GitHub Link: https://github.com/krishanu-datta/schedule_optimizer

Giving Students What They Want:

A New Way to Optimize Student Schedules

Krishanu Datta, Ravi Teja Penikelapati, Rashad Khan

Introduction

Almost every single human being living on Planet Earth operates on a schedule. Laborers have job schedules, students have class schedules, and bands have concert schedules. Often when these same people are off their daytime schedules, they are still following a schedule, perhaps one that they made, in the form of prioritization. People have obligations, responsibilities, hobbies and activities that each have a different level of importance and urgency to them. To complete their set of tasks, humans will internally or explicitly prioritize the tasks in order to form a schedule that optimizes their happiness. For instance, a child might first do the dishes and clean their room to please their parents, before deciding to watch their preferred sport of basketball instead of football on television. This paper focuses on developing an application which can apply these ideas of customization, optimization and prioritization, which people already unknowingly apply in their personal lives, to people's rigid daytime schedules. In particular, this paper looks at the perspective of a high school or college student and analyzes how their tastes and preferences in which subjects they want to enroll in, which professors or teachers they want to learn from, and what types of classes they want to take, can all be used to create an optimal schedule for them.

The schedule optimization application we designed is coded with Python in a Jupyter Notebook IDE. However, the brains of the application is the SAT4j MAXSAT solver, a tool which solves boolean satisfaction and optimization problems. The utilization of this solver is what allows the magic to happen. A normal SAT solver solves boolean satisfaction problems by

accepting a CNF, a string of clauses containing literals, and outputting whether or not the problem is true or false (can or cannot be satisfied). However, a MAXSAT solver accepts a weighted CNF, a string of weighted clauses containing literals, and outputs the set of literals which maximizes the weights of the CNF. Furthermore, the CNF fed into the solver has a series of hard clauses which the solver must satisfy with its optimal output and a group of soft clauses (prioritized by assigned weight) which the solver must attempt to satisfy with its output. In the case of this project, the hard clauses were used to ensure that the student took one and only one class in every time slot, and to guarantee that the student did not take multiple sections of the same subject in their schedule. The soft clauses were wielded to enforce the student's subject choices, professor choices and class type choices. In our application, the resulting output of the MAXSAT solver is a string of positive and negative literals, with the positive values corresponding to class IDs of the classes the student should enroll in.

Approach

Traditional approaches on schedule optimization vary based on level of constraints and complexity. There are massive companies who provide scheduling optimization for businesses and schools across the world. Some of these include Mimosa and Adesoft, who while seem to cater towards the institution, also offer benefits for the student: "Student scheduling (student sectioning) services offered by the software helps in optimizing the scheduling of students that should be placed in each section to minimize student conflicts arising from the students in different sections opting for the same course and balance section sizes" (AskWonder). We think this will work better than the current approach because we do not think there is a current approach to the exact problem we are attempting to solve.

GitHub Link: https://github.com/krishanu-datta/schedule_optimizer

Because we could not find a reliable, existing solution that answers the questions as our application, we could not anchor our goal to a specific value. The metric that measured the success of our application was time, because that is what our application is good for: saving time (and as a result money). For a student, figuring out what classes they will take can take days or even weeks, and if we could minimize that figure to seconds, the application would be a success. Outside of this specific context, schedule optimization is important for many festivals and conferences and by saving them time on this aspect of planning, they can make more money.

Methodology

The first step in building our application was creating an example situation where our program could be applied. This was done in a very typical manner: a brainstorm of ten distinct subjects each with two sections. Each section was assigned one of four time slots and one of five professors. The result was eighteen different sections of classes with unique characteristics. Then, the idea for what the application would accomplish was finalized: a student would input preferred characteristics of the unique classes and receive an optimized schedule according to their inputs. Next, we acquainted ourselves with the SAT4j solver. On the SAT4j website, the header indicates that the solver is a library in Java, implying the solver could be used easily in code; however, we quickly learned this was not the case. The MAXSAT solver we wanted to use was a jar file which was best applied in a command prompt. We discovered that the tool only accepted CNF files in a particular format known as the DIMACS format, so our next challenge became how to format the input data into the proper format for the solver. Learning the ins and outs of the DIMACS format was a step that proved particularly difficult. The format is very simple on the surface: the literals on each row are connected with OR operators and each row is

connected by AND operators; nevertheless, there were so many small intricacies to the format such as the number of variables label needing to be greater than the value of the largest variable and each row needing to end in a zero. One hiccup which hindered our progress for a couple days was saving the file as a WCNF file instead of a CNF file, a requirement special to the MAXSAT solver. After we had mastered the syntax of the SAT4j MAXSAT solver, we moved onto formatting the user data in Python so that it could input the data in a DIMACS format into the solver. We decided to not use scanners or system prompts to gain information, instead opting to ask the user to enter their preferences into variables. Very early on, Python proved to be the right choice for this assignment as it processed the spreadsheet with the schedule data with ease. The Python language and it's library pandas are built to transform data efficiently and that certainly helped with our project. The next portion of coding loops and functions to parse the data and print the DIMACS format proved to be fun and engaging. We enjoyed this aspect of the project because it gave us a chance to problem solve and apply fundamental computer science concepts such as if-else statements and for loops.

While printing the appropriate clauses in DIMACS format was a smooth step in our project, the following step of figuring out how to enter that information into the SAT solver was one of the more difficult aspects of the project. We initially believed that it would be possible to implement the solver directly in the code, but upon hours and hours of combing the internet for examples of the SAT4j solver being used while embedded in Python code, we found nothing. Thankfully, Python has built-in functions that allow us to create a WCNF file, open a pipe straight to the command shell, and read the output of commands written in that command shell. This process of learning to write a WCNF file, execute a command in the command shell and

process the output of that command took hours of troubleshooting and debugging until the application functioned as we wanted it to.

One additional hurdle during our process was learning to use clauses to require one and only one class per time slot using the "at most one" technique. This technique involves having a 2CNF with every combination of classes in a particular time slot where the two selected classes are negated such that you must select a negative version of one of them. Once we discovered this technique, our application behaved better.

After overcoming these challenges and following the steps discussed above, we created a functioning application which returned an optimized schedule for a student based on which subjects they want to enroll in, which professors or teachers they want to learn from, and what types of classes they want to take.

Results

By our own metric for success, our project was a success. The application consistently takes less than 2 seconds to complete. This is evidenced by the fact that the SAT4j SAT solver returns a value indicating the time needed to complete its operation. By combing through the command shell output, our code displays this figure along with the other output. Unfortunately, our results do not generalize well because the code and application has only been tested for one test scenario. If we had more time to work on this project, we would test our code against bigger and alternate scenarios working with different variables.

9 November 2020 CS 2800
GitHub Link: https://github.com/krishanu-datta/schedule_optimizer

Summary

While we applied the MAXSAT solver to solve a very specific scheduling problem for a student and their preferences, the application we created can be generalized to serve alternative purposes. For example, our program could be used to plan an outing to a colossal conference or festival with a vast selection of events and acts. The user could select their tastes in entertainment, time slots and locations that work best for them, and most importantly highlight which events they must attend during their trip. Then, our application could return a customized schedule for the user based on their preferences and freedom of choice. This is just one way in which our creation could be modified to fit another scheduling problem, but the challenge of creating a customized and optimized schedule is one that persists in many contexts. For this reason, we feel our application is helpful and innovative for civilization--because there is a consumer need for this kind of technology.