CS340 - Project: The Registrar's Problem

Amina Ahmed, Joon Luther, Foqia Shahid

September 30, 2022

1 Description

The problem requires handling the issue of how to schedule classes so that students are less likely to have conflicts among the courses they want to take.

The proposed algorithm starts by considering each classroom in order of size, starting from the biggest. For each of the time slots the considered classroom is available for, there exists a corresponding list of students who are available for class during that time slot. The list of available students is then traversed through to find the class that the most amount of students want to take. This can be termed as the most ~~famous~~ popular class. This class is then assigned to that time slot and room, given that the professor who teaches that class is available during that time slot. If the professor is unavailable, the next most famous class is found. Once a class is found that enough students want to take for which a professor is available, the room and the time slot are assigned to the class. The students who prefer to take this class are then removed from the list of available students for that time slot. The class is removed from the student preference list so that it is not considered again for other time slots and rooms. After a time slot and room is assigned to a class, the algorithm moves on to the next time slot and room combination.

you have a popularity based greedy. Do not traverse the student preference list over and over again. Instead, do it once, keep a counter for each class and increment it for each corresponding student preference and go from there.

2 Pseudocode

```
Function schedular()
    for i:1 to classrooms.len do
        for j:1 to timeslot.len do
            c = next most famous class for availableStudents[j]
            while unavailableProfs[j].find(Prof(c)) do
                c = next most famous class
            end
            assignment[i][j] = c
            remove min(capacity[i], # of students to take c) students in c from
              availableStudents[j]
            remove most famous class from Students
        end
    end
    return Schedule
```

how do you determine that?

3 Time Analysis

The algorithm iterates through r rooms for t time slots. This produces $r \times t$ iterations. For each iteration, the following operations are performed:

1. Identify the next most popular class
2. Determine if professor conflict exists
3. Add a class to the schedule

The total run time of these operations comes out to $O(rt \lg s)$. ? how did this term come in?

The input is given as an $sx4$ 2 dimensional array *Students*. Index i of *Students* corresponds to a student s_i and the columns of *Students* contain the preferred classes of s_i. All classes are equally preferred by s_i regardless of order. Looking up a student's set of preferences given the index of the student is $O(1)$.

The final schedule is stored in *Schedule*, implemented as a list of objects *Course*. Each *Course* object has a variable storing course number, time interval, room, professor, and a list of enrolled students. Creating a *Course* object is O(1).

A data structure *ClassCounts* is constructed using *Students*. *ClassCounts* contains a list of class objects c_i. Each c_i includes a list of students interested in taking the c_i. Creating *ClassCounts* takes $O(s)$ time since the algorithm goes through every student.

For each time interval t_i there is a list $UnavailableProfs$ of unavailable professors who are already scheduled to teach at t_i. Creating these lists is $O(tp)$.

For each time interval t_i there is a list *AvailableStudents* of available students who are not scheduled in any other classes c for t_i. In order to find the most famous class, we iterate through the *AvailableStudents* at t_i. We generate a count of each course that appears in students in *AvailableStudents* preferences. We refer to the *Students* array to see the preferred courses. This process is $O(s)$ because there are at most s students and 4 course preferences per student. Counts are stored in an ordered map M where key is course and the corresponding value is count of interested students for time slot t_i. Creating M is $O(\lg s)$. Determining the highest count in M is $O(1)$. If the course with the highest counts in M has an unavailable professor, then the course with the next highest count in M is evaluated. This process repeats until a course can be scheduled with no professor conflict for a total running time $O(c)$. ?

Don't iterate in the loop. Do it once at the get go.

When a class c_i is scheduled for a time interval t_k in room r_j, the max number of interested available students who fit into r_j are enrolled in the class. A new *Course* object is created and added to the *Schedule* list. The algorithm iterates through every element of the list in *ClassCounts* corresponding to c_i. For each student in this list, the algorithm looks up the student in *Students* and deletes their preference for class c_i in the array (like setting the

this is at least O(s)

and your total is at least O(rts), as written.

entries in the array $= c_i$ to null). The deletion process runs in $O(s)$ since there are at most s students enrolled in c_i and 4 preferences to go through per student. *AvailableStudents* is updated to exclude the students who were just enrolled in c_i, taking $O(s)$.

After a class c_i is scheduled at time t_i, *AvailableStudents* for the interval t_i is updated to exclude all students who were just enrolled in c_i. This process is $O(s)$.

4 Proof of Correctness

Proof of termination: The algorithm runs for r*t iterations after which it will terminate since there are no more combinations of rooms and time slots to look at.

Proof of Validity: To prove that the schedule produced is valid, we want to ensure the following:

1. no teacher conflict
2. no room conflict
3. All schedulable classes are scheduled i.e, no empty rooms and slots
4. No classes are scheduled more than once
5. No enrolled students has a schedule conflict

By construction, (3) is true. All rooms are filled for all the time slots and all time slots are filled for each classroom. We are ensuring this since our outer loop runs for each classroom and the inner loop runs for each time slot and the assignment is done for each combination of classroom and time slot.

For (4), since are maintaining a data structure that contains student to course preferences and removing the most famous class after assignment from this list, we ensure that that class is never considered again.

For (5), we ensure no enrolled student has a schedule conflict by removing them from our list of available students for that given time slot. We're doing this at every assignment. Given that we're going to assign class c (as it is the most famous class), we then find the minimum of number of students who want to take c and the classroom capacity in which we are putting c to get the number of students we want to remove from the next consideration for that given time slot.

(2) is ensured by construction since for each classroom, we put a class in a distinct time slot and only move to the next classroom once the distinct time slots are filled for that classroom. Thus for the next classroom, all the time slots are available.

(1) is ensured by checking that the professor of the class we are thinking of assigning to a time slot isn't already teaching in that time slot. We do this in the while loop by finding the professor who is teaching class c and then checking if that professor is already in a list of professors for that time slot, $timeSlotProfs[j]$ where j is the time slot j.

5 Discussion

When designing this algorithm, our main approach was to solve some of the conflicts by construction which we achieved for classrooms and time slots. The natural complications that arose were how we could ensure the professors didn't have conflicts and neither did the students. With the professors, it was an easy problem of solving since it just involved keeping track of which professors were teaching in a given time slot and making sure we don't assign the same professor twice. However, things got more complicated with the students. To solve the conflict problem for students, we decided to keep track of the students that are assigned in each time slot and removing those students who are taking classes in that time slot as we go. Thus, in our consideration of the next most famous course, we would only consider the available students. The algorithm was hard to design since there were many of variables to keep track of and special cases like when the same professor is teaching the top most famous classes. Our algorithm uses a greedy approach which is to choose the most famous class for the given set of available students. Our approach was similar to the one we took in homework problem 2 since we set a criteria and ordered our options based on that.

popularity-based approach has the streigth of being simple, but more popular doesn't mean more conflicted. [redacted] A Conflict only arises if the same student / many students want to take the [redacted] two classes in the same time slot. Yours may not have a good fit because it doesn't address conflict directly.