NuhSchedule: A Web Application for Automated Course Scheduling

by

Joseph Nuhfer

Honors Thesis

Appalachian State University

Submitted to the Department of Computer Science

and The Honors College

in partial fulfillment of the requirements for the degree of

Bachelor of Science

May 2018

APPROVED BY:

Raghuveer Mohan, Ph.D., Thesis Director

Gregory Rhoads, Ph.D., Second Reader

Alice McRae, Ph.D., Departmental Honors Director

Jefford Vahlbusch, Ph.D., Dean, The Honors College

ABSTRACT

NuhSchedule: A Web Application for Automated Course Scheduling.

(May 2018)

Joseph Nuhfer, Appalachian State University

Appalachian State University

Thesis Chairperson: Raghuveer Mohan, Ph.D.

College students take on a large amount of coursework, and require much time and physical energy to be able to successfully complete their tasks. Students would like to develop course schedules that save them time spent on campus, and physical energy walking between halls on campus. These are somehow overlooked among the different schedule possibilities. In this thesis, we present a web application, named NuhSchedule, that is designed to automate the development of course schedules for students at Appalachian State University. Based on user preferences, like course sections, and break times, the application generates and presents a set of schedules that tries to optimize based on schedule conciseness (opting for minimum time on campus) and minimum walking time between halls on campus. These objectives allow students to save valuable time that could otherwise be spent on other activities, and physical energy by walking less between classrooms. A small-sampled beta-testing was performed with the help of actual students. Our schedules were found to be comparable to the manual ones created by them.

Contents

List of Figures

List of Listings

Chapter 1

Introduction

Developing a course schedule is a time-consuming effort for college students. Students must browse through many different course choices, and determine which of the many possible schedules is best for them. This process not only entails choosing courses that do not contain time conflicts, but also determining convenient walking routes and break times. In this thesis, we develop a web application named NuhSchedule that is designed to automate this process, and make course scheduling a more convenient and time-saving endeavor.

In NuhSchedule, a user specifies possible course sections, a desired time cushion for travelling between classes, minimum time for breaks, maximum amount of time spent without a break, and times in which classes may not take place. They are then presented with a set of schedules that tries to conform to their preferences and is nearly optimized based on schedule conciseness (opting for shorter lengths for school days) and minimal walking time. Schedules generated using NuhSchedule save users valuable time and energy. They are concise, allowing for less time spent on campus and more time completing other tasks. They minimize walking, to avoid wasted physical energy during school days. With more time and physical energy to spare, NuhSchedule users are better equipped to handle their studies.

NuhSchedule has been beta-tested for efficiency with a small sample of students. We found that schedules generated by NuhSchedule matched, and in some cases beat, the schedules manually prepared by the students in suitability.

The rest of this document provides a detailed description of the NuhSchedule program. Chapter 2 details a brief literature review of prior efforts made by others to optimize the

automation of course schedules, and what separates NuhSchedule from those efforts. Chapter 3 provides a detailed description of the web application itself, and its many components. Chapter 4 gives a full explanation of the user-testing process and conclusions drawn from it, and describes future work to be done to improve NuhSchedule.

Chapter 2

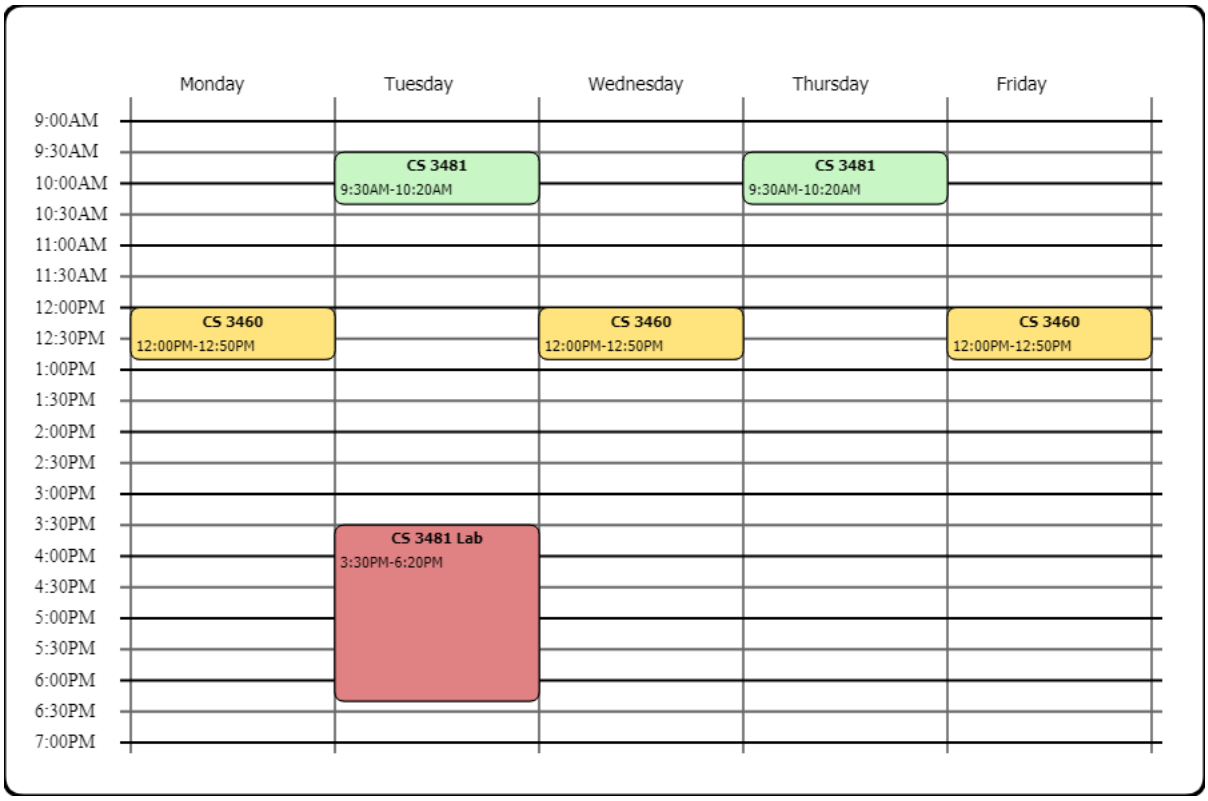
Literature Review

In this chapter, we survey the relevant literature and software that have been developed to assist students in building course schedules. Section 2.1 describes a course schedule visualization tool called Free College Schedule Maker Online. Section 2.2 details a web application named Coursicle that functions as a schedule visualization tool as well, with the additional functionality of allowing students to choose courses from their university. Section 2.3 describes Schedule Maker for Rochester Institute of Technology (RIT), which allows RIT students to choose courses from their university and generate all possible schedules. Section 2.4 compares NuhSchedule to the previous literature discussed.

2.1 Course Schedule Visualization

Free College Schedule Maker Online [9] is a web application developed by Jacob Heruty [8] which allows users to input courses by course name, meeting days and times, and other optional parameters and receive a visualization of their course schedule that can be saved as an image file or printed. Users may choose as many different courses as they like, and may customize the color scheme of the visualization. Users may also export their schedules as a file that may be imported later for future edits.

Figure 2.1: Schedule visualization generated by Free College Schedule Maker Online.



Many other applications can be found with similar functionality, including Schedule Builder Online [5], Class Schedule Maker by GradeTracker [2], and Class Schedules by Canva [4] (which also allows the upload of images to customize visualization further).

While this application allows for easy visualization of college schedules, users are required to manually provide information on each of their courses, which is a time-consuming process. In addition, changes to course schedules require manual editing of the visualization. While NuhSchedule does not provide colored visualizations, it does display schedules in an easy-to-read format, and pulls course information directly from the Appalachian State course database.

2.2 Course Schedule Visualization with Course Selection

Coursicle [3] is a web application that allows users to search through selected universities' course listings, and build a visualization of their course schedule. Coursicle supports hundreds of universities in the United States and Canada. Users may select as many courses as they like; a visualization is then automatically built that may be printed or saved as an image file.

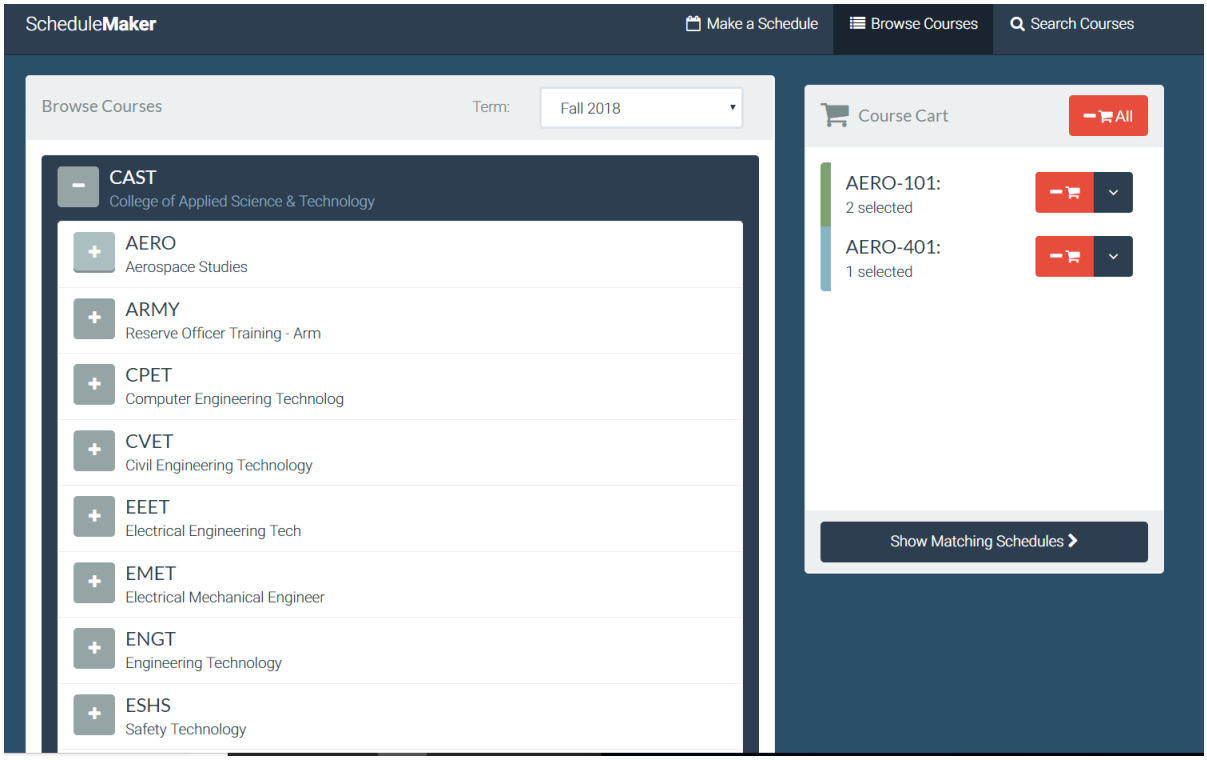
Figure 2.2: Results from searching for Computer Science courses at Appalachian State using Coursicle.

While Coursicle allows for users to search through university course listings (including Appalachian State) and select courses they'd like to take, users are still required to build their schedules themselves. NuhSchedule automatically presents a set of schedules that tries to optimize two important objectives for most students – schedule conciseness and walking time.

2.3 Course Schedule Generation and Visualization

Schedule Maker for Rochester Institute of Technology (RIT) [6] is a web application that lets users browse RIT courses and select course sections they are interested in. Users may also create "Non-Course Schedule Items" that function in the same way as selected courses to represent other weekly activities, and may specify times in which they do not want regular courses. Users may then click a "Show Matching Schedules" button, which generates all possible schedules that fit the user's input, each displayed in a separate multi-colored visualization. Users have the option to print visualizations of their choice, as well as save them as an image file, or even embed them in a webpage in which they remain active for three months.

Figure 2.3: The "Browse Courses" section of Schedule Maker for RIT.

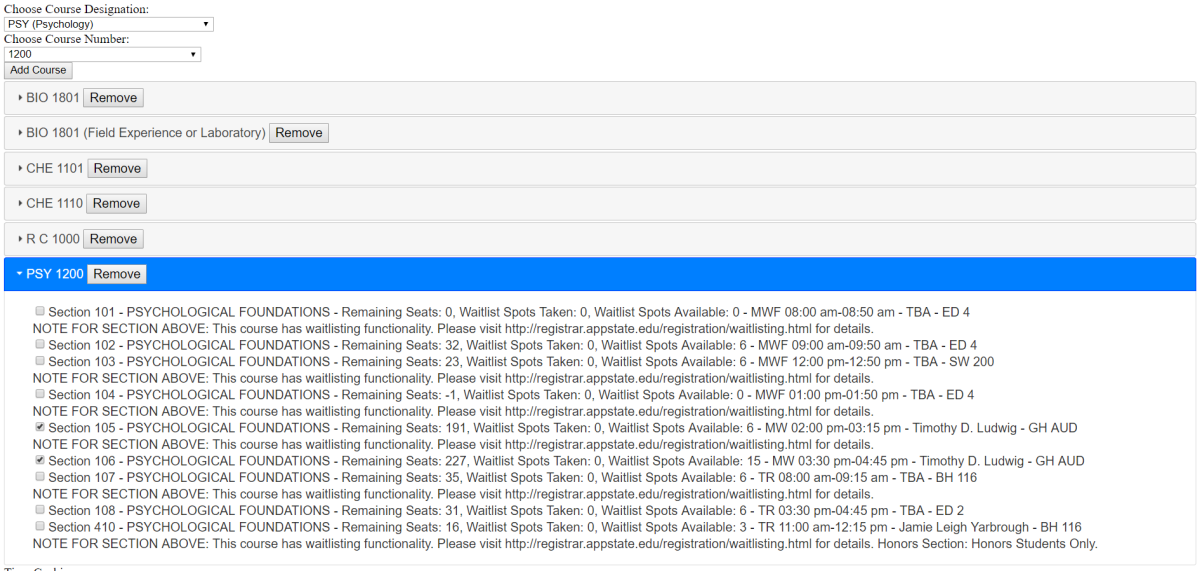


While Schedule Maker allows users to choose individual sections for their courses, and generate all possible schedules that have no time conflicts, it keeps no information on walking time, and therefore could potentially generate schedules that are not realistic for users. In addition, some sets of courses could create hundreds of schedule possibilities, which could be time-consuming to read through to choose the most desired one. NuhSchedule generates a smaller set of "best" schedules, and provides information on schedule conciseness and walking time to aid the user in their choice.

2.4 Comparison to NuhSchedule

While NuhSchedule does not create a multi-colored visualization like the applications described above, our web application provides its output as a simple numbered list of schedules instead. Our application automatically pulls courses from the Appalachian State course listing, and allows users to specify times in which they do not want courses. Functionality exclusive to NuhSchedule includes the ability to specify the minimum amount of time that constitutes a break, and the maximum amount of time that a user may go without a break. Another functionality exclusive to our application is the optimization of schedules generated. While other applications can generate all possible schedules from user selections, NuhSchedule generates all possible schedules, and then tries to provide a set of schedules that minimize walking time between classes, and maximize schedule conciseness, i.e., minimize the total time spent on campus.

Figure 2.4: NuhSchedule's course selection from Appalachian State's course database, and schedules generated from those selections.



Generate Possible Schedules

Schedule 1
Time on Campus: 21hr 25min, Est. Walking Time: 25min
Monday:
CHE 1101-104: 1100-1150 GWH 112
BIO 1801-103: 100-150 RSW 183
PSY 1200-105: 200-315 GH AUD
R C 1000-118: 330-445 SH 301
CHE 1110-129/130: 600-850 GWH 314/307
Wednesday:
BIO 1801-205/217: 800-1050 RSW 180/184
CHE 1101-104: 1100-1150 GWH 112
BIO 1801-103: 100-150 RSW 183
PSY 1200-105: 200-315 GH AUD
R C 1000-118: 330-445 SH 301
Friday:
CHE 1101-104: 1100-1150 GWH 112
BIO 1801-103: 100-150 RSW 183

Schedule 2
Time on Campus: 21hr 25min, Est. Walking Time: 25min
Monday:
CHE 1101-104: 1100-1150 GWH 112
BIO 1801-103: 100-150 RSW 183
PSY 1200-105: 200-315 GH AUD
R C 1000-118: 330-445 SH 301
Wednesday:
BIO 1801-205/217: 800-1050 RSW 180/184
CHE 1101-104: 1100-1150 GWH 112
BIO 1801-103: 100-150 RSW 183
PSY 1200-105: 200-315 GH AUD
R C 1000-118: 330-445 SH 301
CHE 1110-111/127/131: 600-850 GWH 307/314/337
Friday:
CHE 1101-104: 1100-1150 GWH 112
BIO 1801-103: 100-150 RSW 183

Chapter 3

Implementation

In this chapter, we detail our implementation of NuhSchedule. This application functions with three components. Section 3.1 details our web scraping component, written in Python, which extracts data from the Appalachian State course database and formats it for our use. Section 3.2 describes NuhSchedule's front-end, written in HTML5 and JavaScript. It lets users choose courses they are interested in, sets parameters for their schedules, and submits these as input to our schedule generation program. It also displays the output of the program to the user. Section 3.3 details our schedule generation program, written in C++ and then compiled as JavaScript. This program takes course choices and schedule preferences as input, and generates a set of "best" schedules as output.

3.1 Web Scraper Component

We developed a web scraper, written in Python 2.7, that downloads the Appalachian State course catalog and formats it as HTML5 code for front-end use. It is written in a file named `courseScraper.py`, and makes use of packages named *Beautiful Soup* and *Mechanize*.

Installed Packages

Beautiful Soup is a Python package, developed by Leonard Richardson, which simplifies the web scraping process. It processes HTML documents, and transforms them into a tree of Python

objects that makes for simpler navigation and data extraction [11]. Our web scraper makes heavy use of this package, in order to extract data from the Appalachian State course database.

Mechanize, a Python package originally developed for Perl by Andy Lester, automates HTML form handling [7]. We use it to make requests to the course database for the information we need, like the current semester's course selection.

Both libraries are used in conjunction. Mechanize loads submitted forms into the results we need, and Beautiful Soup allows us to extract data from those results.

Implementation Details

Our web scraper creates a virtual browser using Mechanize, and opens the Appalachian State course database web page. We choose the academic term of interest (as of this writing, Fall 2018) and then submit. The page that loads contains a form with all course designations (for example, MAT for Mathematics, or C S for Computer Science) available for the term chosen. We process this page with Beautiful Soup and extract this information, formatting it as HTML form options and writing it to a file named `courses.txt`.

Figure 3.1: A section of the table. Notice the repetitive pattern in the data.

10443	ACC	2100	104	MC	3.000	PRIN OF ACCOUNTING I		MW	02:00 pm-03:15 pm	38	38	0	0	0	0	Penelope Lee Bagley (P)	08/21-12/15	PH 3015
							NOTE FOR SECTION ABOVE: Refer to section 101 for common exam dates.									
10444	ACC	2100	105	MC	3.000	PRIN OF ACCOUNTING I		MWF	11:00 am-11:50 am	27	27	0	0	0	0	Rachel D. Keller (P)	08/21-12/15	PH 3016
							NOTE FOR SECTION ABOVE: Refer to section 101 for common exam dates.									
10446	ACC	2100	106	MC	3.000	PRIN OF ACCOUNTING I		MWF	12:00 pm-12:50 pm	40	32	8	0	0	0	TBA	08/21-12/15	PH 1020
							NOTE FOR SECTION ABOVE: Refer to section 101 for common exam dates.									
10447	ACC	2100	107	MC	3.000	PRIN OF ACCOUNTING I		MWF	12:00 pm-12:50 pm	35	31	4	0	0	0	TBA	08/21-12/15	PH 3015
							NOTE FOR SECTION ABOVE: Refer to section 101 for common exam dates.									
10449	ACC	2100	108	MC	3.000	PRIN OF ACCOUNTING I		MWF	10:00 am-10:50 am	27	27	0	0	0	0	Rachel D. Keller (P)	08/21-12/15	PH 3016
							NOTE FOR SECTION ABOVE: Refer to section 101 for common exam dates.									
12696	ACC	2100	109	MC	3.000	PRIN OF ACCOUNTING I		MWF	01:00 pm-01:50 pm	35	26	9	0	0	0	TBA	08/21-12/15	PH 3015
							NOTE FOR SECTION ABOVE: Refer to section 101 for common exam dates.									

We then fill out the forms on the current page to request all courses for the term. This loads a document containing information on every single course section for the term. A quick examination of the page's format reveals that the information is contained in a table, which means that the data follows a simple pattern that we can take advantage of.

We process the page using Beautiful Soup. Following the patterns of the table, we extract data on each of the course sections (excluding Distance Education courses, since we are only concerned with on-campus courses in this thesis, but as future work the implementation can easily be extended to include these). We output data on each course into a text file, sorted into folders by course designation (MAT 1110 would be recorded in a text file named `1110.txt` in the `MAT` directory). Data recorded for each section includes section number, course name, course reference number, remaining seats and waitlist spots, meeting days and time, instructor, hall name, and room number. Additional meeting times and special section notes are included as well if a section has them. Field experiences, laboratories, and other course sections of similar nature are separated from regular course sections of the same name, since they require separate registration. Sections with regular meeting times are given a checkbox, so users may select them when using our front-end. Other sections, such as online courses, are given a "*" in place of a checkbox. This is so they cannot be selected; only sections with on-campus meeting times are optimizable for schedule conciseness and walking time. These sections are only displayed so the user is aware they exist as an option.

An additional file is created for each course designation named `allSections.txt`. This file contains the course numbers for every course available, formatted as HTML form options. The program closes after the entire page is parsed.

3.2 Front-End Component

The front-end of NuhSchedule is written in HTML5 and JavaScript. It is composed of simple HTML forms, and interacts directly with the text files generated by the web scraper, as well as the schedule generation program, using the JavaScript library jQuery.

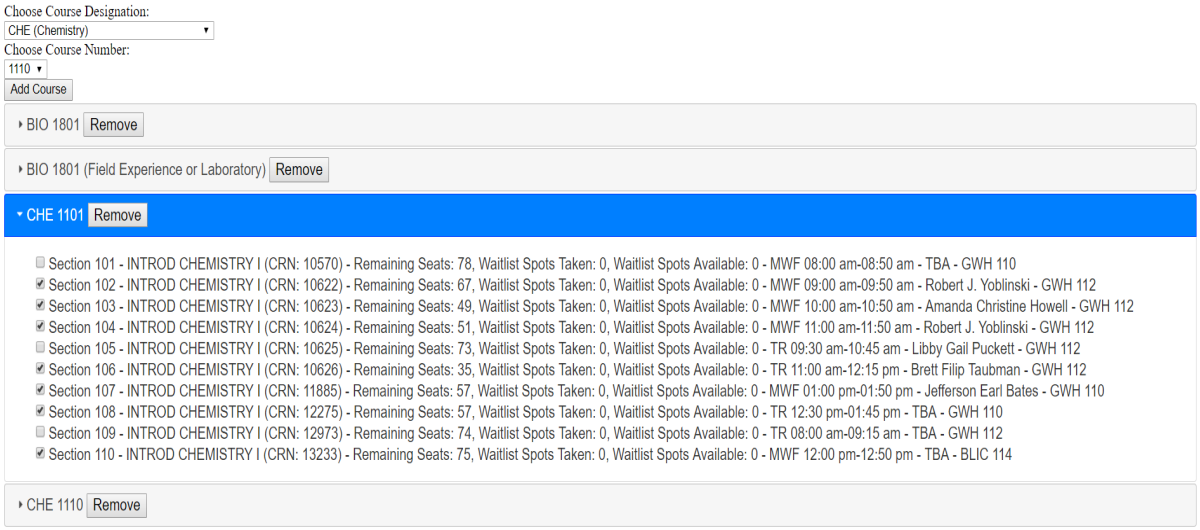
The jQuery Library

jQuery is a JavaScript library that simplifies the development of dynamic web pages. With jQuery, we can add and remove code from our front-end dynamically, and simply handle submitted form data. We can also easily interact with files on our server, such as the ones created by our web scraper [10].

Implementation Details

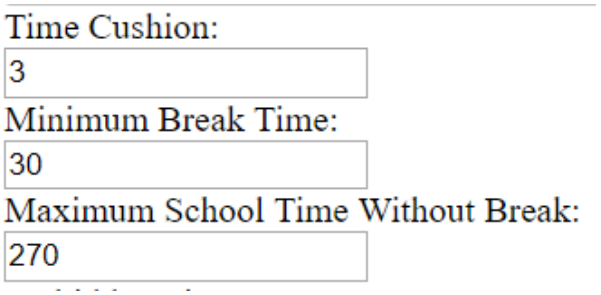
Our front-end contains multiple parts, the first of which allows users to select interesting courses and sections. This consists of two dropdown lists, one to select a course designation, and the other to select a course number. It also consists of a simple button that reads "Add Course", and a collapsible accordion that is initially empty and invisible. Upon page launch, the course designation dropdown list is populated with the contents of `courses.txt`. Once the user selects a course designation, the course number dropdown list is populated with the contents of the corresponding `allSections.txt` file (if MAT is selected, then `MAT/allSections.txt` is loaded). The user can then select a course number, and click the "Add Course" button. Once this happens, the contents of the selected course's corresponding text file are loaded into the accordion (if C S (Computer Science) 1100 is selected, then `C S/1100.txt` is loaded) and the accordion becomes visible. As mentioned in the previous section, this code displays all sections of the course that are available, and checkboxes are placed beside those that meet on-campus. Sections with open seats are automatically checked, while others are not. A "Remove" button is also placed beside the course's name, and when clicked the course's data is removed from the accordion. Users may add as many courses as they like to the accordion, and should check any interesting sections.

Figure 3.2: An example of the first part of the front-end.



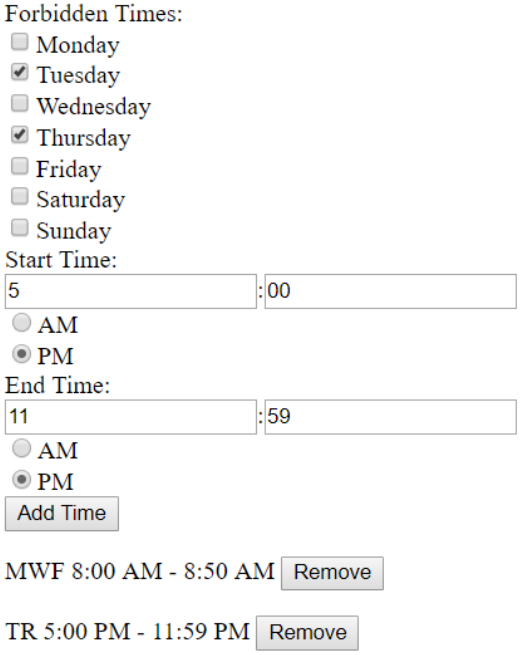
The next part consists of three standard forms where users can input their desired "Time Cushion", "Minimum Break Time", and "Maximum School Time Without Break". All three should be in minutes. Time Cushion represents the amount of extra time users would like to leave to spare when travelling in between classes. For example, if travelling from Hall A to Hall B takes 5 minutes, and the user requests a Time Cushion of 3 minutes, a minimum of 8 minutes will be needed between classes taking place in Hall A and Hall B in any generated schedule. Minimum Break Time and Maximum School Time Without Break are parameters that work in conjunction. Minimum Break Time represents the minimum amount of time that a user considers a break from school activities, and Maximum School Time Without Break represents the maximum amount of time a user would like to allow during a school day without a break.

Figure 3.3: An example of the second part of the front-end.



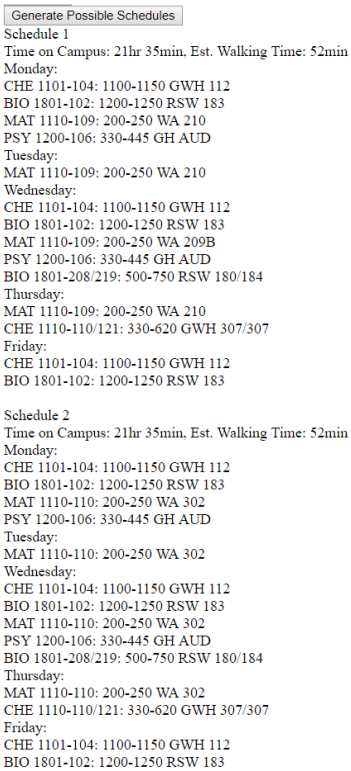
The third part of the front-end consists of seven checkboxes representing the seven days of the week, a pair of forms and a pair of radio buttons representing a start time and an end time, and a simple "Add Time" button. This part is designed to allow the user to mark off desired "Forbidden Times" where no classes can take place in any generated schedule. Once the user fills out the information on their time and clicks on the Add Time button, the time will appear below the button, accompanied by a "Remove" button which the user may click to remove the time. Users may add as many Forbidden Times as they like.

Figure 3.4: An example of the third part of the front-end.



The final part consists of a single "Generate Possible Schedules" button. The user should click on this button only when they have filled out all of the parts described above. Once this happens, text reading "Generating schedules...." will appear under the button. Using JavaScript code, the input file will be built for use with our schedule generation program (the specification of the input syntax is given in Section 3.3). First, for every checkbox in the accordion that is checked, the text describing the section is parsed through and built into input data. This includes special handling of sections with multiple meeting times, and grouping of course sections of the same course type that have identical meeting days, times, and halls. Once all checkboxes have been processed, all possible hall combinations (including those not needed) are added to the input file, along with the walking times between them (we assembled this data manually with assistance from Google Maps, and included it as a large list). Following this, the three fields of the second part are recorded in the input file, and any Forbidden Times are recorded after that. Once this is done, the input file is complete.

Figure 3.5: An example of the schedules generated and displayed.



jQuery then runs the schedule generation program using our input file. Output from the program (the generated set of schedules) is captured and displayed to the screen below the "Generate Possible Schedules" button (brief details on how the output is formatted are given in Section 3.3). Users may browse the set and choose which suits them best. Users may also choose to modify their forms, and click on the Generate Possible Schedules button again; this will cause the previous set of schedules to be replaced.

3.3 Schedule Generation Program

Our schedule generation program is written in C++11, and then compiled into JavaScript using the Emscripten compiler. It takes as input a set of courses (with sections) and other schedule preferences, and outputs a set of schedules that are nearly optimized based on schedule conciseness and minimum walking time.

Input Format

A specific input format is required for our program to run correctly, and it consists of four parts.

The first part of the input consists of the set of courses and their sections. For each course, the course's name is first listed, followed by a line for each section, where each piece of information is space separated. Lines that describe sections are formatted as "SectionNumber DaysOffered StartTime AMOrPM EndTime AMOrPM Hall RoomNumber". An example of the first part is shown below:

Listing 3.1: Example input format for the first part.

```
1 HIS 1200
2 101 MWF 1100 AM 1150 AM BH 243
3 102 MWF 1000 AM 1050 AM BH 243
4 103 TR 1230 PM 0145 PM BH 229
5 104 MWF 0100 PM 0150 PM BH 118
6 105 MWF 0800 AM 0850 AM BH 251
7 106 TR 0930 AM 1045 AM BH 242
```

A special syntax is also available for sections which have multiple meeting times. In this case, the first meeting time would be described as above, and any additional meeting times

would be described with the same syntax on separate lines immediately following the first, but with the section number replaced with "***". An example is shown below:

Listing 3.2: Example of the special syntax for multiple meeting times.

```
101 TR 1230 PM 145 PM WA 303
*** W 0100 PM 0150 PM WA 303
```

There is no limit on the number of courses or sections allowed to be specified. The next part of the input begins with a line that reads "Distances". This contains walking times between pairs of halls. There is no limit on the number of pairs, and each pair is specified on its own line using the syntax "Hall1 Hall2 WalkingTime". An example is shown below:

Listing 3.3: Example of a section of walking times.

```
Distances
AH BH 4
AH WA 13
BH SH 3
BH WA 11
SH WA 13
```

The next part of the input begins with a line that reads "Time". This specifies Forbidden Times as mentioned in the previous section. Times are specified using the same syntax as the first part of the input. An example is shown below:

Listing 3.4: Example of forbidden times.

```
Time
MWF 0800 AM 0850 AM
MWF 1200 PM 1250 PM
MTWRF 0500 PM 1159 PM
```

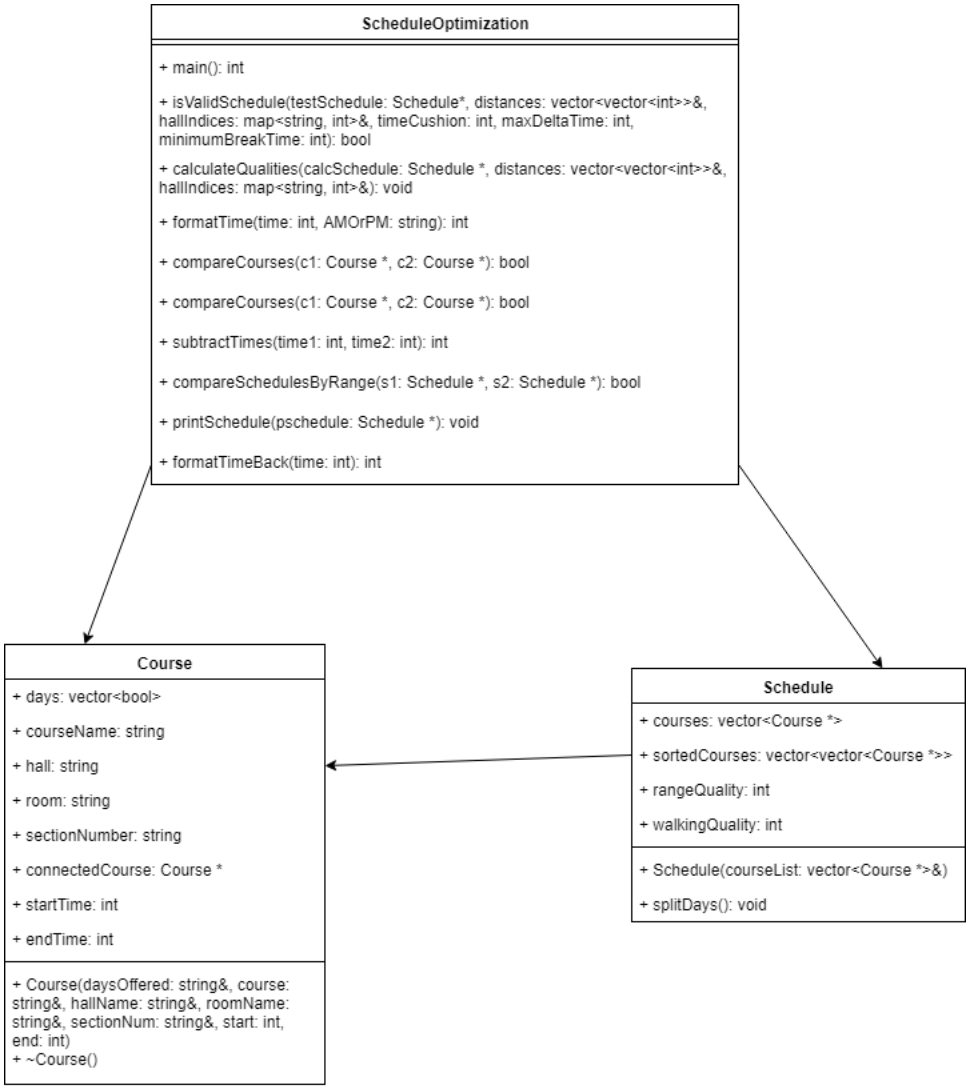
The final part of the input begins when a line appears that reads "Cushion". This is where the parameters Time Cushion, Minimum Break Time, and Maximum School Time Without Break (see Section 3.2) are specified. They are specified as shown below, between lines that read "Cushion", "Break", and "Max" respectively:

Listing 3.5: Example of the final part of input.

```
Cushion
5
Break
30
Max
300
```

Implementation

Figure 3.6: UML diagram of our C++ program.



The program begins by parsing and processing the input file. Information about sections are grouped together by their respective courses. Then, all possible schedules are generated by enumerating all possible combinations of the courses chosen along with their sections. The method `splitDays` is used to analyze each day's schedule individually.

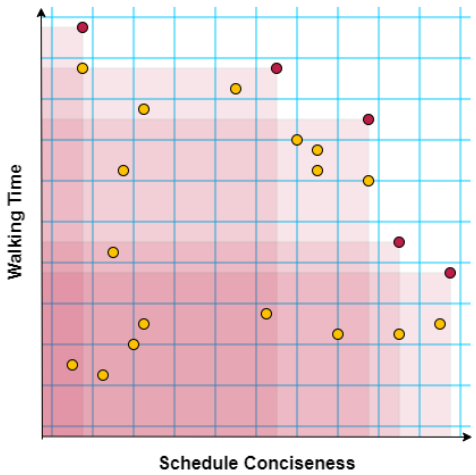
The next step is to discard all schedules that contain conflicts or do not conform to the input. This is done using a function called `isValidSchedule`, which checks for validity by

making two passes through the courses in a schedule. In the first pass, time conflicts are found. Any schedule with overlapping times is invalid (Forbidden Times are included). Also, during this pass, issues with walking time are found. Any schedule that does not allow for sufficient walking time, including the stored Time Cushion, is invalid. During the second pass, breaks are checked for. Any schedule that does not allow sufficient breaks as specified by the stored Minimum Break Time and Maximum School Time Without Break is discarded.

Once a schedule is found to be valid, another method named `calculateQualities` is called that calculates total walking time and schedule conciseness. These are parameters that we wish to optimize. Total walking time is the sum of all walking times between pairs of halls in the schedule throughout the week. Schedule conciseness is the sum of the differences between the start and end times for each day.

Once all invalid schedules are discarded, the program begins to develop a set of "best" schedules. Since there are two objectives, each schedule can be represented as points in two-dimensional space, with schedule conciseness on the x-axis and total walking time on the y-axis. The entire point set is sorted on the x-axis. The optimal schedules are a set of "non-dominated" points. Since we are trying to minimize both objectives, we simply negate the points. Point (x_1, y_1) is "dominated" by point (x_2, y_2) iff $x_2 \geq x_1$ and $y_2 \geq y_1$. An example of non-dominated points is shown below:

Figure 3.7: Representing schedules as a set of points in two-dimensional space. Non-dominated points are indicated in red.



To find our set of non-dominated points, we scan through our sorted set of points from right to left, and keep track of the point with minimum y-value. Each time we find a point that resets the y, we add that to a set of best schedules, because we know that schedule is non-dominated. We also add schedules that exactly tie (in both values) with the previous schedule that was added to the set. We then discard the rest of the schedules.

The `printSchedule` function outputs this set of non-dominated schedules to the user. The schedules are numbered in the order they are printed and are finally displayed as shown in Figure 3.5.

Compilation to JavaScript with Emscripten

Our program is compiled to JavaScript with a source-to-source compiler named Emscripten, developed by Alon Zakai [13]. This allows the program to run entirely server-side with our front-end. We compiled our code with compilation flags that allowed unlimited memory growth, so that larger sets of schedules could still be generated without the program crashing.

Chapter 4

Evaluation and Conclusion

NuhSchedule was beta-tested for efficiency using a sample of ten Appalachian State students who had built their schedules manually for the Fall 2018 semester. We generated a set of "best" schedules for them with NuhSchedule, and found that they either matched their original schedules or beat them in terms of our objectives, which are minimum total walking time and schedule conciseness. Section 4.1 details the evaluation process, Section 4.2 gives a summary of the results, Section 4.3 draws conclusions, and Section 4.4 details future work.

4.1 Process

Each student used NuhSchedule, and they input the desired information, like desired course sections, Time Cushion, Minimum Break Time, and Maximum School Time Without Breaks. We then had them generate a set of schedules, and select one they prefer. Once the student had acquired a preferred schedule from our application, we compared it to their original schedule, based on total walking time and schedule conciseness. After this, we asked each student to fill out a survey, which included questions about whether they preferred their generated schedule to their original schedule and why, and what they liked and did not like about NuhSchedule.

4.2 Results

Seven of the students' manually built schedules matched their generated schedules exactly, two of which adjusted parameters once before receiving their matching schedule. Three of the students' received generated schedules that beat their manually built schedules in total walking time and/or schedule conciseness. All three indicated that they preferred the generated schedule over their original, and expressed similar reasoning that they had "overlooked" this possible schedule configuration.

Schedule 1
Time on Campus: 23hr 50min, Est. Walking Time: 156min
Monday:
CHE 2101-101: 100-150 GWH 112
NUT 3202-101: 230-345 LLHS 232
Tuesday:
HIS 2301-101: 1100-1215 BH 251
NUT 3205-102: 100-215 LLHS 258
BIO 2200-101: 330-445 RSW 158
Wednesday:
CHE 2101-101: 100-150 GWH 112
NUT 3202-101: 230-345 LLHS 232
Thursday:
BIO 2200-203: 900-1050 RSN 203
HIS 2301-101: 1100-1215 BH 251
NUT 3205-102: 100-215 LLHS 258
BIO 2200-101: 330-445 RSW 158
Friday:
CHE 2102-104: 900-1150 GWH 336
CHE 2101-101: 100-150 GWH 112

(a) The student's original schedule.

Schedule 1
Time on Campus: 22hr 40min, Est. Walking Time: 156min
Monday:
CHE 2101-101: 100-150 GWH 112
NUT 3202-101: 230-345 LLHS 232
Tuesday:
HIS 2301-101: 1100-1215 BH 251
NUT 3205-102: 100-215 LLHS 258
BIO 2200-101: 330-445 RSW 158
Wednesday:
CHE 2101-101: 100-150 GWH 112
NUT 3202-101: 230-345 LLHS 232
Thursday:
BIO 2200-203: 900-1050 RSN 203
HIS 2301-101: 1100-1215 BH 251
NUT 3205-102: 100-215 LLHS 258
BIO 2200-101: 330-445 RSW 158
Friday:
CHE 2101-101: 100-150 GWH 112
CHE 2102-103/105/106: 200-440 GWH 454/407/407

(b) The student's generated schedule.

Figure 4.1: This student's schedule was improved by NuhSchedule.

Other interesting information was found as well, mostly involving students' schedule preferences. All students except one chose a Time Cushion of 3 minutes, with the outlier student choosing 5 minutes. Seven students chose a Minimum Break Time of 30 minutes, one chose 15, and the remaining two chose 45. As for Maximum Time Without a Break, preferences varied, but all fell in the interval of 240 and 360 minutes. Two students created Forbidden Times, one to set aside a specific lunch break with friends, and the other to prevent late Friday classes. When choosing additional sections, two students made use of the site Rate My Professors [12] to help aid their decision, which is a public website designed to allow college students to share reviews of professors to their peers.

All except two indicated on their survey that they were completely pleased with the web application. Specifially, one noted that they liked the "speed" of the tool, and another said they liked the "simplicity". The other two both indicated that they did not like the "aesthetic" of the program; in other words, they both believed that the presentation could be improved.

4.3 Conclusions

NuhSchedule seemed to function very well with actual student input. Our objectives of generating nearly optimized schedules by minimizing total walking time and schedule conciseness were met, but with an admittedly small sample size. With nearly no feedback about functionality, we found that NuhSchedule seemed to satisfy student's needs for generating schedules.

4.4 Future Work

NuhSchedule is functional, but is not yet ready for public use. There are multiple improvements that can be made as discussed below.

There are other objectives that we can optimize in the future. Students often have specific bus stops that they are able to arrive to school from. If they live on campus, they live in specific housing. We can incorporate both into our total walking time, to make our optimization more accurate. Also, students often like to plan meals during the day; we can create an option for students to have on-campus meals between their classes. Also, not all students walk to their classes; many ride bikes, and even more ride buses. We would like to include functionality to incorporate bus schedules and routes into the optimization process, and also create options for those who ride bikes.

Our evaluation described in this section only used a small sample of students. A thorough testing on a much larger sample is required to build confidence and make NuhSchedule ready for public use.

Currently, all of the walking times between halls are estimates which we made manually with assistance from Google Maps. We can collect actual walking times from a large sample of students, and calculate more realistic values to replace the estimates.

As currently implemented, our front-end has no input sanitization, meaning there are no checks ensuring the user's input is valid. We can write checks in our JavaScript code, to make sure that no invalid data is entered by the user.

Responding to the feedback given about "aesthetic" in our evaluation, we can take steps to improve the presentation of NuhSchedule, as well as to make our front-end more user-friendly.

Our web scraper currently only runs manually. A script to run it automatically according to a set time interval can be written, so that course data stays updated for users at all times.

Bibliography

[1] Boost C++ Libraries. http://www.boost.org/.

[2] Class Schedule Maker. http://gradetracker.com/lab/class-schedule-maker. Online; accessed 15-April-2018.

[3] Coursicle Appstate. https://www.coursicle.com/appstate/. Online; accessed 15-April-2018.

[4] First Online Class Schedules: Design a Custom Class Schedule in Canva. https://www.canva.com/create/class-schedules/. Online; accessed 15-April-2018.

[5] Schedule Builder Online. https://schedulebuilder.org/. Online; accessed 15-April-2018.

[6] Schedule Maker. https://schedule.csh.rit.edu/generate. Online; accessed 15-April-2018.

[7] Kovid Goyal. mechanize. https://pypi.org/project/mechanize/. Online; accessed 25-April-2018.

[8] Jacob Heruty. Contact Information. http://www.studygizmo.com/contact/. Online; accessed 15-April-2018.

[9] Jacob Heruty. Free Class Schedule Maker Online. https://freecollegeschedulemaker.com/. Online; accessed 15-April-2018.

[10] Dave Methvin. jQuery. https://jquery.com/. Online; accessed 25-April-2018.

[11] Leonard Richardson. Beautiful Soup. https://www.crummy.com/software/BeautifulSoup/. Online; accessed 25-April-2018.

[12] John Swapceinski. Rate My Professors. http://www.ratemyprofessors.com/. Online; accessed 26-April-2018.

[13] Alon Zakai. Emscripten 1.37.37 Documentation. http://kripken.github.io/emscripten-site/. Online; accessed 23-April-2018.