CS340 - The Registrar's Problem

Alana Burgess, Synarah Sitaf, Julia Rieger, Ramla Mohammed

October 2023

1 Description

For all courses in C, the set of courses, we store a preference value q. For all students in S, the set of students, we iterate through their preference lists and increment the preference value of each class by 1.

We start by creating a two dimensional conflicts matrix, where classes that are taught by the same teacher are set to infinity. This ensures that those classes aren't scheduled at the same time.

To populate the schedule, go through the matrix and put the course with the most conflicts together in different time slots. Do this by first putting the courses in the room of each time slot. Then, to put the next courses into their rooms, check the room with the last class placed in each time slot, and put the new course into the time slot where it conflicts the least with the last class placed in that time slot. Then, when all the courses are placed into time slots, sort the courses by their preference value so that the most preferred class is in the classroom with the biggest capacity and the least preferred class is in the classroom with the smallest capacity.

To place students into classes, go through the first preference of each student and try to place them into that class. If that class is not full, put the student in that class and increment the count of students that have gotten a class they want. Go through the second preference of each students and place the student in the class if the class is not full and this class doesn't conflict with the other class the student is in. Again, increment the count of students that got a preferred class as you go. Do this for all the students third preferences and all the students fourth preferences, checking that the class isn't full and doesn't conflict with another of the students class, and increment the count of students that get the class they want when needed, until you get through all the preferences. Go through each student that hasn't been placed into four classes and try to place them into each class in the group of classes until a class that they fit in and that doesn't conflict with another one of their classes until all students are placed into four classes.

Return the schedule and the count of the number of students that got the classes they preferred over the total number of classes on the student preference list.

2 Pseudocode

Course = stores courseId, prefVal, timeslot, Room, Students in class

Room = stores roomId, capacity

Teacher = stores teacherId, course1, course2

allTeachers = array of p Teachers

allCourses = array of c Courses

allRooms = array of r rooms

stuPrefList = 2d array storing s students + their preferred Courses

Conflicts = int matrix of conflict values between each course

Schedule = txr Course matrix indexed by [timeslot, room]

Files: Main.java, LookupScheduleInfo.java, BuildSchedule.java, Course.java, Room.java, Teacher.java

```
Function makeScheduleRandomData(constraints.txt, studentprefs.txt)
    for all s in S > studentprefs.txt do
        for each class in s.plist do
            update class.q
            update conflicts matrix
        end
    end
    Sort classes in order of highest class.q value
    Sort R > constraints.txt by largest capacity
    Create schedule[timeslots][rooms]
    while schedule is incomplete do
        find class pair (i,j) in conflict matrix with highest conflict
        place i in timeslot x who's last class has the least conflict with i and
          i.teacher is not yet assigned
        place j in timeslot (not x) who's last class has the least conflict with j and
          j.teacher is not yet assigned
    end
    Sort each timeslot so the classes with higher class.q values are placed in the
      higher capacity rooms
    for $allsinS > studentprefs.txt$ do
        for each class in s.plist do
            attempt to register s to class
            if class is full or conflicts then
                increase count i
            end
        end
        if $i > 0$ then
            register s to the i classes with lowest class.q values
        end
    end
```

3 Experimental Analysis

3.1 Time Analysis

Theoretical Time Analysis (to be explained in further detail in section 5: Time Analysis) is as follows:

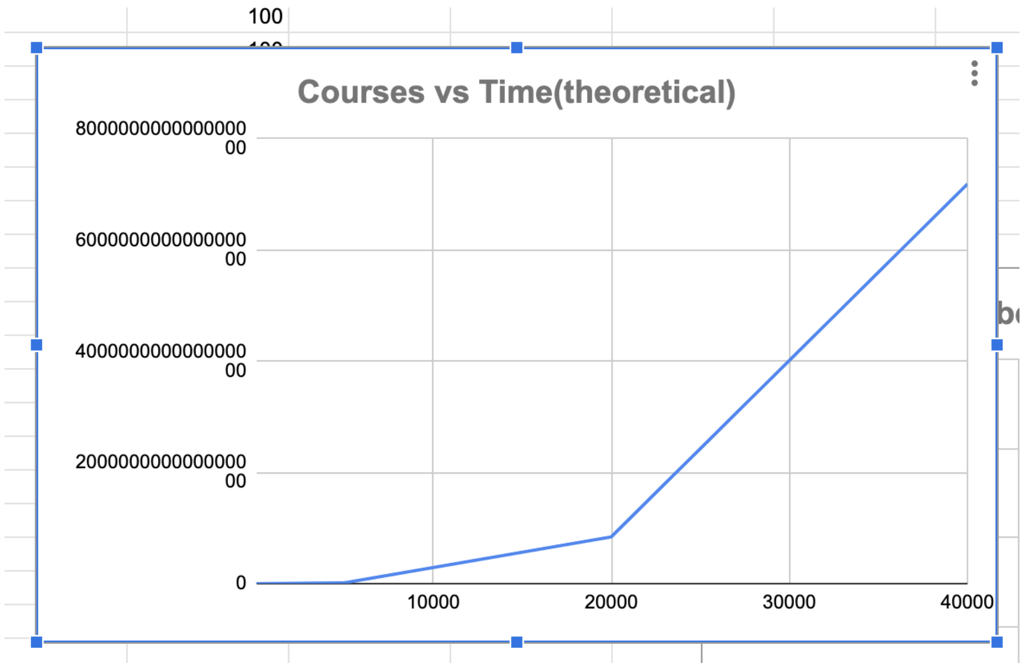


Figure 1: Time by Number of Courses Theoretical

Practical Time Analysis is as follows:

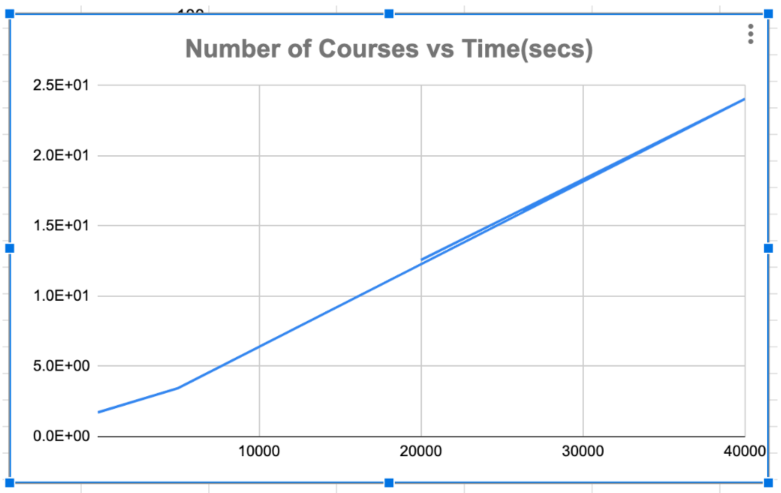
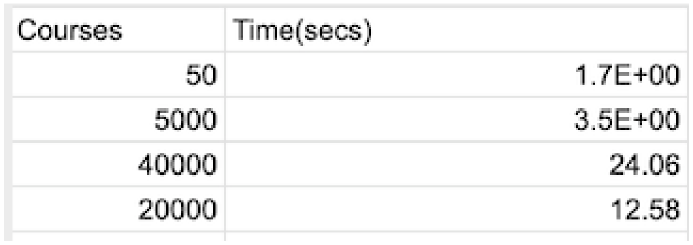


Figure 2: Time by Number of Courses Experimental

While the tests we did verifies our algorithm performs as expected based on our theoretical analysis, with more time a better analysis could be computed with a series 10-20 tests on the randomly generated data, instead of the 4 we did. The largest set of courses we tested was 40,000. It is noteworthy that at 80,000 courses, there was not enough Heap memory to complete the program run.

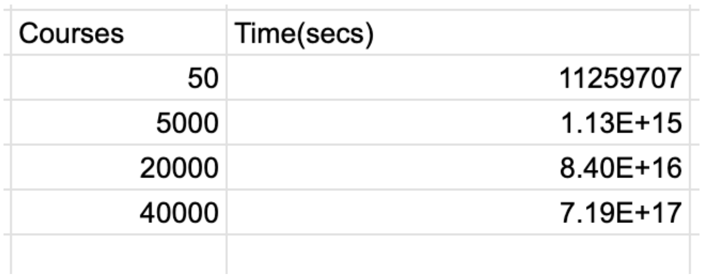
The program was tested over the following values:



Courses	Time(secs)
50	1.7E+00
5000	3.5E+00
40000	24.06
20000	12.58

Figure 3: Table of Time by Number of Courses Experimental

The Theoretical Time Analysis was computed over the following values:

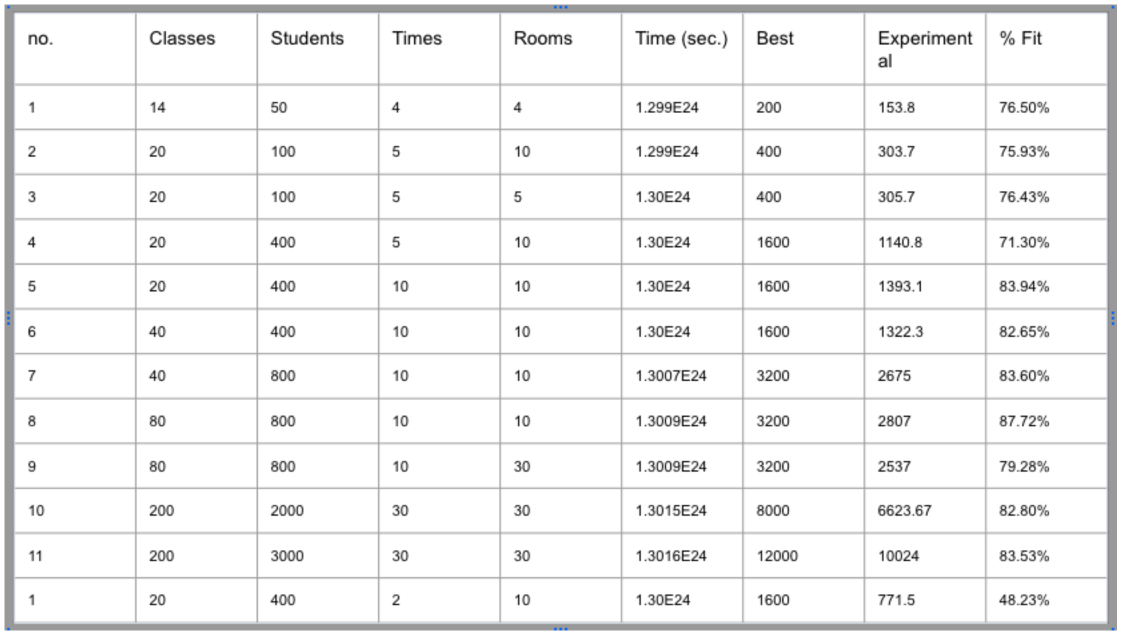


Courses	Time(secs)
50	11259707
5000	1.13E+15
20000	8.40E+16
40000	7.19E+17

Figure 4: Table of Time by Number of Courses Theoretical

3.2 Solution Quality Analysis

In the best fit, we had a fit of .8772. In the worst fit we have .4823. The lower bound is 54.98 percent of the upper bound. Thus the algorithm is always able to achieve at least 54.98 percent of the best case student value. This is based on the table below.



no.	Classes	Students	Times	Rooms	Time (sec.)	Best	Experimental	% Fit
1	14	50	4	4	1.299E24	200	153.8	76.50%
2	20	100	5	10	1.299E24	400	303.7	75.93%
3	20	100	5	5	1.30E24	400	305.7	76.43%
4	20	400	5	10	1.30E24	1600	1140.8	71.30%
5	20	400	10	10	1.30E24	1600	1393.1	83.94%
6	40	400	10	10	1.30E24	1600	1322.3	82.65%
7	40	800	10	10	1.3007E24	3200	2675	83.60%
8	80	800	10	10	1.3009E24	3200	2807	87.72%
9	80	800	10	30	1.3009E24	3200	2537	79.28%
10	200	2000	30	30	1.3015E24	8000	6623.67	82.80%
11	200	3000	30	30	1.3016E24	12000	10024	83.53%
1	20	400	2	10	1.30E24	1600	771.5	48.23%

Figure 5: Fit by Generated Data Set

4 Bryn Mawr Scheduling

We tried to implement 5 extra constraints on our code. 1) We implemented a constraint that student preferences could range from 1 to 5. 2) Room, Teacher, and Course Ids all had to be set up so that they could be placed in an array correctly. 3) We implemented it so that teachers can teach 1-5 classes, or however many they want 4)Each teacher is placed in a different department and Course stores that department- influences building/rooms 5) We implemented a constraint that Rooms could only be used for classes in specified departments and so then classes could only be placed there during specific time slots where those classrooms were open

1- The student preference list is made into a two dimensional array where the inner array is an array list which had a modifiable length. As the student preferences are parsed, varying number of inputs are used to created array lists of varying sizes, so each student can prefer up to 5 classes

2- The Room, Teacher, and the Courses all have integer ids that are way higher than their indexes. This is handled with a counter int that increments every time a room, teacher, and course is created. These are then indexed by their respective counters.

3- Each teacher has an array list of classes instead of just having two classes. This makes it so the teacher can have any amount of classes. When the course is parsed with the teacher that teaches the course, it is added to the Array list of courses the teacher teaches. This is then used to set the conflict matrix up, and make sure the classes the teacher is in aren't at the same times. Because the teacher can have more than 2 classes, their needs to be an extra check that the class isn't being put into a time slot with a class taught by the same teacher.

4 and 5) We created a department class to organize teachers, courses and rooms by departments. Each instance of department has a list of classes taught in that department, a list of teachers in that department, and rooms in that department. As the list of courses and teachers is parsed, the department of the course and teacher are updated and the lists of courses and teachers for each department are updated. When courses are placed into the schedule, the list of rooms they could be placed in within each slot is looked at. This means that within the schedule, if two classes are found to be the next most conflicting classes, they can be placed in two time-slots but only time slots where the rooms in the department can be placed are open. This involved some three dimensional data structure that took account of the department, time slot, and rooms so that we could optimize for the putting courses into time slots they least conflicted with and putting courses into time slots where rooms would be available.

4.1 Data Structures

The student preference list is a two-dimensional array where the first array indexes the students, and each student has an array that stores their preferred classes.

The conflict matrix is a two-dimensional array where the first dimension is the first class and the second dimension is the second class and the integer stored at the intersection of the classes is how many times the two classes show up together in the student preference list.

We then create a two-dimensional array/matrix where the rows represent time slots and the columns represent the rooms. Initially all of the time slots and rooms are set to available. The time complexity for building this matrix is O(T*R), and it allows us to check the availability of specific timeslot-room combinations in O(1) and mark such combinations as available or unavailable.

The teachers, courses, and rooms were all held in respective arrays. A Course data structure stores a courseId, prefVal which is how many students have it on their preference list, and which timeslot, Room, and Students are in class. The teacher has the courses they teach. The room has the capacity and roomId.

Lastly we create another dictionary to track registration. The keys of this dictionary is the class id and the values are the lists of students registered to that class. This allows us to efficiently add a student to a class, remove a student if they decide to drop a class, and check whether a student is registered for a specific class and takes O(C) to build.

5 Time Analysis: Basic Code

S: set of all students, $s = |S|$

C: set of all classes, $c = |C|$

R: set of all rooms, $r = |R|$

T: set of all timeslots, $t = |T|$

P: set of all professors, $p = |P|$

readConstraints (creates allTeachers, allRooms, conflicts) $O(c^2 + p + r)$

readStuPrefs (creates stuPrefList, allCourses) $O(4s)$

placeCoursesInTimeslots $O(c^3 2t)$

placeCoursesInRooms $O(cclogcrlogrt + tr^2)$

registerStudents $O(st + 4s + c + sc)$

outputSchedule $O(t + sc)$

Analyzing this, depending on the differences in proportions of the data we can conclude the worst-case time complexity to be $O(c^3)$ or $O(c^2 logcrlogr)$. The former comes from placing all courses in timeslots, in which technically we must analyze this as c^3 even though in reality it will probably not occur since the outer loops of searching through the conflicts matrix will likely run a more similar to c than c^2. Because $c \leq rt$, it is more likely for $O(c^2 logcrlogr)$ to be the relevant time complexity. If the value of r is closer to the value of c, this could be worse than c^3. We can see these conclusions more clearly in our Experimental Analysis section 3.1.

6 Proof of Correctness

$ProofofCorrectness$. To prove that this algorithm works, six parts need to be shown.

1. proof of termination

2. proof that no teacher is teaching two classes at the same time
3. proof that every student gets four classes, each in a different time slot
4. proof that there won't be more students in a room than capacity
5. proof that class will be assigned to only one room at only one time slot
6. proof that student only has one class per slot

Proof of termination:

The algorithm begins by iterating over all students in S and their preferences in student-prefs.txt. The loop processes each student and their preferences only once, and since there are only n students it will terminate after n iterations. After processing student preferences, the algorithm proceeds to sort the classes in constraints.txt by their largest capacity. This will terminate once all classes are sorted. The code then initializes a schedule and enters a loop that continues until the priority queue Q is empty. Inside this loop, the algorithm attempts to find an available timeslot and room for the class a. It ensures that the teacher is not yet assigned, and it assigns the class to a timeslot and room. The code does not enter an infinite loop since it selects available timeslots and rooms based on the class's requirements and the current schedule. If i is greater than zero, the code attempts to register students to the i classes with the lowest class.p values. This loop runs until i=0 meaning the student is unable to register for other classes

Proof that no teacher is teaching two classes at the same time:
Suppose that a teacher is teaching two courses at the same time. This would mean that after going through the conflict matrix that the two courses which the teacher is teaching are scheduled for the same time. This would not be possible because before scheduling two courses at the same time the code checks that the conflict value at this point is not infinity and if it is then the courses would not be scheduled at the same time. Thus two courses with the same teacher can not be scheduled at the same time. For Bryn Mawr Conflicts: Suppose that a teacher is already assigned a time slot in the given schedule in one class room. Then suppose that the teacher is assigned another class room in the same time slot. This would mean that a teacher would be teaching two classes in the same time slot. Every time a teacher is assigned a time slot the row of time slots is checked before assigning a time slot. In this row is all the other classes being taught during this time slot in all the other classrooms. Thus this is a contradiction that a teacher can be teaching two classes at the same time. Thus no teacher can be teaching two classes at the same time.

Proof that every student gets four classes, each in a different time slot:
Suppose that this isn't true and that there is a student that couldn't get into any class offered at a specific time slot. This would mean that the student either couldn't get into the preferred class they wanted to in that slot, or they didn't prefer any of the classes in the time slot. Then, the algorithm searched through every available class and no class

they could take was available (meaning that an open class remained but was in a time slot where 1) they already had a class, or 2) the classes that were in the time slot they didn't have a class in were already full). But this is a contradiction because 1) we know that the student had no other class in this time slot to conflict with any class, so if there is an available class there will be no conflict and 2) there has to be an available class because the addition of all the classroom capacities is greater than the number of students.

If it is the case that a student is unable to get into the class that they want and all other classes are full except for one in a time slot that they already have a class in then we go through the students and find one that was also placed in another class after being kicked out of a class they preferred. If switching these students makes it so that they are now both in 4 classes that don't overlap then these students are each placed in the new respective classes. If this conflicts with either of the students then a new overflow student is chosen until a valid switch is made. There will have to be one valid switch in the list of students.

Thus all cases are considered and each student will get four classes each in a different time slot.

Proof that there won't be more students in a room than capacity:
Suppose that a class is assigned to a room which has a certain capacity. Supposed that there are more students that prefer a certain class than the available capacity for the class. Supposed that then more students are enrolled in a specific class than available space in the class. Each time a student is assigned to a class there is an if statement inside the for loop which checks whether the class is full or weather this conflicts with another time in this time slot that the student is already registered for. If it does then the student is assigned to a class that has not yet reached full capacity, and that is not at the same time of one of its other classes. This contradicts that more students than the capacity of a class can be enrolled in a certain class. Thus there will not be more students in a room than the capacity.

Proof that class will be assigned to only one room:
Suppose this is not true and a class is assigned to two rooms. This is a contradiction because the algorithm only assigns one class to one room at a time, and no class is assigned to a room twice because the class is only looked at once, when it's accessed from the class array, and then never looked at for assigning again.

Proof that class will be assigned at only one time slot:
Suppose this is not true and a class is assigned to two time slots. This is a contradiction because the algorithm only assigns one class to one time slot at a time, and no class is assigned to the time slot twice because the class is only looked at once, when it's accessed from the class array, and then never looked at for assigning again.

Proof that student only has one class per slot:
Suppose this is not true and a student is assigned to two classes at the same time slot. This

means that the student was assigned to a class at a specific time slot and then somehow was assigned to a different class at the same time slot. This would only occur if the student preferred two or more classes that were at the same time. But this is a contradiction because the algorithm checks if a student has already been assigned a class in that time slot before it assigns the student to a class in that time slot. The algorithm will never find that a student has already been assigned in that time slot and then assign it to another class in that time slot.

Suppose than that after the student is placed into all the preferred classes it can be, the student is placed into a class that is in the same time slot as a class it was previously placed into. This is a contradiction because the algorithm always checks if the student was already placed in that time slot before putting it in a class, and will only put the student in a class at a timeslot the student is already in.

7 Discussion

Why did you chose this algorithm?

We chose this algorithm because in the beginning of class we went over greedy algorithms and so we started out with that idea. We thought about only prioritizing popularity but soon realized that we would have too many conflicting classes happening at the same time of we only looked into popularity. So we shifted into looking at popularity and creating a conflict matrix and making a schedule prioritizing them both. We also looked into a dynamic programming algorithm but struggled to figure out how to fully implement so we decided to continue with a gr eddy algorithm.

Last checkpoint we thought of using the conflict matrix and so when we talked about it in lab we decided to try it out. This is still greedy, we are prioritizing putting the classes with the most conflict with each other in different times slots because it resulted in a higher student point value than only placing courses in time slots by preference. We included sorting each time slot in accordance to room size and course preference because this also resulted in a higher student point value on average, alongside keeping track of the last course added to each timeslot in order to compare conflict values when choosing a timeslot to place a course in.

What complications did you encounter while creating it?

We had a majority of our complications from coding the randomly generated data programs before considering the constraint data. This made it difficult to code and led to many bugs because we had to re-do much of our initial program structuring. Coding the constraints was clearly difficult because there were more moving parts and attributes of objects to consider.

What characteristics of the problem made it hard to create an algorithm for?

While all of the greedy algorithms that we have done before just optimize for one dimension, this problem has so many different dimensions that it is hard. If we could just put every student in their preferred class that would be easy but we have to accommodate for classes being in the same time slot, classes being taught by the same teacher, class room size, and having a certain number of times slots. It also became more of a challenge when we had to take into account real Bryn Mawr data because there were even more constraints that had to be taken into account.

What algorithmic category or categories does your algorithm fall into?

This algorithm is greedy for both making the class scheduling and the student scheduling. When the classes are scheduled, they are scheduled by first conflict and next preference.

What algorithms that we've studied is your algorithm similar to?

It is somewhat similar to interval scheduling. Instead of the classes being ordered with end time, they are ordered by how much they conflict, so that the classes with the most conflicts are put in different time slots and can not overlap. It's different because there is more to optimize.