned a teacher based on the sorted preferences. The routine checks for conflicts (if a teacher is already assigned to another course at the same time).
- **Conflict Resolution**: If the preferred slot is taken, it uses a recursive reassignment strategy.
- **Output**: Final class routine in Excel, color-coded per academic year.

➢ Flowchart:

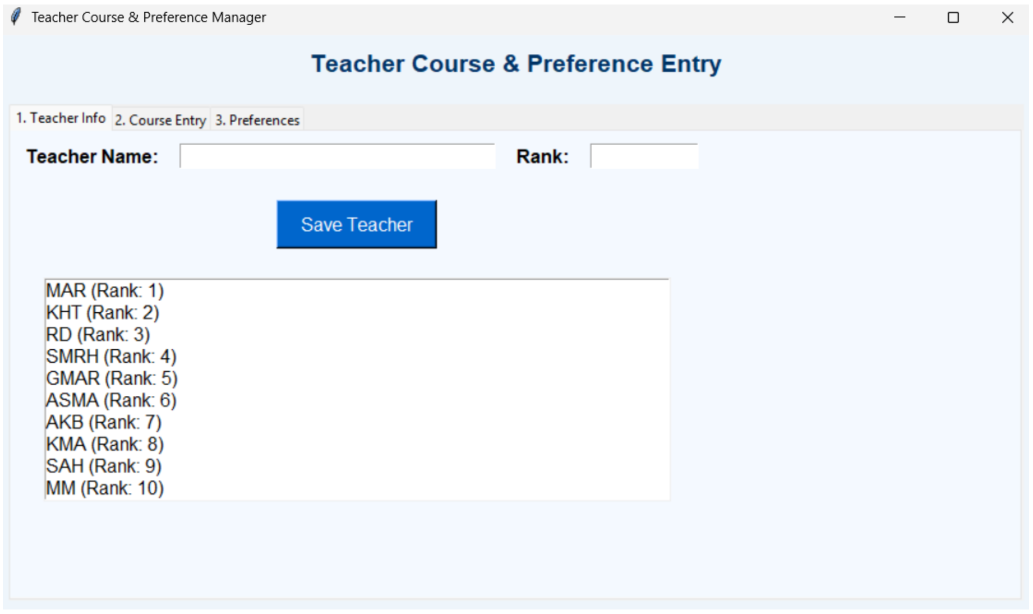

Start
Input Teacher rank, Preferences,Courses
Save Inputs to JSON files
Read Inputs form JSON files
Sort Courses by Teacher's rank
For each Course
Is Sessional?
Yes
No
Try to Schedule Sessional Course
Try to Schedule Theory Course
Is Conflict?
No
Schedule done for this course
Yes
Shift Existing Course
For 2 Times
Possible to Shift?
Yes
No
Is Conflict?
No
Schedule done for this course
Yes
Shift Existing Course
For 2 Times
Possible to Shift?
No
Yes
No
Schedule All Courses?
Yes
Generate Routine in Excel format
End

5. Result Section:

Implementation Outcome:

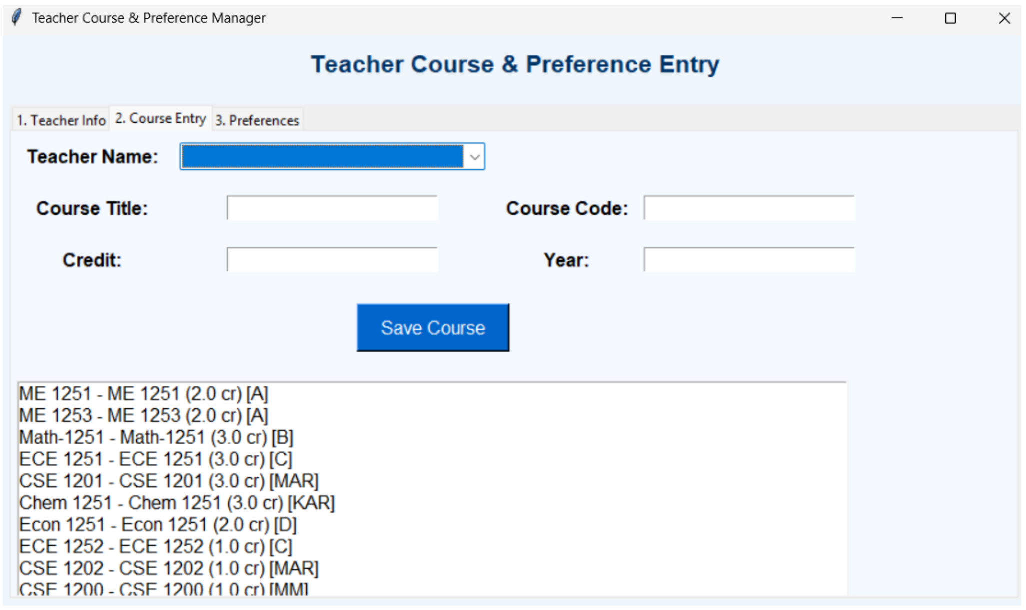
Interface For Taking Inputs:

After taking all the inputs save this information to JSON files for implementation.



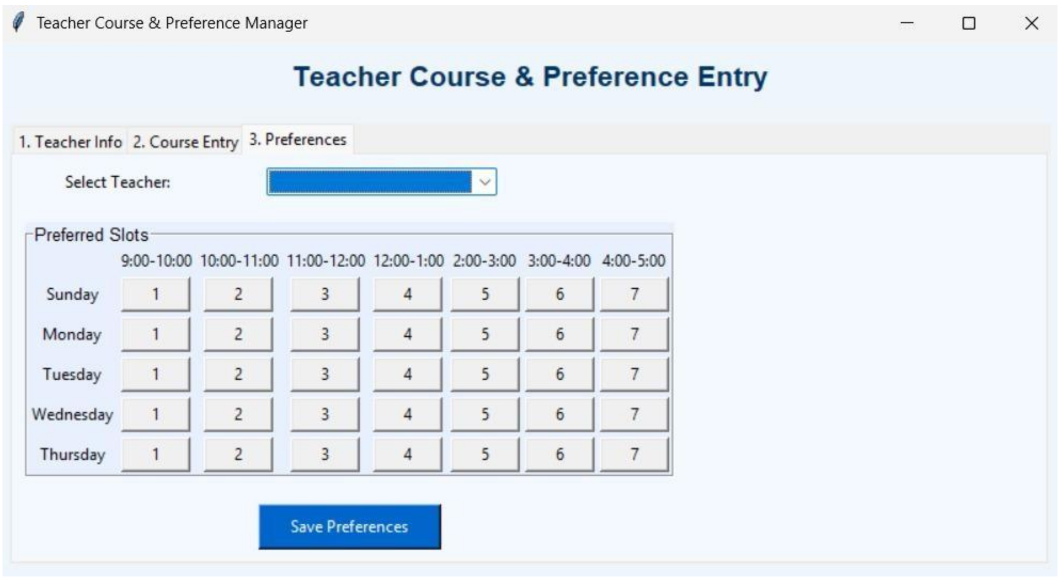
```
MAR (Rank: 1)
KHT (Rank: 2)
RD (Rank: 3)
SMRH (Rank: 4)
GMAR (Rank: 5)
ASMA (Rank: 6)
AKB (Rank: 7)
KMA (Rank: 8)
SAH (Rank: 9)
MM (Rank: 10)
```

Figure 1: Input Teacher List (name and rank)



```
ME 1251 - ME 1251 (2.0 cr) [A]
ME 1253 - ME 1253 (2.0 cr) [A]
Math-1251 - Math-1251 (3.0 cr) [B]
ECE 1251 - ECE 1251 (3.0 cr) [C]
CSE 1201 - CSE 1201 (3.0 cr) [MAR]
Chem 1251 - Chem 1251 (3.0 cr) [KAR]
Econ 1251 - Econ 1251 (2.0 cr) [D]
ECE 1252 - ECE 1252 (1.0 cr) [C]
CSE 1202 - CSE 1202 (1.0 cr) [MAR]
CSE 1200 - CSE 1200 (1.0 cr) [MM]
```

Figure 2: Input Assigned course for teacher(with course details)



	9:00-10:00	10:00-11:00	11:00-12:00	12:00-1:00	2:00-3:00	3:00-4:00	4:00-5:00
Sunday	1	2	3	4	5	6	7
Monday	1	2	3	4	5	6	7
Tuesday	1	2	3	4	5	6	7
Wednesday	1	2	3	4	5	6	7
Thursday	1	2	3	4	5	6	7

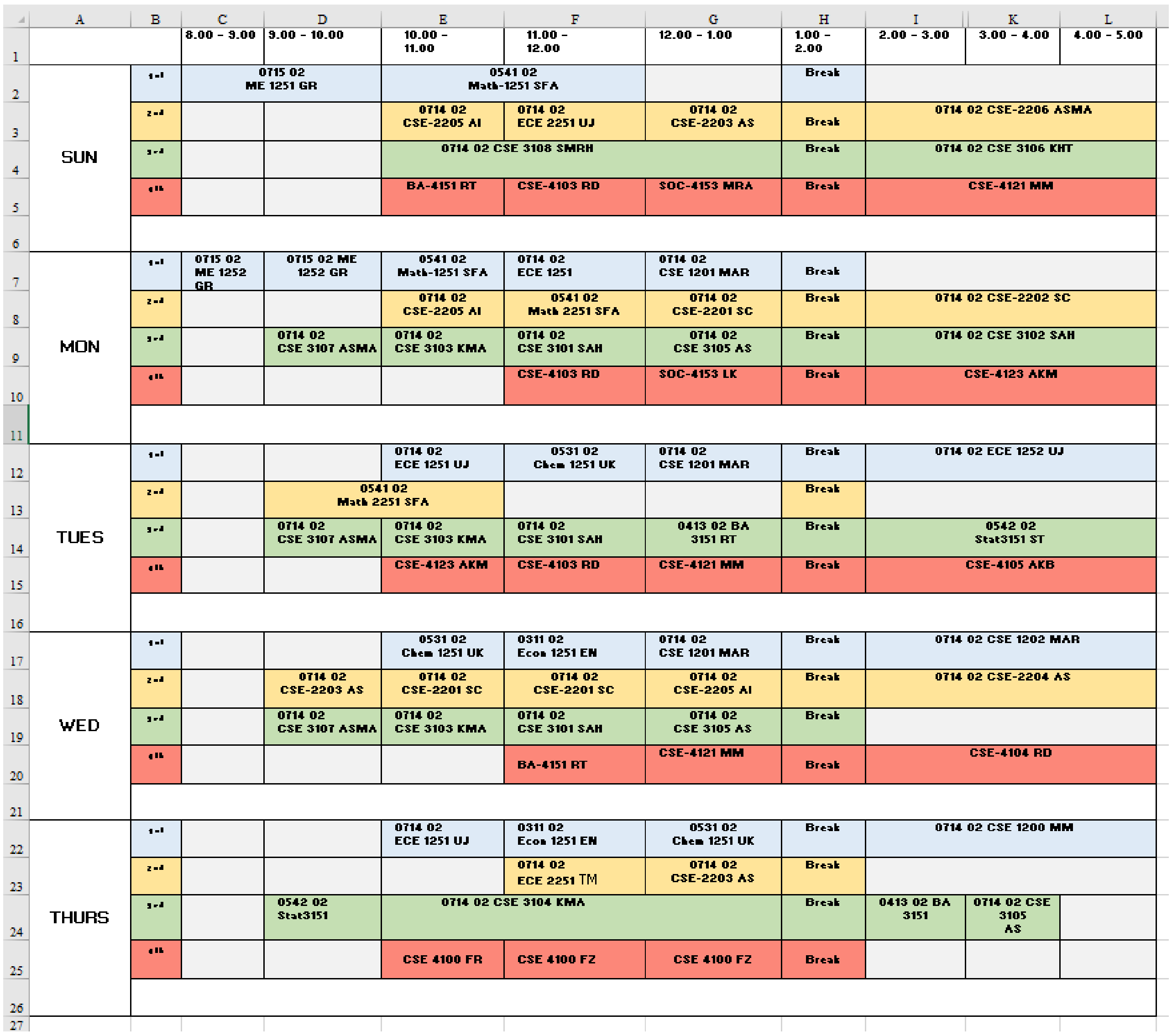
Figure 3: Input Teacher Preference

➢ Output:

A file routine_final.xlsx is generated containing:

- Class routine per weekday and year.
- Each slot includes course code and teacher name.
- Color-coded cells for different years.
- Empty slots highlighted in gray.

Class Routine



		8.00 – 9.00	9.00 – 10.00	10.00 – 11.00	11.00 – 12.00	12.00 – 1.00	1.00 – 2.00	2.00 – 3.00	3.00 – 4.00	4.00 – 5.00
SUN	1st	0715 02 ME 1251 GR		0541 02 Math-1251 SFA			Break			
	2nd			0714 02 CSE-2205 AI	0714 02 ECE 2251 UJ	0714 02 CSE-2203 AS	Break	0714 02 CSE-2206 ASMA		
	3rd			0714 02 CSE 3108 SMRH			Break	0714 02 CSE 3106 KHT		
	4th			BA-4151 RT	CSE-4103 RD	SOC-4153 MRA	Break	CSE-4121 MM		
MON	1st	0715 02 ME 1252 GR	0715 02 ME 1252 GR	0541 02 Math-1251 SFA	0714 02 ECE 1251	0714 02 CSE 1201 MAR	Break			
	2nd			0714 02 CSE-2205 AI	0541 02 Math 2251 SFA	0714 02 CSE-2201 SC	Break	0714 02 CSE-2202 SC		
	3rd		0714 02 CSE 3107 ASMA	0714 02 CSE 3103 KMA	0714 02 CSE 3101 SAH	0714 02 CSE 3105 AS	Break	0714 02 CSE 3102 SAH		
	4th				CSE-4103 RD	SOC-4153 LK	Break	CSE-4123 AKM		
TUES	1st			0714 02 ECE 1251 UJ	0531 02 Chem 1251 UK	0714 02 CSE 1201 MAR	Break	0714 02 ECE 1252 UJ		
	2nd		0541 02 Math 2251 SFA				Break			
	3rd		0714 02 CSE 3107 ASMA	0714 02 CSE 3103 KMA	0714 02 CSE 3101 SAH	0413 02 BA 3151 RT	Break	0542 02 Stat3151 ST		
	4th			CSE-4123 AKM	CSE-4103 RD	CSE-4121 MM	Break	CSE-4105 AKB		
WED	1st			0531 02 Chem 1251 UK	0311 02 Econ 1251 EN	0714 02 CSE 1201 MAR	Break	0714 02 CSE 1202 MAR		
	2nd		0714 02 CSE-2203 AS	0714 02 CSE-2201 SC	0714 02 CSE-2201 SC	0714 02 CSE-2205 AI	Break	0714 02 CSE-2204 AS		
	3rd		0714 02 CSE 3107 ASMA	0714 02 CSE 3103 KMA	0714 02 CSE 3101 SAH	0714 02 CSE 3105 AS	Break			
	4th				BA-4151 RT	CSE-4121 MM	Break	CSE-4104 RD		
THURS	1st			0714 02 ECE 1251 UJ	0311 02 Econ 1251 EN	0531 02 Chem 1251 UK	Break	0714 02 CSE 1200 MM		
	2nd				0714 02 ECE 2251 TM	0714 02 CSE-2203 AS	Break			
	3rd		0542 02 Stat3151	0714 02 CSE 3104 KMA			Break	0413 02 BA 3151	0714 02 CSE 3105 AS	
	4th			CSE 4100 FR	CSE 4100 FZ	CSE 4100 FZ	Break			

➢ Comparison:

Feature	Manual Scheduling	This System
Time Required	High	Low
Preference Satisfaction	Limited	High
Conflict Resolution	Manual	Automated
Reusability/Repeatability	Low	High

6. Conclusion:

The developed class routine generation system automates an otherwise labor-intensive academic task. By integrating teacher preferences and prioritizing assignments through rank-based logic, the tool ensures fairness and efficiency. The Tkinter GUI allows userfriendly input, and the Excel output provides professional and usable formats for administrators. This system offers significant improvements over manual methods and can be further enhanced with room allocation, department-specific constraints, and integration with institutional databases.

7. References:

[1] A. Schaerf, "A survey of automated class routine generation," *I. J. Education and Management Engineering*, vol. 2022, no. 2, pp. 38–48, Apr. 2022. DOI: 10.5815/ijeme.2022.02.05.

[2] A. K. Roy and M. Hossain, "An intelligent class schedule generator using genetic algorithm and backtracking," *Asian Journal of Research in Computer Science*, vol. 13, no. 3, pp. 28–44, 2022. [Online]. Available: https://www.journalajrcos.com

[3] D. Kumar, A. Sharma, and R. K. Chauhan, "A hybrid algorithm for automatic timetable generation," *International Journal of Computer Applications*, vol. 40, no. 15, pp. 6–13, Feb. 2012. DOI: 10.5120/5051-7361.