Optimal UTSG Timetable Builder

Richard Yang, Riyad Valiyev, Hannah Omeiza, Hussain Elahi

April 3, 2023

1 Introduction

Before each academic term, students at the University of Toronto are responsible for designing their own timetables. However, even when they have the courses they need to enroll in within their grasp, it may turn out to be a struggle to pick the most befitting lecture sections so that the classes are optimally scheduled. For commuters, amateur athletes and students with other occupations, it is particularly a relevant issue since they have more other activities to keep in harmony with their university classes, hence a greater need for a well-organized schedule.

In addition to that, not all aspects of the UofT's current course-enrolment software, ACORN, are ideal. For instance, based on the long-standing 'UofT Time' system, all classes start ten minutes after the hour and end right on the hour. This gives students a 10-minute commute window between the lecture buildings so that they can attend the entirety of their classes even if the lectures are scheduled back-to-back. But is, in fact, the walking distance between every two buildings at UofT under 10 minutes? While the answer to this is clearly "no", does ACORN caution students when they enroll in two consecutive lectures happening in buildings that are more than 10 minutes apart? Or do students have to manually go on to Google Maps and check the walking time between their successive lecture locations by themselves? Unfortunately, the latter is the case at present.

With all these nuisances in mind, observing the necessity for a more facilitated and convenient planning tool, we decided to focus our project on the following research question:

Using a tree-like data structure, how can an AI design the most suitable timetable for UofT students given their availability and the courses they want to enroll in?

As such, our initiative was to develop a software that helps students, upon specifying which courses they are interested in taking in which term, obtain the most time-efficient schedule at the UofT. By doing so, not only did we make it possible to generate timetables that have at most 10 minutes of traveling distance between the locations of back-to-back lectures, but we also made sure that our product provides the most flexible time-schemes to its users within the frame of their availability. In particular, we programmed our system to produce timetables that keep the user's day in the university campus as short as possible while ensuring that they have enough breaks between the classes, so as not to get exhausted along the way.

By means of this project, we aimed to assist the UofT students in their time-management skills and have them avoid undesired frustration, arising especially amongst first-year students unfamiliar with the campus setting.

2 Datasets

2.1 List and timings of courses

A json file containing course codes and lecture information for all CSC, STA and MAT courses at the St.George campus of UofT. The lecture information includes lecture sessions, their timings, locations, and

other miscellaneous data about the session. Only the data relevant to this project's purposes were included in the format below:

- ⋆ Python Dictionary/Mapping in the following format:
  {Courses: . . . , {Lectures/Sections: . . . , {Sessions:... {Meeting Day:..., Meeting Start Time:..., Meeting End Time:..., Meeting ID:...,Meeting Location:...} } } }
- ⋆ Source: UofT 2022-2023 Fall Winter Session Timetable, Faculty of Arts and Sciences Course Listings (https://timetable.iit.artsci.utoronto.ca/)

2.2 List of buildings

A csv file that contains a list of names of buildings on the St.George campus of UofT where courses are held. It also matches each building name to its building code, which is used to identify it.

- ⋆ Format: Building Name — Building Code
- ⋆ Source: List of UofT buildings from Wikipedia(https://en.wikipedia.org/wiki/List_of_University_of_Toronto_buildings)

3 Computational Overview

Using courses and their lecture sessions, we will create a Schedule tree containing all possible schedules, with different combinations of lecture sessions. The tree starts with a 'Default Lecture' node and afterwards, at each depth level, there will be a single course and the nodes at this level will refer to each of the course's lecture sessions. So, when another course is added, its lecture sessions will be added as subtrees of each lecture session of the previous course. In this way, branching paths shall be created, containing all the different possible schedules, using different combinations of lecture sessions. In this recursive tree design, the root of each tree will be a lecture session, and its subtrees will be a dictionary mapping lecture codes to another instance of the Schedule class. So, as more courses are added, they will be recursively added into this tree, thus increasing the overall depth of the tree. However, not all of the possible paths through this tree will be valid, as there may be some conflicting lectures. To check if two lectures are conflicting, we check if the first lecture starts before the other ends, and ends after the other starts. Additionally, the spread of classes might be sub-optimal. Therefore, we shall find the best timetable possible from the various paths through the tree, based on a points system that deducts points from possible paths through the tree using some criteria, such as the timings of the lectures, the days that they are held on, and whether they are back to back. For back to back lectures, we calculate the walking time between the two buildings the lectures are held in, using the Google Maps API. Because we must create a tree of all possible schedules, and must then calculate the best possible path through it, trees play a central role in our representation of the data.

The first step to creating the optimal timetable is extracting and cleaning data from our datasets. So, we read the csv file, 'building_names_and_addresses.csv', which matches building codes to their actual locations. The data from this csv file is also stored as a dictionary in the 'Catalogue' class. Additionally, in 'catalogue.py', we extract the important information, such as lecture sessions, their timings and the building codes in which the lectures are held, from the json file, 'all_data.json'. This data is stored in a 'Course' class, and a dictionary is created, mapping each course name to its respective Course. This dictionary is an attribute of the 'Catalogue' class. We also make sure to add NA values for any building codes that are not recognised.

After the user inputs whichever courses they wish to take, we shall use this data to help create a schedule tree. For each course that they input, we shall get all the lecture sessions from the Catalogue class, and recursively add each one into the tree. When adding courses, we will check if the lecture session we plan on adding conflicts with any of the pre-existing sessions, and if so, it will not be added to the tree.

Next, we ask the user for days that they wish to avoid having classes in, and the time period that they want to have all of their classes fall into. We will use this information in order to create the optimal timetable for that student. But first, we have to get all the valid paths through this tree, which refers to all paths in which all inputted courses were successfully added, meaning that there are no conflicts. Using these valid paths, we analyse each of them, through a points system. Each potential schedule starts with 100 points, and points are either added or deducted based on the following:

- ⋆ If two courses are back to back, and the walking time between them is greater than 10 minutes(the time gap between the two classes), then the travel time is deducted from the score. This walking time is calculated using the Google Maps API and the 'googlemaps' library. If the travel time is less than 10 minutes, it is awarded 20 points.
- ⋆ If any course starts before the user's selected earliest start time, then 7 points are deducted per hour that it starts earlier than the user selected
- ⋆ If any course starts after the user's selected earliest end time, then 7 points are deducted per hour that it ends later than the user selected
- ⋆ If any lecture is held during one of the exclusion days, 20 points are deducted

The schedule with the highest final score is the one that is selected and then outputted. The outputted timetable (the interactive display) will use the information from our optimal algorithm. Its look will have columns which represent one hour periods, and rows which represent days. We decided to have as many columns up to the latest lecture time in our result as an attempt to make our table more dynamic. We used the plotly package for this.

A plotly table has a header and a cells attribute. The header for us was a list of one hour time periods from 9:00 am up to the latest class hour. Each column would have a header of a one hour time period (i.e. 9:00 - 10:00). The cells attribute used timetable information converted from our data classes to a list of lists. We needed the information to be in the right position in the list of lists. Hence, we took advantage of list indexing.

Each list in the list of lists has a maximum of five items for 5 days of the week. The number of lists in the cells attribute corresponds with the number of elements in the header attribute. This means in pseudo code: table.cells [1] will be the information underneath table.header[1]. Once we got the header information using our custom functions; we created a default table with the cells attribute having only the days of the week as the first list in the list of lists while the other lists had 5 empty strings. This translates to an empty time table with just time information and days of the week at the far left; where each day of the week corresponded to a row of the table.

We changed the empty strings in the cells attribute only if we were in the right position to change them. We are at the right position when we are in the right time column and the right day column. We use an index based loop through the default table and loop through the time table information which contains a set of dataclasses we called 'sessions'. We check if the session's start time and end time corresponds to the default table at the index we are. If it does, we are at the right time column. That is table.cells[index] . We then loop through the items in this list.

As each item corresponds to each day of the week, we converted the days of the weeks to numbers in a dictionary; 0 for Monday up to 4 for Friday. If the session's day corresponded to the index we were at in the for loop; we could replace the empty string with the session's information. That is the course code and the location. The header and cells information was made using our functions and they were called in the plotly code that converts this information into a visual table.

4 Instructions for Operation

To build your own optimal timetable, simply run the main function. In the console, you should get a prompt which asks you for an input. There are several steps in this process. The instructions are outputted in the console, but if you ever get confused refer back to here.

First, you will need to input the semester for which you'll be constructing this timetable for. (Either F or S, F for fall and S for winter). Second, the program will ask for the courses that you want to take. Keep in mind that we did not put all of UofT's courses into the program. We only included the relevant ones. Try any statistics, computer science, or mathematics course. Input as many courses as you want to take for that semester. Keep in mind that some courses are only offered in certain semesters. (Csc111 is only offered in the winter semester). If you input a course that is not offered that semester or you input courses that completely conflict with one another, it will ask you to redo your inputs.

Thirdly, the program will ask you for exclusion days. Exclusion days are the days when you would prefer not to take classes. Note that although the program will take this into consideration, it is not guaranteed as some courses are only offered on certain days. Fourthly, the program will ask you for the hours at which you would want to start and end classes. Input a number from 9-22, signaling at what hour you would like to start / end classes. The program will take this into consideration, but it is not guaranteed. Finally, it will output a timetable using plotly! Enjoy your optimal schedule.

Example outputs:

	9 :00 - 10 :00	10 :00 - 11 :00	11 :00 - 12 :00	12 :00 - 13 :00	13 :00 - 14 :00	14 :00 - 15 :00	15 :00 - 16 :00	16 :00 - 17 :00
Monday	STA130 LEC-0101 (Leslie L. Dan Pharmacy Building)	STA130 LEC-0101 (Leslie L. Dan Pharmacy Building)						
Tuesday						MAT137 LEC-0601 (Sandford Fleming Building)	CSC111 LEC-0201 (Lassonde Mining Building)	CSC111 LEC-0201 (Lassonde Mining Building)
Wednesday						MAT137 LEC-0601 (Sandford Fleming Building)	MAT223 LEC-0401 (McLennan Physical Laboratories)	MAT223 LEC-0401 (McLennan Physical Laboratories)
Thursday						MAT137 LEC-0601 (Earth Sciences Centre)	CSC111 LEC-0201 (Lassonde Mining Building)	
Friday							MAT223 LEC-0401 (McLennan Physical Laboratories)	

Term: S
Courses: MAT137, CSC111, MAT223, STA130
Exclusion Days: WE (Wednesday), FR (Friday)
Start/End times: 9 - 16 (Morning - Afternoon)

	9 :00 - 10 :00	10 :00 - 11 :00	11 :00 - 12 :00	12 :00 - 13 :00	13 :00 - 14 :00	14 :00 - 15 :00	15 :00 - 16 :00	16 :00 - 17 :00
Monday			CSC110 LEC-0201 (Myhal Centre for Engineering Innovation & Entrepreneurship)	CSC110 LEC-0201 (Myhal Centre for Engineering Innovation & Entrepreneurship)	MAT137 LEC-0501 (McLennan Physical Laboratories)			
Tuesday					MAT223 LEC-0301 (Sidney Smith Hall)	MAT223 LEC-0301 (Sidney Smith Hall)	CSC110 LEC-0201 (Lassonde Mining Building)	CSC110 LEC-0201 (Lassonde Mining Building)
Wednesday					MAT137 LEC-0501 (Lassonde Mining Building)			
Thursday					MAT137 LEC-0501 (McLennan Physical Laboratories)	MAT223 LEC-0301 (Sidney Smith Hall)	CSC110 LEC-0201 (Lassonde Mining Building)	CSC110 LEC-0201 (Lassonde Mining Building)
Friday								

Term: F
Courses: CSC110, MAT137, MAT223
Exclusion Days: None
Start/End Times: 12 - 22 (Afternoon)

5 Changes to project plan

We haven't made any significant changes to the overall idea and concept, goals and plans that were delineated in the project proposal. The only changes made are technical improvements on the computational models and datasets. One suggestion made by the TA in the feedback to our proposal was that an option for the student to have all their lectures in just a few days should be added. So, we implemented this through a feature that allows students to pick certain days that they wish to avoid having classes on. When finding the best possible timetable, we check if it has any courses on these excluded days, and if so, it is disincentivised, using the points system. We have augmented the dataset of our courses to an extent that it now contains all CSC, MAT, and STA courses rather than only CSC courses which is what we were planning in the proposal. Upon that, we inspected that the running-time of our overall program does not suffer from this change at all. Moreover, we realized that the completion requirements of CS-related programs at UTSG include the courses that are mostly offered by these three departments*, which implies that our decision to broaden the range of departments was spot-on in addressing the user audience of CS students.

6 Discussion

As a result of many trials and errors experienced during the computational process, we have successfully managed to finalize our project and have it work in accord with the idea we originally came up with. The program performs the operations we had intended it to perform, produces the desired output, and presents it through aesthetically pleasing visualizations. That being said, we also did encounter a few limitations in the computational and dataset-related dimensions of our project, which are thoroughly outlined below.

6.1 Datasets

At one point during our work, we considered including all courses at the University of Toronto into our json dataset rather than only CSC, STA, and MAT ones. This way, our product would have been accessible to all students at the UofT regardless of their program of study. However, we consulted Prof. David Liu about this idea during his office hours and his advice was that we should keep it limited to a few departments, as including all courses would be beyond the scope of our course and estimated scale of this final project.

On top of that, the process of us extracting the course data from UofT's website was fully manual. Since there was not a dataset of courses online, nor an option to download the courses listed in the UofT's timetable into a file, we had to access the Google developer tools for each department's course listings in the aforementioned website, and transfer the relevant data into a file by ourselves. Considering the number of faculties and departments at the UofT, doing this process for all academic units would have been extremely time-consuming. For these reasons, we decided to stick to the departments that mainly concern CS students only.

6.2 Computational Strategies / Algorithms

The idea behind this optimal schedule planner is to abuse the concept of trees to our advantage. By creating a large tree containing every possible schedule, we can recurse down each path in the tree, adding and deducting scores on the way based on preferences. The path with the highest score will be, in our definition, the optimal timetable.

We removed points for the paths that contained classes outside of the student's preferred hours and days. However, we understand that students do not like having large gaps between classes, as you just sit around and wait for the next class. So, we rewarded the algorithm for having paths that contain back to back classes by giving those paths additional points per back to back class. Additionally, we understand that classes who

have more than a 10 minute travel time are a terrible hassle to make it to. If the classes are back to back but have a lengthy travel time, instead of rewarding them, we deduct points for the time it takes to travel, to disincentivize those schedules.

In order to construct the tree, at every level of the tree, we add all possible lectures for a certain course. For example, if the courses are MAT137, CSC111, and MAT223, the first level will be all MAT137 lectures. The second level of the tree will be all possible, non conflicting CSC111 courses that branch off from the MAT137 lectures. Any conflicting lectures are removed automatically, as those are undesirable. In the end, we have all possible schedules. After applying the algorithm above, we can come up with the best timetable.

6.3 Display

Coming up with the best timetable was the point of our project but we also wanted an interactive, beautiful output. At first we thought of using the pygame package to help with this. We used plotly instead as it made it less difficult to format our table to be understood by users. Plotly didn't give us the issue of finding a certain x and y coordinate to place our timetable information. As the resulting information in the timetable could be different depending on the information given to the program by the user, we wanted to avoid the pitfall of having our display possibly becoming incoherent, making the user not see the timetable information properly.

A plotly table needs a list as a header of the table. In our case, the headers were each one hour long. It needs a list of lists as the cells beneath the header. In our case this was the days of the week, and the important timetable information. (The lecture section and location).

We wanted to avoid long timetable information from being cut off in the cells of the table, so we enabled text wrapping by setting the column width attribute of the plotly table beforehand.

Using this form of plotly table allows horizontal and vertical scrolling in case of a large table that may stretch across the screen. One can also save this table in png format by clicking the camera at the top right corner of the display pop up screen.

Underneath each time heading is five cells for five days of the week. More than one hour lectures are not represented in one cell but in two, breaking the time into subsequent one hour periods. For example, If I have CSC110 for 2 hours on Monday from 9:00 - 11:00, I look at the 9:00 - 10:00 column on the first row (because this is the Monday row) and the 10:00 - 11:00 column, also on the first row.

6.4 Further Exploration

On the way to developing our project even further, we can consider expanding its scale so that it works with the courses from all departments of the Faculty of Arts and Science. Doing so would be the first stage of upgrading our software. Later on, we can also increase the variety of factors that are taken into account while producing the most suitable timetable. As the question of "what makes a timetable optimal?" is quite open-ended, there can be many suggestions of elements that add to the optimality of a schedule. One of them, for instance, may be the instructors' scores according to the student evaluations of teaching in courses. That is, during the selection, the timetable can prioritize the section taught by the professor with the highest evaluation score in a given course. Another possibility could be adding further criteria decided upon by the user, such as whether they wish for their courses to be spread out, allowing for each timetable to be tailored to the user. In the last phase, we can extend the usage of this platform across all the academic units operating in all three campuses of UofT.

7 References

Plotly. "Tables in Python." Plotly, Plotly, Accessed 29 Mar. 2023, https://plotly.com/python/table.

"2022 - 2023 Fall Winter Session Timetable." U Of T Faculty of Arts and Science 2022 / 2023 Timetable, https://timetable.iit.artsci.utoronto.ca/.

Googlemaps. "Google-Maps-Services-Python/Googlemaps at Master · Googlemaps/Google-Maps-Services-Python." GitHub, https://github.com/googlemaps/google-maps-services-python/tree/master/googlemaps.

"Googlemaps." PyPI, https://pypi.org/project/googlemaps/.

"List of University of Toronto Buildings." Wikipedia, Wikimedia Foundation, 27 Feb. 2023, https://en.wikipedia.org/wiki/List_of_University_of_Toronto_buildings.

"University of Toronto." Map.utoronto.ca, https://map.utoronto.ca/.

"Computer Science — Academic Calendar." Academic Calendar, https://artsci.calendar.utoronto.ca/section/Computer-Science. Accessed 2 April 2023.