Jonathan Kelaty

Manal Zneit

Course Scheduling Using Constraint Satisfaction

Project Description

Our project was on the problem of course scheduling at the university level. We approached the problem as a constraint satisfaction problem and implemented many techniques for searching the state space efficiently for large datasets of courses to schedule. We implemented several hard constraints when scheduling courses such as ensuring there are no instructors having courses with conflicting times, or rooms designated to be used at the same time for different courses. We implemented two search algorithms, which are dynamically selected depending on the number of courses to be scheduled to ensure we strike a balance for time complexity and soft constraints met for larger datasets, which are the primary focus for this project.

Team Member Contributions

- **Team:** Both members contributed toward developing the scheduling algorithm, discussing implementation ideas, and working through implementation decisions.
- **Manal Zneit:** Implemented the algorithm and the code to solve the course scheduling CSP.
- **Jonathan Kelaty:** Created the schedule visualization tool and scrapped and normalized the dataset of courses from Hunter.

Approach

Our basic approach was to use a backtracking search which attempts to assign each variable a value from its specified domain. When we have assigned all variables and all our constraints are met, the schedule is then returned and output to a file. For constraints, our two primary hard constraints are that there are no instructors with course time conflicts, as well as rooms with no time conflicts. We also use these constraints to prune subtrees of the search space to improve running time. For larger datasets, we use an iterative approach which does not perform backtracking. We also employ random restarts for the case when our algorithm gets stuck at a local minimum. For this approach, the schedule with the largest evaluation is returned.

Procuring the Dataset

In order to test our scheduling program, we wanted to use real-world datasets as opposed to course data produced programmatically, as we didn't know how to anticipate how courses would be distributed during the day, and we didn't want to unknowingly devise a dataset that was either far too constrained or not constrained enough, as this would not give us an accurate representation of how well our program performs on real-world data, which is its intended usage. In order to obtain real-world data, we decided to use course data for Hunter College's upcoming Fall 2020 semester on CUNY's course search online.

In order to scrape this data efficiently, we had to search for the largest number of courses to return on a single web page. CUNY's course search interface requires a minimum of two filters, which we set to in person classes and courses designated for the regular academic session, and also ensured to include courses that were already full, which accounted for the majority of

the courses that are being offered. In order to actually parse the data, the webpage was saved locally and jQuery was included in the header, and a simple script was written to parse the DOM and store the data for each course section in a JS object, after which it was stored as a JSON string and parsed to a CSV file. Initially, our CSV of courses contained approximately 2100 course sections.

Data Normalization

Upon investigation of the scrapped course data, a significant portion was determined to be useless, because Hunter had many courses with garbage designations for room, time, and instructor assignments. For example, a very large number of courses had “TBA” designated for one or more of the aforementioned course attributes, or “Staff” specified for the course instructor, which had us wondering, where is all the data? In addition, many courses were assigned at times that didn’t make sense, such as from 12:00am - 12:00am for every day of the week. As a small note, we also discovered that Hunter had many scheduling abnormalities such as a course which had 4 separate sections, but scheduled for the exact same time and day. Anyway, in order to normalize the course data, we went through each course attribute, one column at a time, and fixed any issues, such as duplicate text or invalid attribute assignments. We also deleted all courses that are unable to be scheduled, which consisted of all sections listed as independent study/research as well as some other niche courses that didn’t follow the schedule model we based our algorithm on, which to be clear, is for in-person courses which have a time, room, and instructor designation, which are our constrained variables.

Course Variables and Domains

Now that we've removed all unschedulable courses and normalized our remaining courses, we wanted to use this real-world dataset of courses to create a derived dataset which contains data that can realistically be obtained by a university which will be sent to the scheduler. When considered what this data would look like, we had to establish what our variables that we're assigning are and what their domains are. Our variables were fairly obvious, since each course section will be assigned a room, instructor, and timeslot (day(s) and time). When considering domains, we figured each course will likely have a target for the enrollment capacity for that it based on historical demand for a certain course. Based on this target capacity, the university should realistically have access to a database of all the rooms they have available to them as well as their maximum capacities, which can be referenced to generate a list of possible rooms which can be assigned for a given section, thus, this would generate the domain of rooms for each course section. For our dataset, we just assumed that all the rooms used across an entire department, were available to all course sections within that department, since we didn't have access to data for the rooms available at Hunter nor their maximum capacities.

For instructors, we similarly assumed that each department would be able to report which professors within that department would be qualified to teach certain courses, thus generating our domain for instructors for each section. The justification behind this is that we don't anticipate that the university itself is able to keep track of every single instructor's qualifications, but individual departments are more equipped to keep track of all faculty within their department, so it makes sense to put the onus on them to provide this data. For our dataset, we initially decided to follow a general rule that if a professor is assigned to teach a specific course, then they would

be qualified to teach any lower-level course within that course's department, which while this is not always a correct assumption, is generally true for many courses. Unfortunately, due to far too many courses not having an instructor assigned to them, many course sections were left with an empty set of qualified instructors, so we had to expand the domain of instructors for a given course to be all instructors within that course's department, similar to the room's domain. Note that at this point, our domains are highly constrained due to many courses not having any room or instructor assigned to them in Hunter's actual course data.

Finally, and most problematically, we had to consider what the domain for our timeslots would be. To be clear, we define a timeslot as a combination of the day(s) of the week and the time of day that a course section is scheduled to meet. The issue with this is that it's very difficult to define what the domain for this variable is. We have a very large number of combinations of days and times we can schedule, which makes our domain grow very large as we increase the granularity for times during the day. For example, we can consider scheduling courses only at the start of the hour, every hour during the day, or we can consider all timeslots at all 10 or 15 minute intervals during the day. More granularity makes our domain for the timeslot grow significantly which will increase the running time of the scheduler and reduce the consistency of assigned times across the entire schedule, which can be a subjectively perceived as a flaw in the schedule itself. However, less granularity in the timeslots results in a far more constrained domain, meaning we might not be able to find a valid schedule. Our program implementation defines the timeslot domain at 1 hour intervals, which we found worked very efficiently for a medium-sized dataset of approximately 500 courses, but was far too constrained to satisfy the full dataset that we generated. With that in mind, we still needed to figure out what the interface

would be for the user when generating the course data to pass to the scheduler. We felt that it made the most sense to allow a relative time specification for a given course, which would be either morning, afternoon, or evening. This tells our algorithm to assign a range of times that fit into the specified time as the timeslot domain for that course. What this allows the user to do is create multiple sections of the same course, but schedule them at varying times to account for students who prefer or need to take courses at specific times during the day.

Instructor Preferences

Our program also allows the user to input supplementary data for instructors' preferences, which we treat as soft constraints, meaning while our program attempts to satisfy as many of these as possible, it prioritizes finding a valid schedule given the hard constraints of room and instructor time conflicts. The data that the program expects is very simply just a table with each entry being the instructor, and then the relative time (morning, afternoon, or evening) that they would prefer to teach a course as well as a list of courses that they have a preference to teach. This data is completely optional so data can be missing or instructor entries can be entirely absent, and the algorithm will treat these cases as if the instructor has no preference. We imagined that this data would likely be procured by an institution by simply sending out a mass online survey to all instructors, and then a program could easily parse this data and format it to be accepted by the scheduler, and in the case that any instructors do not take the survey, this would not impact the scheduler's ability to generate a schedule. For our dataset, we simply manually created a small sample of pseudo-random preferences for a subset of instructors. The dataset is obviously not going to be truly representative of what a real-world dataset of

preferences would look like, but we simply do not have access to this information, and sending out a mass survey to all of Hunter's faculty without permission for the purpose of this project seemed inappropriate.

Evaluation

In order to evaluate our program, the most fundamental aspect we can look at is whether or not it manages to generate a valid schedule given the dataset of courses. When attempting to schedule the full data set of 1200 courses, we found that the algorithm simply wouldn't complete in any reasonable amount of time, if it all, indicating our dataset is far too constrained, which we anticipated early on when we had to supplement missing or obscured data from Hunter's actual course data. We identified the probable sources for the data being over-constrained as the randomization of course data that we didn't have access to or was absent, the distribution of course attributes' domains across departments due to many courses lacking any data, as well as the limited timeslot domains we implemented. We found that we can run our program with a dataset of approximately 500 courses, which completes in under 20 seconds when run locally.

Further objective evaluation was more difficult as we found it fairly difficult to identify objective parameters for which to judge the generated schedule. Similarly for a subjective evaluation, where you'd likely have to actually schedule an institution's courses for a semester, and gain insight from the students who had to navigate the schedule for themselves when signing up for their classes, where they'd likely be more sensitive to any significant scheduling issues that are present. We also have issues evaluating our program's schedule compared to Hunter's actual schedule due to a lack of 1:1 correspondence between the two due to a severe lack of data.

It wouldn't really make sense to compare a schedule of 500 courses to one with 2000+, because the larger schedule will just be more constrained and ultimately Hunter has access to more of the data required than we do, so it's unlikely we'd see a significant improvement.

Visualization

To accompany our project, a visualization tool was built to interact with our program and visualize the generated schedule in a very useful way. The frontend was built using React and allows the user to upload CSVs with the course data and instructor preferences to feed into our program. A separate Heroku app was created to handle API requests from the frontend app to run the python scheduler script and return the generated schedule. As a small implementation note, in order to overcome Heroku's 30 second timeout for requests, the data is sent as a POST request to the server, and the process is enqueued by a worker, after which the job ID is returned back to the front end. Every few seconds, the frontend app will ping the Heroku app until the process completes and the schedule is returned. The user can then interact with the schedule by hovering over the courses laid out on a timetable, and filter for the constrained variables such as room and instructor assignments to visually show that the constraints have been satisfied.

Future Work

Concerning future work for this project, it became very clear to us that the first step in making any significant improvement to our program would be gaining access to more data and ensure that it's normalized. We were also missing many other less important features in our program such as support for online classes, alternative session periods, and assigning multiple

instructors per course, just to name a few. Another area that we think could improve the subjective quality of the generated schedule would be to take into account the frequency for which a group of courses are taken concurrently, so that the program can apply a new soft constraint of not overlapping courses that are frequently taken together. This would be especially useful for incoming students who are likely all taking similar courses to satisfy core or major requirements.

Closing Remarks

Overall, this was a very enjoyable project to work on, especially thinking through all the possible considerations for how to algorithmically search through the state space and generate a valid schedule given a fairly wide range of possible constraints. I think there is still a significant amount of improvement that is left to be achieved for this project but was not fully realized due to the limited scope we had to work under, but will hopefully see more development over time. While this was not the focus of the project, I had also had a lot of fun making the visualization tool, and I'm glad I was able to make it so well-polished and properly deployed to be used by anyone who comes across this project.