CMU Spring 2022 Scheduling

21-393 Operations Research II Project

Stephanie Erickson, Will Powell, Siqi Zeng, Chaeyeon Ryu, Laya Venkatesan

Abstract

This paper seeks to find a scheduling of courses for the Spring 2022 semester at Carnegie Mellon University (CMU) using a greedy algorithm. Specifically, we look to schedule all courses for the full semester, mini-3, and mini-4 while satisfying space, time, and academic constraints. We output 10 schedules, one for each day of the week for each mini-3 and mini-4, containing all full semester courses for that day along with the respective mini courses. We schedule both undergraduate and graduate courses with a daily time span of 13 hours, consistent with courses running from approximately 8:00am - 9:00pm as CMU does.

Data

In order to find an optimal scheduling, we look to several main sources of data. The first is data from the registrar, which is used to gather all available courses along with their respective max enrollment, department, duration, and instructor. We also use classroom data from the registrar that includes building, room number, and max capacity. The next set of data comes from Stellic, a degree-auditing pathway which is used to determine which courses should not overlap as they are supposed to be taken in the same semester (academic conflicts).

Registar Data

Course Schedule

S22_Registrar_Schedule_Courses_1nov21

From the registrar, we were able to obtain data for the schedule of classes for Spring 2022. This data includes college, department, title, course number, section, if the course is a mini, course level (undergrad vs grad), how the course is taught (IPE or Remote), max enrollment, Registrar Schedule, days the course is taught, Begin Time, End Time, instructor's, instructors' Andrew ID, CALENDAR_ID, Calendar Name, if there is a Final Exam, room group, and "ACT ENR GROUP". Of this data, we ignore the start and end times as we look to create our own schedule and instead focus on the days of the week, and max enrollment to determine when and where to schedule a course. After data cleaning and manually merging cross-registered courses, we were able to extract these two excel files ready to be used in our main function.

Final data:
Full Semester Courses
Mini Courses

Classroom

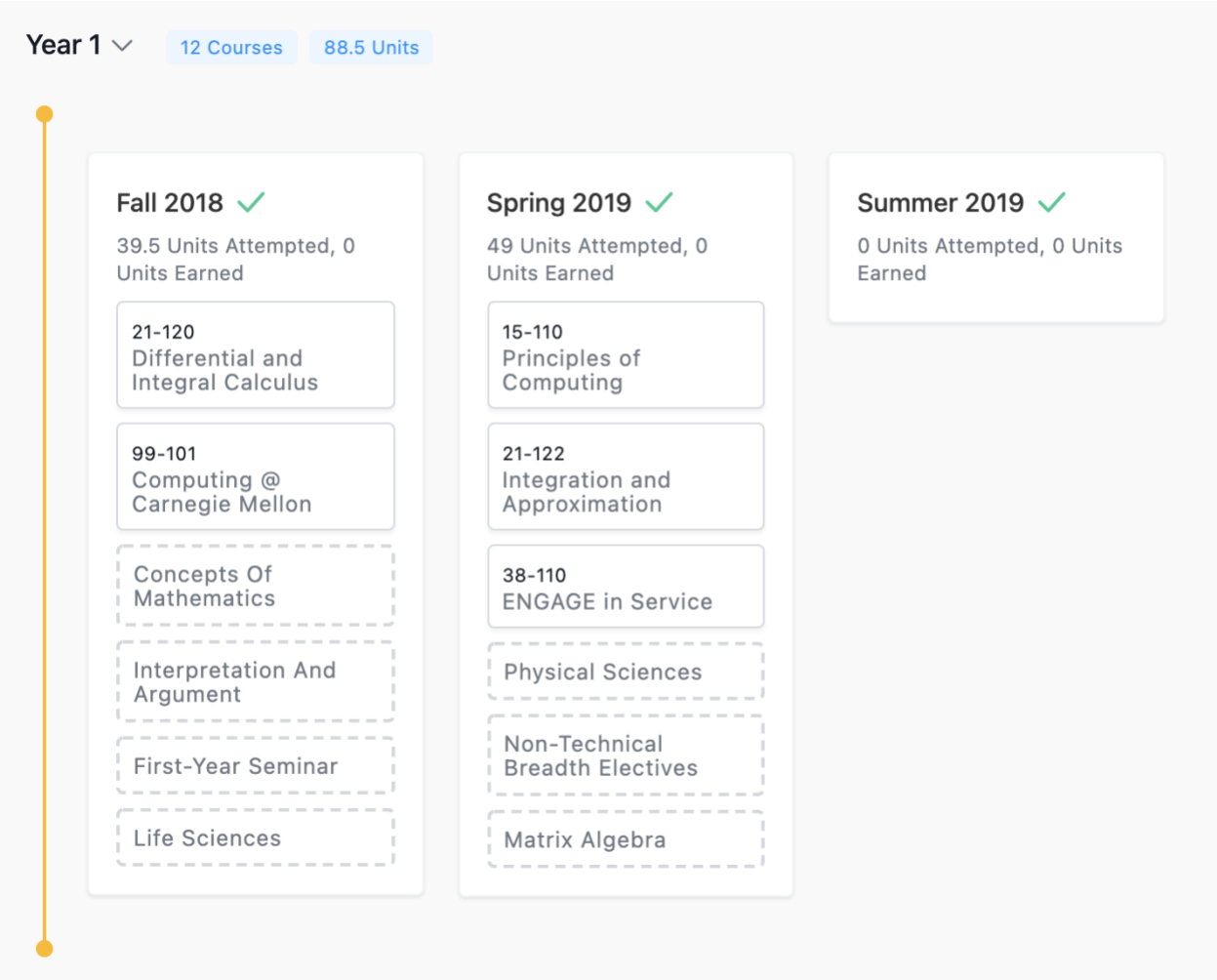
S22_Registrar_Classrooms_1nov21

From the registrar, we have the above excel sheet which lists the building, room number, reg_schedule, and the max capacity of the room. Of these columns, we use all but the third.

Stellic

Priority Output
Priority Matrix

Stellic is a degree-auditing platform used by CMU. On Stellic, different pathways are listed for each major. A pathway lists which courses should be taken in which semester. Below is an example pathway for Mathematical Sciences (Operations Research & Statistics) for Year 1:



We used Stellic to gather individual pathways for the 161 majors at CMU and identify courses that should not overlap as a specific major suggests they be taken at the same time. Majors in Doha campus are excluded.

First, we extracted all the courses for spring semesters in each major. For instance, the example above will result in [[15-110, 21-122, 38-110]], and this list will be expanded as we add more majors and years. This 2D list is then cleaned by removing courses that are no longer offered at CMU. After that, we made a list of pairs of courses with counts for priority in Priority Output. Such priority counts can be used in greedy algorithms to make sure that courses that are taken by many majors don't overlap with each other in order. From here, we created a priority matrix listing all courses in a row and a column. If there is a conflict between two courses in row i and column j, entry ij in the matrix is marked as their count number in Priority Output, else it is marked as a 0.

Assumptions and Constraints

Basic Assumptions & Constraints

There are several basic assumptions and restrictions we must follow when scheduling the courses. First, we must make sure the room chosen is appropriate for the class. By this we mean it meet the following criteria:

1. Only one class in a room at a time
2. Size - We need to ensure that the room is big enough for the course; thus the room capacity as marked in (data source) must be greater than or equal to the max enroll as listed in the registrar data.

For simplicity and lack of available data, we ignore the type of classroom (lab, small classroom, lecture hall, studio, etc.) and any special room/ building requirements needed for a course and choose to focus simply on the above constraints.

Next, we have several restrictions as to the timing of classes:

1. No professor can teach two courses at the same time
2. Each class has duration and days of week as listed from Registrar data
3. Each class must be scheduled within the 13 hour period we set up the schedule to include (approximately 8:00am - 9:00pm), which is consistent with the length of the day at CMU in practice.

There are also a few exceptions:

1. Courses in Mini 3/ Mini 4 may overlap in terms of professor and classroom as they are taken at different points of the semester.
2. Courses cross-listed between departments and undergraduate/ graduate levels must be at the same time/ place.

Major Constraints

The individual pathways for each major gathered from Stellic are used to construct the constraints for class pathways that cannot overlap. The data is taken from the matrix where if there is a conflict between two courses in row i and column j, entry ij in the matrix is marked as their count number in Priority Output, else it is marked as a 0.

Multiple Sections/Recitations

Many of the courses at CMU have several sections of lecture, denoted CRSE#-1, CRSE#-2, ... in the schedule and have multiple recitation sections per lecture, denoted CRSE#-A, CRSE#-B, … When scheduling, the lectures may be at the same time given there are two different professors for the course, else they must be at different times. Recitations are taught by separate TAs, so they may be scheduled at the same time.

Cross Listing

Some courses are cross-listed between an undergraduate and graduate level, or between departments. For example, 10-301 and 10-601 represent the same course, but one is listed at the graduate level and one at the undergraduate level. Similarly, 03206 and 42203 represent a cross-listing between departments. Since these courses are the same, they must be scheduled at the same time. In our schedule, we list these courses as 10-301/601 and 03206/42203.

Method:
Initially, we wanted to automatically take cross-listed courses by coding, but there were many exceptions and cases. For example, some courses like 10-301 and 10-601 had the same course name (MACHINE LEARNING) while some courses like 54-333 (PRODUCTN PERSNL MGT) and 54-334 (PRODCTN RESRCE MGMT) had slightly different course names while offered by the same professor and at the same time. Hence, we decided to manually look at all the courses and combine cross-registered courses into one. Factors determining cross registered courses were instructor's name, duration, course name, and the days.

Data Manipulation

Once the data were collected, we began manipulating the data in R to better handle the constraints described above. We added indicator columns for each day of the week (1 if class is held that day, 0 else) and if the class is a mini and if so which mini it is in. We also added a column for class duration and combined rows for classes with multiple professors (previously each professor had a unique row despite it being the same class - in the updated format both professors were listed under the same row). Following the data manipulation, we produce several new spreadsheets - one for each day of full semester courses, one for each day of mini

3 courses, and one for each day of mini 4 courses. In the Python code we also combine the building and room number into a singular column for the classroom dataset.
The data for each day for the full and both mini schedules were loaded into python using the pd.read_excel function. We also created a python class, “Lecture”. This class had fields for course number, section, max enrollment, professor, and duration. The function listLectures reads through the dataframe created from the excel files and fills the fields with the correct information from the excel file. The output is a list containing Lecture class data structures that contain all the necessary information about all the lectures. Using a class data structure makes it easier to access the relevant features for each course while scheduling and provides a more uniform structure to the code.

To better handle constraints surrounding professors’ specific schedules, we create a prof_dict. Each professor is a key in the dictionary and the values are 10 matrices corresponding to each schedule initialized to 0’s then changed to 1’s to represent if a professor is busy during that time.

Scheduling Process

To schedule the courses for S22, we created 10 schedules, one for each day of both the first half (mini 3) and second half (mini 4) of the semester, which can be done via 3. Main Function in the appendix.

Overview

Each schedule is a matrix in which the rows are time indices in [0,n] representing blocks of 5 minutes each and the columns represent available rooms. When a class is scheduled in a room i for times j-k, the corresponding time slots are filled in such that the entries ij - ik in our matrix have the course number, as well as relevant lecture number or recitation letter. After the last time slot a class occupies, we place a 15 minute break (3 time slots) to represent transition time between classes concurrent with CMU policy. This appears in the schedule as “break.”

As only one course can occupy a spot in the matrix, this implicitly handles the time and space constraints described above. Once we have created our blank schedules, we use the code above to schedule the courses.

The main scheduling function, schedule(courses, days, profs, classrooms, times, acc=0), is a recursive function of up to 10 iterations. The function has a nested for loop, looping through all the days and through each course within the day. For each course, it applies the helper function assign(course, day, prof_dict, M_all, pointr, pointc) that attempts to assign the course to an available slot. If it successfully scheduled the course, the assign function returns a label marked true, and false if the assignment failed. If the assignment failed when we called this in the scheduling function, we add the unassigned lecture to a list of unassigned courses. At the end

of our iterations, if the list has length greater than 0, we switch up the order and try to schedule again.

The output is the schedules as described above, with the courses in a matrix where the rows are the 5-min time slots and the columns correspond to the classrooms, which are then written into an excel file. The code produces two separate schedules for the first half and second half of the semester to account for mini courses.

Assignment Function

Before using the assignment function, we order the list of courses by priority matrix to ensure that we schedule those classes in most conflicts first. In the assign function, we take in the course we wish to schedule, the days it needs to be scheduled on, our dictionary of professors, all the schedules, and our pointers. First, we get the professor(s) for that course. Next, we loop through the classrooms, from smallest to largest. This ensures that each course is placed in the smallest possible classroom, leaving large classrooms available for large lectures and avoiding the possibility of a 20 person class being placed in DH2210 (a large lecture hall), for instance. Next, we loop through available time slots, starting at the beginning of the day. We next check our capacity, size, professor availability, and time constraints. If these constraints are satisfied, we schedule the course for this time slot on each day it takes place and update the professors' schedules accordingly in prof_dict. We return our pointers, the updated schedules, the updated professor dictionary, and True if the course is scheduled, and return our pointers with values +1, the schedules, the professor dictionary, and False if the course was unable to be scheduled. If the course was scheduled, we also add in the 15 min/ 3 time-slot break to the schedule at this point.

Constraint Helper Functions

To check our constraints ini the assignment function, we define four helper functions, capacity(course, classroom), schedule_prof(times, day, profs, profs_dict), major(course, times, M_all), and notFilled(time_interval, rooms, roomidx, days, M_all).

The capacity helper compares the max capacity of the classroom to the max enrollment of the given course. If the room is large enough to accommodate the course, we return True, else False.

The schedule_prof helper checks that all professors for a course are not already teaching a course at the same time and returns True if all professors are available to teach the course at that time and False else.

The major helper function checks to make sure that we satisfy our academic/ major constraints from the Stellic pathway data. If there are no conflicts we return True, else we return False.

Lastly, the notFilled helper checks to ensure the room is not filled with either another class or a break for the duration of the class, returning True if the classroom is available and False otherwise.

Results

raw_dataframe - raw results - This folder contains the spreadsheets as output from the code.
visualize - visualizations - This folder contains the schedules visualized in a more user-friendly format. Users can scroll through the schedule and may also hover over the schedule to see which course is which on the schedule.

Using our algorithm we were able to create the 10 schedules - one for each day of each mini 3 and mini 4 - all located in the folder above. Every course for the semester was scheduled, demonstrating the effectiveness of our algorithm. However, it should be noted that due to some of our assumptions - i.e. that all rooms can be used for all classes - the schedule is likely not feasible for CMU. To correct this, a list of courses requiring special equipment/ setup could be obtained, along with appropriate rooms, and we could schedule these classes due to their strict constraints first (in essence increasing their priority) before running our algorithm on the remaining courses.

Additionally, due to the greedy nature of our algorithm, we fill some classrooms and leave others empty. We also notice our schedule appears triangular, with some classrooms filled only at one end of the schedule and others filled the entire day. We beelive this comes from filling smaller classrooms and earlier time slots first. Thus, further work could be done to more evenly distribute the courses. This could also allow more time between classes and utilizing these other spaces could potentially allow for a shortening of the academic day. This also has implications for the ability to implement a true moratorium. Historically, CMU had had a moratorium from 4:30pm-6:30pm to serve as a time without classes for clubs and athletic teams to practice without conflict. This year, the moratorium was not strict and there were classes within this period. Given the room availability seen on the schedule, it should be possible to implement this again for the S22 semester. The room availability also indicates the ability to hold classes in-person rather than online, given that some of the least scheduled classrooms are also the largest - i.e. DH2210.

Conclusion

The scheduling process is a complex problem that CMU must solve each semester. Given our simplifying assumptions and constraints, we were able to create an algorithm that does effectively schedule courses for S22. However, as noted above the schedule is not necessarily feasible and has room for improvement in terms of optimal space allocation. Future work could be done in collection of the data that is needed to remove simplifying assumptions such as type of classroom, building, equipment needed as well as refining the algorithm to consider how busy a classroom is when assigning classes.