Student Schedule Planning With Integer Programming Using AMPL

William Kanieski

Friday, May 5th, 2023

Abstract:

A major issue for college students and advisers alike is how to schedule college courses throughout one's academic career. There are numerous specific factors to take into account for different majors, including course and credit hour requirements as well as electives and general education courses that must be satisfied prior to graduation. Additionally, many courses require previous courses as prerequisites, or can only be taken during certain semesters. Thus, in order to ensure that students can meet the academic requirements for their major, their schedules must be planned out in advance with these factors in mind. This is especially important for students pursuing multiple majors or minors, and failure to prepare an adequate schedule might result in having to attend additional semesters in order to graduate. Using AMPL and the course requirements outlined on Ohio University's website, our model aims to use integer programming to generate valid student schedules with the goal of minimizing total classes or credit hours taken while still satisfying all the requirements needed for a student's major. In this paper, we will go into detail about the structure of our model, how it can be adapted to multiple different majors, and how we can improve its structure for future projects.

Literature review:

Design of the model:

AMPL is a programming language designed for the purpose of modeling and solving large algebraic problems involving optimization and scheduling. Its format closely resembles the mathematical layout of most linear, discrete, and nonlinear optimization problems. An AMPL model typically begins with sets of objects or numbers which can be used to iterate over variables or constant parameters. These variables or parameters can be constrained to be integers or binary numbers, or made to fall within a certain range. After all variables and parameters are declared, a line of code is written to maximize or minimize a certain objective value, followed by a list of sets of constraints involving the previous variables and parameters. Once written, this model can then be loaded along with a data file containing the values of the model's parameters. The model is then solved with a program suited to the problem's requirements. An optimal objective value is then displayed, and the user can find the values of the variables necessary to achieve it using console commands. If no solution exists, the problem is deemed infeasible. The problem might also be unbounded, have infinitely many solutions, or be degenerate.

Example data sets:

Limitations and potential improvements:

The most significant and obvious limitation of our model is in regards to the actual time intervals during which each course takes place. Because it only focuses on semester availability and does not factor in time of day, it is possible for our program to generate a schedule which necessitates taking two courses that are offered at the same time, requiring the student to be in two places at once. A potential solution to this problem would involve adding new constraints and variables to the model involving time slots and student availability. These constraints would prevent scheduling conflicts by dividing each day in a student's week into discrete time intervals during which a student may take at most one class. This could be implemented in the main model with the following code:

Our model also assumes that students will pass all of their courses on the first try, and that they will not change their minds or alter their schedules due to new classes being offered or changing interests. In order to fix this, our model must be able to accommodate revision at the beginning or end of each semester. This can be done by reexamining the old model after the student has taken some courses and adding constraints for past semesters, setting the variables for classes already taken to 1 and classes not taken to 0. If a student needs to repeat a course, the constraint requiring each class to be taken only once can be removed for that specific class by having the rule only apply to a set of courses that does not include it.

Another complexity of Ohio University's prerequisite system is that certain course prerequisites can be met by more than one class, or by various combinations of courses meeting certain requirements. A simple example of this occurs in the Bachelor of Science in Computer Science major, where CS4250 requires either MATH3200 (Applied Linear Algebra) or MATH3210 (regular Linear Algebra) to be taken as a prior course. However, these situations are usually rare, and in our computer science model we mitigated this by simply having MATH3200 as a prerequisite for CS4250, as the two linear algebra classes effectively functioned as the same course. In general, if we were faced with a major where these situations were more common, we could implement a set similar to the "requirements" set, but for each course, where a minimum number of courses in the set must be taken as prerequisites instead of only one. This would not be practical for most majors, however, as it would increase the number of constraints as well as the complexity of the datasets for only a marginal payoff in terms of accuracy.

Additionally, it might be impossible for students with certain major and minor combinations to complete their degrees within eight semesters. While this might not be optimal, it should still be incorporated into the model as an option. Some majors might even allow students to graduate early, depending on how soon they can take certain courses or finish their capstone projects. We can address this by increasing the number of semesters students are allowed to attend, and having a binary variable that determines whether a student attends a given semester. These binary variables can then be tied into the other constraints, and the minimum number of semesters required or the maximum number of semesters allowed can then be hardcoded into the dataset for each particular major. We can also expand the model by including summer semesters, as many STEM majors will take summer classes if they have demanding programs. However, many students may have jobs over the summer and would not be available to attend these semesters. Therefore, it might also be prudent to introduce new binary parameters here to determine whether summer courses are an option.

Conclusion:

Course scheduling is a complex problem which must consider a multitude of different factors. Some academic programs are more complicated than others, and it is difficult to construct a general model for academic scheduling that is guaranteed to be tailored to an individual student’s needs. However, our AMPL model provides a more standardized method to generate a test schedule that will be suited to a given major, as long as we can provide the proper dataset for the program. We can then examine these test schedules and determine whether or not they are feasible, and if necessary, the model can be revised to provide a sequence of courses that better fits a student’s interests and program requirements. The model can also be used to test the feasibility of an existing schedule, and determine what aspects of it might need to be changed for a given program, or highlight certain courses that a student might be able to test out of in order to satisfy their major requirements in a shorter period of time. In this way, integer programming and discrete optimization can help both students and advisers to come up with improved schedules and gain a better understanding of how to develop one’s academic career.

References: