Group Project: The Registrar's Problem

CS 340, Fall 2016
Prof. Sorelle Friedler

Deadline: November 14th, 12:45pm

1 Problem Description

You will be considering the issue of how to schedule classes so that students are less likely to have conflicts with the courses they want to take. To study this issue, you'll start by considering how to schedule classes *after* students select their preferences. The registration process will ask students to select 4 classes that they would like to register for (with all classes assumed to be equally important). The registrar's office (you) will then take these preferences and create a schedule that allows as many students as possible to take most of the classes they want.

Specifically, the registrar's office needs to create a schedule that handles the following issues:

1. **Classes**: A list of classes is given. Every class must be scheduled in one room, for one time slot, with one teacher, and with a list of enrolled students.
2. **Room sizes**: A list of classroom sizes is given in terms of the number of students the classroom can hold. A class scheduled in a room may not have more students enrolled than will fit in the room. Only one class can be scheduled in the room at a given time.
3. **Class times**: A number of possible class times is given. These are assumed to be simple non-overlapping slots.
4. **Teachers**: A list of teachers and names of the classes they teach is given. Each teacher teaches two classes (so the number of teachers is exactly half the number of classes, which must be even). No teacher may teach more than one class at a given time.
5. **Student preferences**: A list of students and corresponding class requests for four classes is given. No student may be scheduled for more than one class that meets at the same time.

A schedule is considered optimal if it meets all of these constraints and the best value for student preferences satisfied is achieved. The value of the student preferences is calculated by awarding 1 point for each class the student is enrolled in. The total point sum over all students is the student preferences value achieved. Clearly, the maximum such value is 4 times the number of students.

1.1 Your Tasks

You are *not* required to create an algorithm that achieves the optimal schedule for the given. Instead, your goal is to create an algorithm that consistently returns a schedule that is not "too far" from the optimal, and that runs "quickly." Quickly, in this context, means that given 1000 students preferences, your algorithm should be able to run seemingly instantaneously. The method of comparison to the optimal will be described in more detail in Section 3.2. The main portions of this project are as follows and will be described in more detail in the rest of this document:

1. Design the algorithm.
2. Analyze the algorithm's running time.
3. Do an experimental analysis to consider the quality of the schedule your algorithm creates.
4. Write-up these results.
5. Present these results and do a demonstration of your program for your classmates.

Meeting these expectations will ensure that your group gets a C for the project. In order to get an A or B, you will need to do some additional work, described in Section 4. If you plan to do these tasks, you will likely find it useful to work on them from the beginning of the project and not wait to do them all at the end.

To ensure that your group does not wait until the last minute, which would be disastrous, there will be two project checkpoints during this process. Meeting these checkpoints will account for 20% of your project grade. The rest of the grade will be divided approximately as follows: Presentation - 10%, Paper - 70% (Algorithm description, analysis, and discussion 50%, Experimental analysis 20%).

2 Programming Guidelines

You may choose to write the program implementing your algorithm in any programming language of your choice. Keep in mind that some programming languages are inherently faster than others, and that the programming language you choose should complement your algorithm design. Your program should take as input a file describing class, room, time, and teacher constraints and another file describing student preferences. Sample constraint and student input files are provided in the files `constraints.txt` and `student_prefs.txt` respectively. Your algorithm should output a file called `schedule.txt` containing the created schedule in the same format as the provided sample file. In addition, your algorithm should print the calculated student preference value achieved in the form:

```
Student Preference Value: XXX
```

Where XXX is the final value. On the line following that, you should also print the best case student value - 4 times the number of students, or the student preference value if all students are enrolled for all their requested classes:

```
Best Case Student Value: XXX
```

Since you may write in any programming language, you must provide a file called `run.sh` that when called with "`sh run.sh class_info.txt student_prefs.txt schedule.txt` takes the two input files, runs the algorithm, and outputs to `schedule.txt`. The given `is_valid.pl` script will check to make sure that your output is valid.

3 Paper Guidelines

The paper should be as long as it takes to sufficiently respond to the issues in the following two sections. This is likely to be between 5 and 10 pages, with additional pages needed depending on the number of additional tasks attempted. It should be typed and a *single* hard copy (i.e., one for the entire group) should be handed in by the deadline.

Your paper should also include a one to two paragraph *abstract* that emphasizes your findings (but not, in detail, how you came to them). This should be readable to someone who was not in the class, so that I could send it to the registrar as a set of recommendations!

3.1 Algorithm Description, Time Analysis, and Discussion

You should write-up a description and analysis of your algorithm as described in the "Algorithm Write-up Guidelines," except that the proof of correctness should only show that the schedule created is valid, not that it is also optimal. You should also include a discussion of your group's algorithmic choices. Consider answering some of the following questions: Why did you chose this algorithm? What complications did you encounter while creating it? What characteristics of the problem made it hard to create an algorithm for? What algorithmic category or categories does your algorithm fall into? What algorithms that we've studied is your algorithm similar to?

3.2 Experimental Analysis

Your group should also perform and write-up an experimental evaluation of your algorithm's performance with the goal of understanding how close to optimal the created schedule is. You have been provided with a random constraint and student preferences generator: `make_random_input.pl`. You should use this program (and any modification of it that you care to make) to do the following two analyses:

1. **Time Analysis**: Verify that your algorithm performs as expected based on the theoretical time analysis (included in the write-up described in Section 3.1). Describe your experiment design and explain how the resulting numbers verify the time analysis.
2. **Solution Quality Analysis**: The goal of this analysis is to compare your solution to the optimal, without needing to determine the optimal in all cases. You may use the best case student value as an upper bound on the optimal value. Use the random instance generator to experimentally justify that your algorithm achieves some lower bound on the optimal value. This lower bound should be expressed as a fraction of the upper bound. For example, you might say that your algorithm is always able to achieve at least half the best case student value. For the quality analysis, you may also find it useful to modify `make_random_input.pl` to generate instances in such a way that it can also generate the optimal student preferences value or a known solution value for comparison with your algorithm's output.

4 Haverford scheduling

In order to receive an A or B on this project, you must also consider how this scheduling question relates to the schedule at Haverford. By looking at the real Haverford scheduling information, you should come up with **recommendations** for how the schedule could be changed so that even when it is created *before* students register, more students could receive the classes they'd like to be in. These recommendations should be *implementable* by the college.

In order to receive a B on this project, your group must successfully study at least 2 additional questions. In order to receive an A, your group must successfully study at least 5 additional questions. You may want to check with the professor to be sure your additional tasks are sufficiently difficult. You should still be able to generate a valid schedule and should be sure to include a description of how you handled these tasks. You may choose whatever input format you would like for these modifications to the problem (as long as the previous unmodified version of your algorithm still works). For each of these modifications you should not only explain how you were able to create an algorithm (or modify your existing algorithm) to handle the modification, you should also give an analysis and recommendations for the Haverford schedule based on your modification.

5 Presentation Guidelines

As part of your final presentation, you will be required to demonstrate your code on a given input instance. Therefore, your code must be fast enough to run during a demonstration! In addition to this demonstration, your presentation should include the information from the paper at a high-level. All group members should be equally involved in the presentation and it should last no more than 10 minutes.

6 Checkpoints and Deadlines

6.1 Checkpoint 1: September 29th at 2pm

By the first checkpoint, your group should have decided on an algorithm and written the algorithm description, analysis, and discussion section of your paper. A rough draft of this section should be printed out and submitted in class. Since this deadline is early, you will only have covered greedy algorithms and will likely need to take a greedy approach to this problem. That's ok! Greedy approaches will still vary from group to group and can be very successful.

6.2 Checkpoint 2: October 20th at 2pm

By the second checkpoint, your group should have programmed your algorithm in your chosen programming language. The code should be complete enough so that it can run on a given instance. If possible, you should additionally have started the experimental analysis and appropriate debugging of your code. The code should be committed and you should make sure that it passes `is_valid.pl`.

6.3 Final Deadline: November 14th at 12:45pm

The full write-up is due as a printed-out copy in class. The class period and discussion section will be spent giving presentations on your algorithm and a demonstration of your code. The final version of the code should be committed by 12:45pm as well. **No late projects will be accepted.**

7 Some Questions Answered

Anticipating some questions you may have as you start working on this, here are some answers.

- *If more students sign up for a class than can fit in the largest room, are multiple section of that class created?* No.
- *Are there ever sections of a class?* No. These classes are "simple classes," like ours, with exactly one teacher and some number of students that all fit into a single room.
- *What about the corner cases cases when there might be more students than could actually ever be scheduled or more classes than could be scheduled?* See the code for `make_random_input.pl` to see what I'm assuming. As long as your solution works for any instance that `make_random_input.pl` generates, it is complete enough. (For this project we care most about average inputs, and less about pathological ones.)
- *What if a class isn't scheduled?* How do I include it in the output schedule? Just don't include that class at all.