Operations Research II Final Project

Ashley DiOdoardo, Emily Ma, Jenny Ou, and Reed Luttmer

December 2023

1 Introduction

1.1 Motivation

Carnegie Mellon has a rigorous Mathematics degree program regardless of the classes a student takes over the course of their 4 years here. Students, every semester, have to make decisions on what classes to take while considering interests, major requirements, and difficulty of classes. All else aside, the difficulty of classes has a impact on the student experience and ability to participate in activities outside of the classroom. The goal of this project is to create a schedule that minimizes the Faculty Course Evaluation (FCE) hours, which are the average number of hours spent by students on all course-related activities for a given class. Specifically, we aim to minimize the maximum number of FCE hours in any semester.

FCE hours provide a useful way to measure how difficult and time consuming a class actually is based on actual student feedback. Thus, we thought it would be a sufficient measure to minimize. Our 8-semester schedule can be used as a guide for future students to plan out a manageable schedule for their time at CMU.

1.2 Previous Projects

One of our goals is to improve and expand upon previous projects of a similar topic. One project, from Fall '19, has provided a solution, except it has some assumptions that we have made stronger. For instance, in their paper, they note that they assume any class can be taken any semester - which is not the case and provides solutions that are not reasonable. In addition, their solution is solely math classes. Our project, instead, focuses on the core curriculum of the Mathematical Science (Statistics and Operations Research concentration) and provides a more thorough planning guide which includes statistics, business, and computer science courses as options for the depth electives. Furthermore, their provided solution includes taking 700 level courses as a first-year student. This is highly unlikely to do, and our solution will provide a more practical solution for the general student. Therefore, we have excluded graduate level courses and included solely undergraduate level classes.

Another project, from Fall '18, provides a solution, but it only contains classes for 4 semesters. Also, this schedule has prerequisite conflicts such as taking 21-270 and 21-370 in the same semester. This is also an issue as 21-370 is only offered in the Fall. Our project will focus on creating a schedule for the full 8 semesters, as well as ensuring that all prerequisite requirements are met to provide a more accurate solution to the problem. The solution provided in the past project includes taking up to 6 math classes a semester, which is not manageable for the average student. We will focus on balancing the quantity of math courses by including other major requirements and Gen-Ed courses. Overall, improving the faults in these two projects were a focus of our project.

2 Formulating ILP

2.1 Decision Variables

We first want to create decision variables. We are trying to figure out the optimal schedule, so the decision variables will center around which courses to take and when. Specifically, we have the binary variable $x_{i,j}$ to represent whether course i is taken during semester j or not. We will let N denote the number of courses and S denote the number of semesters, and thus $x_{i,j}$ is defined for $i \in [N]$ and $j \in [S]$.

2.2 Objective Function

Our goal is for the student to not have any semesters that are too busy. To that end, our objective is to minimize the maximum FCE in any given semester. Thus, we let f_i represent the FCE of course i for $i \in [N]$. Then the objective function can be written as

$$\min \quad \max_{j \in S} \left\{ \sum_{i=1}^{N} f_i \cdot x_{i,j} \right\}.$$

We run into a problem however. Our goal is to have an integer linear program, but the max function isn't directly linear, and would therefore be hard to encode into the software we will use. Hence, we create one more decision variable a which will represent the max FCE of any semester. Then our objective is to minimize a, but we must also add constraints ensuring that a is at least the total FCE of each semester:

$$a \geq \sum_{i=1}^{N} f_i \cdot x_{i,j} \quad \forall j \in [S].$$

Then in a feasible solution, a will not necessarily represent the maximum FCE of a semester (it could be higher than the maximum), but in any optimal solution, it will.

2.3 Constraints

We have quite a few constraints to add. First of all, any particular class can not be taken more than once. So

$$\sum_{j=1}^{S} x_{i,j} \leq 1 \quad \forall i \in [N].$$

Moreover, there is a maximum number of units that a student is allowed to take per semester before their schedule is considered an overload. We will call this variable U_o (for the average Mellon College of Science student, $U_o = 54$). Then to ensure that no semester is more than U_o units, we let u_i represent the number of units that course i is worth. We obtain the constraint:

$$\sum_{i=1}^{N} u_i \cdot x_{i,j} \leq U_o \quad \forall j \in [S].$$

Likewise, in order to graduate, there is a minimum number of units a student must take during their time at college, which we will call U_g (for CMU students, $U_g = 360$ units). To ensure that at least this many units are taken, we have the constraint:

$$\sum_{j=1}^{S} \sum_{i=1}^{N} u_i \cdot x_{i,j} \geq U_g$$

We must then deal with prerequisites. Many courses have multiple courses that could fulfill the prerequisites. For example, the probability course 21-325 has a prerequisite of 21-268 or 21-259 or 21-269 or 21-256 (which are all 3D calculus courses). We dealt with this by creating groups of classes, labeled $G_1, G_2, \ldots, G_\gamma$, where γ is the total number of groups. Then we have the set $P = \{(i, g) \in [N] \times [\gamma] : \text{course } i \text{ has group } G_g \text{ as a prerequisite}\}$. So in our example, we would put the four calculus courses above in one group (say G_1), and then there would be the ordered pair $(21325, 1) \in P$ representing the fact that a course from group G_1 must be taken as a prereq for the course 21-325. Importantly, only one course from the group needs to be taken, not multiple or all of them, and this will always be the case. We therefore end up with the constraint:

$$\sum_{k=1}^{j-1} \sum_{h \in G_g} x_{h,k} \geq x_{i,j} \quad \forall j \in [S],\ \forall (i,g) \in P$$

which dictates that if course i is taken in semester j, and if group G_g is a prerequisite for course i, then at least one course h in the group G_g must be taken in a semester before j (so notably a semester k between 1 and $j-1$, inclusive).

We also have to encode the classes required to graduate. We define requirements $R_1, R_2, \ldots, R_\rho$, which are groups of required classes. Each set R_r contains all course numbers $i \in [N]$ that count towards requirement r, and in order for the requirement to be satisfied, at least one course $i \in R_r$ must be taken (importantly, not all courses have to be taken, one is enough). For example, an Operations and Statistics concentration student must take one probability course, but they can chose between 21-325, 15-259, and 36-218 to fulfill this requirement. So we would put all three of these courses in one requirement R_r. Then to make sure all requirements are met, we have the constraints:

$$\sum_{j=1}^{S} \sum_{i \in R_r} x_{i,j} \geq 1 \quad \forall r \in [\rho].$$

There is however one requirement that can't be formulated as above: a student must take at least 45 units of depth electives, from a long list of depth electives. For this, we define the set D to be all course numbers $i \in [N]$ that count as depth electives, and then we have the constraint:

$$\sum_{j=1}^{S} \sum_{i \in D} x_{i,j} \cdot u_i \geq 45.$$

The final set of constraints we must consider concern the times during which courses are taken. We need to make sure that students aren't enrolled in two courses that have lectures at the same time, and we also must ensure that a student doesn't plan to take a Spring-only class in the Fall or a Fall-only class in the Spring. To do this, we divide the week into T time slots. For example, at CMU, classes start every half hour, so we divided the week into 30-minute time slots, ranging from 8-8:30am on Monday to 8:30-9pm on Friday. We also assume that any class offered in the Spring will be at the same time every Spring, and that any class offered in the Fall will be at the same time every Fall. Then we have the binary data variables $y_{i,1,t}$ and $y_{i,2,t}$ which are 1 if course i is offered during time slot t in the Fall (season 1), and Spring (season 2), respectively, and 0 otherwise. To make sure only one class is taken per time slot per semester, we add the constraints:

$$\sum_{i=1}^{N} x_{i,j} \cdot y_{i,2-j\%2,t} \leq 1 \quad \forall t \in [T],\ \forall j \in [S]$$

which sum over all courses that could be offered in the given time slot and semester and make sure at most one of them are taken. In this context, we are using % as the mod operator to denote the remainder when j is divided by 2, so that $2 - j\%2$ evaluates to 1 when j is odd (since the odd semesters are in the fall) and 2 when j is even.

Finally, we ensure that no Spring-only class is taken in the fall, and vice-versa. If a course i is offered in the Fall, it will meet during at least one fall time slot, and then $\sum_{t=1}^{T} y_{i,1,t} \geq 1$, whereas if course i isn't offered in the Fall, then we have $\sum_{t=1}^{T} y_{i,1,t} = 0$. If course i isn't offered in the Fall, we want $x_{i,1}, x_{i,3}, \ldots, x_{i,2\lceil S/2 \rceil - 1}$ to all be 0, which we can encode by forcing their sum to be 0. And we know their sum will be at most 1 (we encoded this above because each course can be taken at most 1), so enforcing that the sum is less than or equal to 1 wouldn't have any consequences. Thus we add the constraints

$$\sum_{k=1}^{\lceil S/2 \rceil} x_{i,2k-1} \leq \sum_{t=1}^{T} y_{i,1,t} \quad \forall i \in [N]$$

to make sure that course i won't be taken in the Fall if it isn't offered in the Fall. Likewise, we have the constraints

$$\sum_{k=1}^{\lfloor S/2 \rfloor} x_{i,2k} \leq \sum_{t=1}^{T} y_{i,2,t} \quad \forall i \in [N]$$

for courses that aren't offered in the Spring.

2.4 Overall Variable Definitions

After all this work, we end up with the following variables. Of the variables below, only $x_{i,j}$ and a are the decision variables, all other variables are *data variables*, meaning that they are being input into the program, their values aren't being changed to find an optimal solution.

$$N = \text{the total number of courses}$$

$$S = \text{number of semesters}$$

$$x_{i,j} = \begin{cases} 1, & \text{course } i \in [N] \text{ is taken in semester } j \in [S] \\ 0, & \text{otherwise} \end{cases}$$

$$f_i = \text{the FCE of course } i$$

$$a = \text{max FCE across semesters (enforced via constraints)}$$

$u_i =$ how many units course i is
$U_o =$ max number of units in a semester to avoid overloading
$U_g =$ number of units needed to graduate
$G_g = \{i \in [N] : \text{course } i \text{ is part of the } g\text{'th group of classes}\}$
$\gamma =$ The number of groups G_g
$P = \{(i, g) \in [N] \times [\gamma] : \text{course } i \text{ has a course from group } G_g \text{ as a prereq}\}$
$R_r = \{i \in [N] : \text{course } i \text{ is a course that can be taken to fulfill requirement } r\}$
$\rho =$ The number of requirements R_r
$T =$ number of time slots in a week

$$y_{i,1,t} = \begin{cases} 1, & \text{if course } i \in [N] \text{ is offered during time slot } t \in [T] \text{ in the Fall} \\ 0, & \text{otherwise} \end{cases}$$

$$y_{i,2,t} = \begin{cases} 1, & \text{if course } i \in [N] \text{ is offered during time slot } t \in [T] \text{ in the Spring} \\ 0, & \text{otherwise} \end{cases}$$

2.5 Overal ILP

Thus we obtain the ILP:

minimize a subject to

$$a \geq \sum_{i=1}^{N} f_i \cdot x_{i,j} \quad \forall j \in [S] \qquad \text{(ensure } a \text{ is the max FCE in any semester)}$$

$$\sum_{j=1}^{S} x_{i,j} \leq 1 \quad \forall i \in [N] \qquad \text{(no class is taken multiple times)}$$

$$\sum_{i=1}^{N} u_i \cdot x_{i,j} \leq U_o \quad \forall j \in [S] \qquad \text{(no semester is a unit overload)}$$

$$\sum_{j=1}^{S} \sum_{i=1}^{N} u_i \cdot x_{i,j} \geq U_g \qquad \text{(the required number of units to graduate are taken)}$$

$$\sum_{k=1}^{j-1} \sum_{h \in G_g} x_{h,k} \geq x_{i,j} \quad \forall j \in [S],\ \forall (i, g) \in P \qquad \text{(prerequisites are satisfied)}$$

$$\sum_{j=1}^{S} \sum_{i \in R_r} x_{i,j} \geq 1 \quad \forall r \in [\rho] \qquad \text{(required courses are taken)}$$

$$\sum_{j=1}^{S}\sum_{i\in D} x_{i,j}\cdot u_i \geq 45 \qquad \text{(45 units of depth electives are taken)}$$

$$\sum_{i=1}^{N} x_{i,j}\cdot y_{i,2-j\%2,t} \leq 1 \quad \forall t\in[T],\ \forall j\in[S] \qquad \text{(no classes are taken at the same time)}$$

$$\sum_{k=1}^{\lceil S/2\rceil} x_{i,2k-1} \leq \sum_{t=1}^{T} y_{i,1,t} \quad \forall i\in[N] \qquad \text{(no Spring-only course scheduled for Fall)}$$

$$\sum_{k=1}^{\lfloor S/2\rfloor} x_{i,2k} \leq \sum_{t=1}^{T} y_{i,2,t} \quad \forall i\in[N] \qquad \text{(no Fall-only course scheduled for Spring)}$$

$$x_{i,j}\in\{0,1\} \quad \forall i\in[N],\ j\in[S]$$

3 Implementing ILP with CMU-Specific Data

3.1 Data Sources

Our data for the project consisted of both FCE Data and Course Scheduling Data. The FCE data was obtained from the FCE database for the following semesters: Fall ’21, Spring ’22, Fall ’22, Spring ’23. The category “Hrs Per Week” was used as a measure for how hard and time consuming a class is. We averaged the FCE for each course across the four semesters in which data was collected from.

Our Scheduling Data was obtained from the CMU Schedule of Classes. We obtained the days and times that courses are offered for the Spring and Fall semesters (Fall ’23, Spring ’24 as that is what is currently viewable on the website). Furthermore, for each of our courses, we looked it up in the Schedule of Classes and obtained any prerequisite requirements. Here, we also obtained the actual units for each course. We formatted our data using Google Sheets, and transforming it into the necessary formats (such as for date/time of course offering) following the outline mentioned in our constraints (Section 2.3) for the ILP.

We selected classes that are required in the core curriculum as well as math, computer science, business, and statistics courses that can count towards the required depth electives for the major. We also included general education requirement courses that are outlined in the next section. There were a total of 96 courses that data was retrieved for.

3.2 Assumptions

In order for our ILP to output an accurate schedule in a reasonable amount of time, we had to make some initial assumptions. First, we assumed that the Schedule of Classes for Fall '23 and Spring '24 will be representative of the course offerings for all four years. This assumption is reasonable, as courses tend to be offered at the same time every year. Furthermore, we had to encode generic general education (Gen-Ed) requirement courses in order to meet the minimum number of units to graduate. To do so, we encoded 9 unit Gen-Ed requirements with various FCE units (between 5 and 9). We included nine of these courses at various times in both semesters. From our experience, typical Gen-Ed courses that students take vary between 5 and 9 FCE units and are offered various times and days. Thus, we believe these assumptions to accurately represent course offerings at CMU.

We assumed that the students come to CMU with AP credit for both Calculus I and Calculus II, as this is the case for most mathematics majors. There is a general outline for when courses should be taken, including first-year courses and the Engage requirements. We hard coded the classes in the the table below in order to follow the suggested timeline to complete the Engage requirements, as well as to complete first year requirements and necessary coursework (such as 21-128 and 38-101). It would be atypical for a student to take these courses outside of the suggested timeline, thus we fixed them for all outputs as follows:

Course #	Course Name	Semester
21-128	Mathematical Concepts & Proofs	1
Various	First-Year Writing Requirement	1 or 2
38-101	Eureka Discovery & Impact	1
38-230	Engage in Wellness Looking Inward	4
38-330	Engage in Wellness Looking Outward	5
38-430	Engage in Wellness Looking Forward	7
38-110	Engage in Service	7
38-220	Engage in Arts	7

Lastly, there are special topics statistics courses (36-46X) that vary in topics and offering times across semesters. For clarity and accuracy, we combined all of the special topics courses into one general special topics course, by averaging the FCEs for all of the offerings in the data. We selected at time in the spring and fall semesters that is accurate based on past data. Therefore, potential solutions will contain a general offering in which the student can choose from the semester-specific topic offerings.

4 Solving with Gurobi

For solving the ILP, we chose to use Gurobi Optimization Software. Though the software requires a purchased license to use, we had previously obtained an academic license from our 21-292 Operations Research I class. Thus, due to our familiarity with the software and availability, we chose to use it for the project. We coded our ILP using Python (code can be seen at the end of the paper) and the Gurobi software. We used the Python library Pandas in order get the data into Python and convert it into the necessary data structures for the model. When run locally on a computer from 2020, it took approximately 2 hours to produce a solution.

5 Solving with Heuristic Algorithm

In addition to the Gurobi optimization, we also created a heuristic algorithm in Python. Given course schedules with corresponding units and FCEs, course prerequisites, and graduation requirements, the algorithm outputs a feasible solution. The algorithm followed the same assumptions as the Gurobi model's, with one additional: each of the first six semesters has five classes total. Three of the classes must be technical and two are nontechnical.

After reading in the data files for each course, we created a dictionary with the class as the key and a list of the prerequisites needed in order to take said class as the value. Similarly, we created a second dictionary with each course, k, as a key, where the value was a list of classes whose prerequisites course k can be used to fulfill. Because there were a limited number of technical classes under consideration, a list of them was manually included. The next step included categorizing the classes which satisfied various graduation requirements, such as depth electives or taking a calculus course. Lastly, several boolean variables were initialized to indicate whether a specific requirement had been satisfied yet along with variables for the number of semesters to create a schedule for and the current number of depth and overall units.

We decided to implement a greedy algorithm in order to determine which classes to add to the current semester schedule. For each semester, among the classes that were not yet chosen, our algorithm picked the class that was the prerequisite for the most number of classes using our dictionary that we initialized above. For the class that was picked, the algorithm checked if all necessary prerequisites for the course were satisfied as well if there a section offered at the same time as a free slot in the schedule. The final condition was checking if the course was technical/nontechnical and if so, had the maximum three technical/two nontechnical condition had been satisfied yet. If these conditions were both satisfied, we added this course to our schedule and updated the graduation requirement indicator variables. We continued this "picking" process until five classes were added to the current semester schedule.

It is important to note that a couple of courses were hard-coded due to assumptions on the model. First-year writing, Eureka, and 21-128: Concepts of Mathematics, were all added to the first semester. Similarly, the MCS Engage Requirements was added manually to each schedule, as was specified previously in Section 3.2.

The final output of this algorithm was a feasible course schedule.

6 Analyzing Results

In this section, we present a comparative analysis of our results with respect to various benchmarks, including:

6.1 Results from Gurobi

We first present the results obtained using Gurobi. These results serve as a foundational comparison for our findings.

The optimal schedule is as follows:

Fall 1 (38.7)	**Spring 1 (38.8)**	**Fall 2 (38.8)**	**Spring 2 (38.8)**
21-241 21-128 03-135 76-101 38-101	21-228 21-261 38-304 Gen-Ed Gen-Ed Gen-Ed	21-259 21-373 70-122 21-201 Gen-Ed Gen-Ed	21-292 15-110 73-102 03-161 38-230 Gen-Ed
Fall 3 (38.8)	**Spring 3 (38.7)**	**Fall 4 (38.2)**	**Spring 4 (38.7)**
21-325 21-369 21-469 03-125 38-330	21-374 36-226 36-410 33-141 Gen-Ed	21-441 21-393 36-401 38-110 38-220 38-430	73-103 73-230 36-402 70-371 03-133

Overall, the schedule is correct. For each course, all of the prerequisites have been satisfied in prior semesters and all of the graduation requirements have been satisfied. Moreover, the average number of FCEs for each semester, is within a similar range of 39 hours. Because the

FCEs for each semester is within a similar range, this solution helps achieve the objective of minimizing the maximum FCE among the 8 semesters. We can see that the optimal schedule is extremely similar to our personal experiences so far with a healthy balance of technical and non-technical courses added to each semester.

One interesting thing to note in the schedule is that 38-304: Reading and Writing Science, is scheduled for Spring 1. Although this is a spring-only course, it is typically taken during Spring 3 rather than Spring 1. However, this is not a significant issue because it is not required, but recommended to be taken in Spring 3. Additionally, in Spring 4, 73-103: Principles of Macroeconomics and 73-230: Intermediate Microeconomics are both scheduled. This is not an issue either because 73-103 is not a prerequisite for 73-230, but students do not typically take these courses in the same semester.

6.2 Comparison to Heuristic Algorithm's Schedule

We want to compare the Gurobi optimal solution with the feasible one produced by the heuristic algorithm.

The feasible schedule using the greedy approach is as follows:

<table>
<tr><th>Fall 1 (43.0)</th><th>Spring 1 (41.6)</th><th>Fall 2 (44.1)</th><th>Spring 2 (44.8)</th></tr>
<tr><td>21-241
21-128
33-141
76-101
38-101</td><td>21-259
21-228
15-110
Gen-Ed
Gen-Ed</td><td>21-260
21-325
15-112
Gen-Ed
Gen-Ed
21-201</td><td>21-292
21-270
03-125
73-102
36-226
38-230</td></tr>
<tr><th>Fall 3 (47.3)</th><th>Spring 3 (41.1)</th><th>Fall 4 (43.9)</th><th>Spring 4 (38.1)</th></tr>
<tr><td>21-355
21-373
21-370
36-401
70-122
38-330</td><td>21-374
21-420
03-121
36-402
Gen-Ed
38-304</td><td>21-469
21-393
21-441
73-103
Gen-Ed
38-110 / 38-220/ 38-430</td><td>73-230
36-410
70-371
03-133
Gen-Ed</td></tr>
</table>

Compared to the optimal schedule, the feasible schedule had higher FCEs for all eight semesters, but were still relatively close to one another with the exception of Fall 3. The higher FCEs are likely due to the additional constraint of three technical and two nontechnical courses for each semester. Because of this assumption, more math classes were assigned to the schedule. Math classes typically have higher FCEs than the average nontechnical course. Interestingly, the same number of nontechnical classes were assigned to both of the schedules, although not in the same semesters.

If planning a schedule that would allow you to have the most number of course options in following semesters, the heuristic algorithm may work better by choosing courses based on the number of courses it is a prerequisite for, especially in the earlier semesters. However, for a college experience with the work relatively evenly distributed among the eight semesters, the optimal solution is a possible way of achieving this goal while also ensuring that all of the graduation requirements are satisfied in time.

6.3 Comparison to Past Papers' Schedules

Next, we compare our derived schedules with those documented in prior papers. This comparative analysis highlights the deviations, improvements, or similarities between our schedules and those previously published.

There was a similar study from Fall 2019 where the students also scheduled courses for a mathematical sciences major. Their goal was to plan a schedule for a general math major that minimized the maximum number of FCEs and minimized the hours of free time during a student's week. Essentially, they wanted to a schedule that had the least amount of free time between classes as possible, so students can take classes back to back rather than having many small blocks of unproductive free time. Their results are as follows:

Semester	Mathematics Courses	General Electives	Time Between Courses per Week
1	21-127 and 21-241	33-767 and 1 general elective	5 hours
2	21-120 and 21-373	2 general electives	12 hours
3	21-122 and 21-623	2 general electives	12 hours
4	21-259, 21-355 and 21-738	1 general elective	3 hours
5	21-341 and 21-765	36-225 and 1 general elective	12 hours
6	21-228 and 21-882	2 general electives	8 hours
7	21-260	15-390 and 2 general electives	2 hours
8	21-356	15-810 and 2 general electives	0 hours

The maximum FCE units per semester using this schedule is 33.89 units

Total time between classes weekly for all semesters is 54 hours

Their schedule was mostly focused on scheduling only the math classes (21-xxx) over 8 semesters and when to take general electives. However, we chose to schedule all of the courses an Operations Research and Statistics major is required to take, besides the general electives. These included math, statistics, economics, accounting, computer science, and the MCS core courses such as the Engage requirements, the physical science requirement, the life science requirement, etc. The order of math classes were similar in the beginning, but began to diverge at semester 3/4. The previous students also allowed graduate courses in their schedule, which could be biased in terms of FCEs since grad students tend to spend less time on grad courses than undergrads do. They were also able to achieve a lower maximum FCE unit per semester of 33.89 compared to our maximum FCE unit of 38.8, although their solution doesn't meet the 360-unit graduation requirement.

There was another similar study conducted in Fall 2018. Their goal was to plan a schedule for a general math major, allowing studemts to choose from a list of depth electives that give the mathematics degree some "value" and take the easiest general education electives. They used a valuation system by using the overall scores from FCEs to evaluate the depth electives based on their value. The students used Integer Programming to determine which depth electives to take and a greedy algorithm to choose the general education electives that required the lowest FCEs. It should be noted that they assumed that all the courses a student wishes to take for a particular semester does not have any scheduling conflicts. Their results are as follows:

Semester 1	Semester 2	Semester 3	Semester 4
21-120 21-127 36-225 21-292 76-101 70-100	21-122 21-228 21-241 21-270 21-370	21-259 21-260 21-341 21-355 21-300 21-371	21-356 21-373 21-301 21-374 21-484

Their schedule was also mostly focused on scheduling only the math classes (21-xxx) over the four years and didn't include when to take general electives. They also planned the schedules by years, indicating which classes to take in a certain year, rather than a certain semester. As stated earlier, we were able to to schedule all of the courses an Operations Research and Statistics major is required to take, besides the general electives. There were also a few feasibility concerns with their outputted schedule, even when considering each "semester" as denoted in their table as a year. For example, 36-225 is a fall-only class and requires 21-120 as a pre-req, but both are under the same academic year. 21-292 is recommended to be taken in year 1, but requires 21-241 and 21-228 which are both scheduled in year 2. Moreover, 21-270 is a spring-only class and is a pre-req for 21-370, but both are scheduled in year 2. The order of math classes were similar in the beginning, but began to diverge at semester 3/4. This may be because their solution model used the most recent catalog (2018) and major requirements, but the course catalog may have been updated in the recent years.

While prior studies focused primarily on math courses, our schedule prioritized a wider array of general education requirements requirements, omitting bias potentially introduced by prioritizing specific courses (such as grad classes) or overlooking critical prerequisites and class time constraints. This holistic approach ensured a well-rounded curriculum for an Operations Research and Statistics major. By addressing these broader considerations, our schedule was not only feasible but also expanded the scope, enabling a more comprehensive optimized schedule for future students pursuing this major

6.4 Comparison to Published Schedule

Furthermore, we will compare our schedule against the CMU suggested schedule on the CMU website. The suggested schedule is as follows:

Fall 1 (44.3)	**Spring 1 (50.4)**	**Fall 2 (36.5)**	**Spring 2 (45.3)**
21-120 21-241 38-101 76-101 99-101 Technical Breadth	15-110/112 21-128 21-122 Technical Breadth Gen-Ed	21-201 21-228 21-259 73-102 Technical Breadth	21-260 21-292 38-230 70-122 Technical Breadth Gen-Ed
Fall 3 (38.9)	**Spring 3 (49.9)**	**Fall 4 (48.6)**	**Spring 4 (46.1)**
21-369 21-325 73-103 Depth Elective 38-330	36-226 36-410 73-230 Depth Elective Science & Society Cultural/Global	21-393 36-401 Depth Elective 38-110/220/430 Gen-Ed Free Elective	36-402 Depth Elective Depth Elective Gen-Ed Free Elective

While sharing a similar pathway in math course scheduling, our analysis revealed a contrasting spread of FCEs over the suggested four-year period. Their recommended plan exhibited a wider range of FCEs, fluctuating between 36.5 to 50.4 hours per week. In contrast, our optimized schedule showcased a more consistent distribution, ranging from 38.5 to 38.8 hours per week. In other words, our schedule was able to minimize the variance in FCEs, ensuring a more balanced and manageable workload for students. Notably, our selection of depth electives aimed at minimizing the maximum FCE, instead of leaving these choices to individual student discretion (like the recommended schedule did). Additionally, our ordering of non-math courses, including economics and accounting, differed.

7 Conclusion

Our study in optimizing college courses has yielded interesting observations. We saw that our schedule managed to evenly distribute the workload across all semesters. This resulted in a balanced academic course load, ensuring students wouldn't face overwhelming spikes in coursework at any point during their college experience. The optimized schedule was also very similar to our own academic paths and mirrored the courses many of us have taken or planned to take. This correlation emphasizes the relevance and applicability of our findings to real-life academic journeys, further validating the importance of thoughtful course planning and optimization for a smoother college experience.

However, our study isn't without its limitations. The dynamic nature of class schedules and the variability in FCEs due to individual professors pose challenges in creating a universally applicable scheduling strategy.

Broadening this approach to encompass a variety of majors, minors, and concentrations beyond Operations Research and Statistics could offer valuable insights into diverse academic schedules. We could also consider optimizing the course planning strategy to minimize the overall FCE over the four-year period. Prioritizing classes that might increase workload but significantly contribute to future job prospects, such as those in machine learning, finance, or programming, could be potentially beneficial for students as well. Another modification could be to minimize the FCE during senior year, so students can have more time to apply to jobs and/or graduate schools.

Our study holds practical significance in addressing the challenge of managing academic loads in college effectively. Enhancing the scheduling process could help students in balancing their academics with other non-academic interests such as extracurriculars, sports, etc.

8 Appendix: Code for Gurobi Solver

The following four pages show the Python code used to encode our ILP and run it through the Gurobi optimization software:

```python
from gurobipy import *

#------------------Defining the model---------------------------
# Initialization. The name is arbitrary
model = Model('take3')

#-----------------Importing Data-----------------------------

import pandas as pd
import numpy as np

sched = pd.read_csv("PD_Schedule.csv")
units = pd.read_csv("PD_Units.csv")
PR = pd.read_csv("PD_PR.csv")
groups = pd.read_csv("PD_Groups.csv")
req = pd.read_csv("PD_Req.csv")

semCount = 8 # semCount is the number of semesters
courseCount = len(units) # courseCount is the number of courses

# Schedule Matrix
fall = sched[sched.Sem == "F"]
fall = fall.drop("Sem", axis = 1)
fall = fall.drop("Num", axis = 1)

spr = sched[sched.Sem == "S"]
spr = spr.drop("Sem", axis = 1)
spr = spr.drop("Num", axis = 1)

timeVec = list(np.stack((fall, spr), axis = 1))
#timeVec_i,s,g is if course i is offered in timeslot g in season s (fall or spring)

timeCount = len(timeVec[0][0])
print(timeCount)

# FCE Vector
fceVec = list(units["FCE"]) # f_i is the FCE of course i

# Units Vec
unitVec = list(units["Units"]) # unitVec_i is the units of course i

# Group Vec
groups = groups.drop("Num", axis = 1)
groupVec = groups.values.tolist()
groupCount = len(groupVec[0])
"""
groupVec layout (group1 contains course 1 and n)
            Group 1     Group 2     ...     Group n
Course 1    [1              0                    0]
Course 2    [0              0                    1]
...
Course n    [1              0                    0]

"""
```

```python

# PR Vec
PR = PR.drop("Num", axis = 1)
prereqVec = PR.values.tolist()
prereqCount = len(prereqVec[0])
# says the group numbers that are prereqs for each course
"""
prereqVec layout (course 1 has prereq of group 2)
            Group 1     Group 2     ...     Group 3
Course 1    [0              1                     0]
Course 2    [0              0                     0]
...
Course n    [0              1                     0]

"""

# Requirements Vector
req = req.drop("Num", axis = 1)
depthVec = req["Depth Electives"]
depthReqUnits = 45
req = req.drop("Depth Electives", axis = 1)
reqVec = req.values.tolist()
reqCount = len(reqVec[0])
# says the courses needed to fulfill requirments
"""
reqVec layout (to fulfill requirement 1 need course 1 or 2)
            Req 1     Req 2     ...
Course 1    [1         0
Course 2    [1         0
...
Course n    [0         1

"""





#------------------Creating decision variables---------------------
# Define binary variables x_i,j
x=[([0]*semCount) for i in range(courseCount)];
for i in range(courseCount):
    for j in range(semCount):
        x[i][j] = model.addVar(vtype=GRB.BINARY,
                               name="x_({},{})".format(i+1, j+1))

a = model.addVar(vtype=GRB.CONTINUOUS, name = "semester_max")

# Pushing created variables to the model
model.update()

# Each class is taken once
for i in range(courseCount):
    model.addConstr(sum(x[i][j] for j in range(semCount)) <= 1,
                    name='Class {} taken once'.format(i+1))
```

```python

# Don't overload in any semester
for j in range(semCount):
    model.addConstr(sum(unitVec[i]*x[i][j] for i in range(courseCount)) <= 54,
                    name="Don't overload in sem {}".format(j+1))

# Loop through courses
# Loop through groups
# If group is prereq for that course
# loop through group, add constr
for i in range(courseCount):
    for g in range(groupCount):
        if (prereqVec[i][g] == 1):
        # If course i requires group g as a prereq
            for j in range(semCount):
                # Then if course i is taken in semester j, at least one class (c)
                # from group g must be taken in a semester (k) lower than j
                model.addConstr(sum(sum(groupVec[c][g] * x[c][k] for k in range(j))
                                    for c in range(courseCount)) >= x[i][j],
                                    name="Group {} prereq for {} in sem {}".format(g+1,i+1,j+1))

# Each graduation requirement must be satisfied
for r in range(reqCount):
    model.addConstr(sum(sum(reqVec[i][r] * x[i][j] for i in range(courseCount))
                      for j in range(semCount)) >= 1,
                      name="Satisfy requirement {}".format(r+1))

# Need at least 45 units of depth electives
model.addConstr(sum(sum(depthVec[i]*x[i][j]*unitVec[i] for i in range(courseCount))
                    for j in range(semCount)) >= depthReqUnits,
                    name = "Satisfy depth requirements")

# One class per time slot per semester
for g in range(timeCount):
    for j in range(semCount):
        model.addConstr(sum(x[i][j]*timeVec[i][j%2][g]
                            for i in range(courseCount)) <= 1,
                            name="One class in slot {} for sem {}".format(g+1,j+1))

# Courses only taken in seasons they are offered in
for i in range(courseCount):
    model.addConstr(sum(x[i][2*j] for j in range(4)) <=
                    sum(timeVec[i][0][g] for g in range(timeCount)),
                    name="Course {} not taken in fall if spring only".format(i+1))
    model.addConstr(sum(x[i][2*j+1] for j in range(4)) <=
                    sum(timeVec[i][1][g] for g in range(timeCount)),
                    name="Course {} not taken in spring if fall only".format(i+1))


# 360 units in total needed to graduate
model.addConstr(sum(sum(x[i][j] * unitVec[i] for i in range(semCount))
```

```python
for g in range(timeCount):
    for j in range(semCount):
        model.addConstr(sum(x[i][j]*timeVec[i][j%2][g]
                            for i in range(courseCount)) <= 1,
                            name="One class in slot {} for sem {}".format(g+1,j+1))

# Courses only taken in seasons they are offered in
for i in range(courseCount):
    model.addConstr(sum(x[i][2*j] for j in range(4)) <=
                    sum(timeVec[i][0][g] for g in range(timeCount)),
                    name="Course {} not taken in fall if spring only".format(i+1))
    model.addConstr(sum(x[i][2*j+1] for j in range(4)) <=
                    sum(timeVec[i][1][g] for g in range(timeCount)),
                    name="Course {} not taken in spring if fall only".format(i+1))


# 360 units in total needed to graduate
model.addConstr(sum(sum(x[i][j] * unitVec[i] for j in range(semCount))
                    for i in range(courseCount)) >= 360,
                    name = "Total courses needed to graduate")


# Hardcoding to make schedule more reliable
x[44][0] = 1; # concepts in sem 1
x[72][0] = 1 # eureka in sem 1
x[75][3] = 1; # engageInwards in sem 4
x[76][4] = 1; # engageOutwards in sem 5
x[77][6] = 1; # engageForwards in sem 7
x[73][6] = 1; # enagageService in sem 7
x[74][6] = 1; # engageArts in sem 7
x[71][2] = 1 # underGradColloq in sem 3
model.addConstr(sum(x[65][j]+x[67][j]+x[68][j]+x[69][j] for j in range(2,8)) <=0,
                name="No first year writing after first year")


# To minimize the max FCE among semesters (but keep this linear), we create
# variable a which is at least the FCE for each semester and minimize that
model.setObjective(a, GRB.MINIMIZE)
for j in range(semCount):
    model.addConstr(a >= sum(fceVec[i] * x[i][j] for i in range(courseCount)),
                    name = "Max FCE >= sem {} FCE".format(j+1))

# Printing the model in a separate file for easier look
model.write('take3.lp')


#-------------------- Solving the LP -------------------------------------

model.optimize()

#------------------ Outputting the solution -----------------------------

# Prints the non-zero variables and its values in a table format
model.printAttr('X')
```

9 Appendix: Code for Heuristic Algorithm

```
'''
heuristic algorithm
    input: course schedules
    output: feasible schedule of classes

**assumption: 3 technicals, 2 non-technicals

1. read in course schedule
2. make dictionary based on prerequisites (class --> classes prereq for)
3. make dictionary (class --> classes needed as a prereq)
4. 8 times: loop and choose 3 technicals randomly --> add to list of
   visited classes
5. have a list for classes chosen in one semester

'''

import pandas as pd
import numpy as np

# read in data
sched = pd.read_csv("PD_Schedule.csv")
units = pd.read_csv("PD_Units.csv")
prereq = pd.read_csv("PD_PR.csv")
groups = pd.read_csv("PD_Groups.csv")
req = pd.read_csv("PD_Req.csv")

needed_prereq = {}
prereq_for = {}

# for each class, adds in prereqs needed
for i in prereq.index:
    curr_class_i = prereq.loc[i]['Num']
    classes_needed_for_i = [col for col in prereq.columns if prereq.at[i,
   col] == 1]

    needed_prereq[curr_class_i] = classes_needed_for_i

prereq_classes = [('OR_PR', 21292) , ('DTF_PR', 21370), ('MF_PR', 21270),
   ('DE_PR', 21260), ('Py_PR', 15112), ('D_PR', 21228), ('Con_PR', 21128),
    ('Cal_PR', 21268), ('M_Pr', 21241), ('Al_PR', 21373), ('An_PR', 21355)
   , ('P_PR', 21325), ('MI_PR', 73102), ('MA_PR', 73103), ('OM_PR', 70371)
   , ('S_PR', 36226), ('MR_PR', 36401)]

# for each class, adds in what classes require it as a prereq
for c in prereq_classes:
```

```
    curr_prereq = c[0]
    filtered_classes = prereq[prereq[curr_prereq] == 1]

    curr_num = c[1]
    prereq_for[curr_num] = filtered_classes['Num'].tolist()

technicals = [21228, 21236, 21238, 21241, 21254, 21256, 21259, 21260,
    21261, 21266, 21268, 21269, 21270, 21292, 21301, 21325, 21329, 21341,
    21355, 21356, 21366, 21369, 21373, 21374, 21378, 21380, 21420, 21441,
    21484, 15110, 15112, 15122, 15150, 15210, 21128, 21237, 21240, 21242,
    21300, 21370, 21371, 21393, 21469, 21360]

# classes that satisfy each of these requirements
depth_req = (req[req['Depth Electives'] == 1])['Num'].tolist()
d_req = (req[req['D_R '] == 1])['Num'].tolist()
m_req = (req[req['M_R'] == 1])['Num'].tolist()
cal_req = (req[req['Cal_R '] == 1])['Num'].tolist()
de_req = (req[req['DE_R'] == 1])['Num'].tolist()
p_req = (req[req['P_R'] == 1])['Num'].tolist()
e_req = (req[req['E_R '] == 1])['Num'].tolist()
bs_req = (req[req['BS_R'] == 1])['Num'].tolist()
ps_req = (req[req['PS_R'] == 1])['Num'].tolist()

# checks if requirements have been satisfied
depth_req_sat = False
d_req_sat = False
m_req_sat = False
cal_req_sat = False
de_req_sat = False
p_req_sat = False
e_req_sat = False
bs_req_sat = False
ps_req_sat = False
total_req_sat = False

all_available_classes = sched['Num'].tolist()
taken_classes = []

# initializes number of units and semesters, minimum number of depth and
    total units
num_sem = 8
depth_req_units = 45
total_req_units = 360
max_num_tech = 3
max_num_non = 2

curr_depth_units = 0
curr_total_units = 0
```

```
final_schedule = {}

for s in range(1, num_sem + 1):

    curr_num_tech = 0
    curr_num_non = 0

    considered_classes = []

    curr_schedule = [0] * 130

    # initializes semesters with assumptions
    if s == 1:
        curr_sem_classes = [21128, 38101, 76101]
        max_num_classes = 5
        curr_num_tech = 1
        curr_num_non = 2
    elif s == 3:
        curr_sem_classes = [21201]
        max_num_classes = 6
    elif s == 4:
        curr_sem_classes = [38230]
        max_num_classes = 6
    elif s == 5:
        curr_sem_classes = [38330]
        max_num_classes = 6
    elif s == 6:
        curr_sem_classes = [38304]
        max_num_classes = 6
    elif s == 7:
        curr_sem_classes = [38110, 38220, 38430]
        max_num_classes = 8
    else:
        curr_sem_classes = []
        max_num_classes = 5

    curr_num_classes = len(curr_sem_classes)

    while curr_num_classes < max_num_classes:
        # greedily pick classes that is the prereq for most from technical
    + semester
        # then check if prereqs needed are satisfied
        # then check if any other classes from chosen are scheduled during
    that time
        # then check if grad requirements are needed
        prereq_sat = False
        schedule_conflict = True
```

```python
        not_taken = False
        not_considered = False
        offered_in_sem = False

        # gets set of classes that have not been taken yet or attempted to have been taken
        filtered_dict = {key: value for key, value in prereq_for.items() if (key not in considered_classes) and (key not in taken_classes)}
        if not filtered_dict:
            print("The dictionary is empty.")
            break

        # finds class that the most number of classes require it
        max_prereq_class = max(filtered_dict, key=lambda k: len(filtered_dict[k]))

        # checks if all prereqs for chosen class have been taken
        if all(c in taken_classes for c in needed_prereq[str(max_prereq_class)]):
            prereq_sat = True

        # finds what semester it is offered
        index = sched.index[sched['Num'] == str(max_prereq_class)].tolist()[0]
        print(index)
        print(sched)
        print(sched['Sem'])
        curr_offered_sem = sched['Sem'][index]

        # if the current semester matches the semester when the course is offered
        if (((s % 2) == 0) and curr_offered_sem == 'S') or (((s % 2) == 1) and curr_offered_sem == 'F'):
            offered_in_sem = True

        #check scheduling conflict
        curr_offered_sem = sched['Sem'][index]

        max_prereq_class_sched = sched.loc[index, sched.columns.difference(['Num', 'Sem'])]

        # check if there is an avaialble time in current schedule for when the class is offered
        result = [(a + b) % 2 for a, b in zip(curr_schedule, max_prereq_class_sched)]

        chosen_time = []
        chosen_yet = False
```

```python

        for i in range(len(curr_schedule) - 1):
            if not chosen_yet:
                if result[i] == 1 and result[i + 1] == 1:
                    schedule_conflict = False
                    chosen_yet = True
                    chosen_time.append(i)
                    chosen_time.append(i + 1)
                    if i != (len(curr_schedule) - 2):
                        if result[i + 2] == 1:
                            chosen_time.append(i + 2)

        # checks if class has been considered or taken yet
        if max_prereq_class not in considered_classes:
            not_considered = True

        if max_prereq_class in all_available_classes:
            not_taken = True

        # increments either nontechnical or technical count by 1
        if max_prereq_class in technicals:
            curr_num_tech += 1
        else:
            curr_num_non += 1

        # checks if all constraints satisfied
        if prereq_sat and (not schedule_conflict) and not_taken and (curr_num_tech <= max_num_tech) and (curr_num_non <= max_num_non) and not_considered and offered_in_sem:

            print(f"adding {max_prereq_class}")

            curr_sem_classes.append(max_prereq_class)
            all_available_classes.remove(max_prereq_class)
            taken_classes.append(max_prereq_class)
            prereq_for.pop(max_prereq_class)

            index = units.index[units['Num'] == max_prereq_class].tolist()[0]
            curr_total_units += units[index]['Units']

            for t in range(len(chosen_time)):
                curr_schedule[chosen_time[t]] == 1

        considered_classes.append(max_prereq_class)

        # count as grad req
        if curr_total_units == total_req_units:
```

```
            total_req_sat = True

        if max_prereq_class in depth_req:
            index = units.index[units['Num'] == max_prereq_class].tolist()[0]
            curr_depth_units += units[index]['Units']

            if curr_depth_units == depth_req_units:
                depth_req_sat = True

        if max_prereq_class in d_req:
            d_req_sat = True
        if max_prereq_class in m_req:
            m_req_sat = True
        if max_prereq_class in cal_req:
            cal_req_sat = True
        if max_prereq_class in de_req:
            de_req_sat = True
        if max_prereq_class in p_req:
            p_req_sat = True
        if max_prereq_class in e_req:
            e_req_sat = True
        if max_prereq_class in bs_req:
            bs_req_sat = True
        if max_prereq_class in ps_req:
            ps_req_sat = True

    final_schedule[s] = (curr_sem_classes, curr_schedule)
    print(final_schedule)
```