FCE Schedule Optimization

Aarushi Aggarwal, Raegan Brinkman, Emma Hayes, Michelle Zhu

21-393

Group B

Abstract

Every college student faces the struggle of deciding which classes to take on a semesterly basis. When considering a class to take, students must focus on a number of criteria such as unit requirements, prerequisites, weekly workload, and subjective difficulty stemming from faculty course evaluations (FCEs). When it specifically comes to students in the Mellon College of Science, there are a myriad of classes with specific requirements, such as which semester a student must take them, which can lead to a mishap in scheduling requirements if not followed correctly. Recognizing the importance of these criteria, we researched a way to optimize undergraduate degree planning to minimize the amount of unnecessary stress and ensure a smoother progression through the undergraduate curriculum. In this study, we present a novel approach to schedule optimization for undergraduate students in the Mellon College of Science, specifically those majoring in Mathematical Sciences with a concentration in Operations Research and Statistics, through the utilization of linear programming. Our objective was to create a linear program that takes into account the same factors that a student would, including unit requirements, prerequisites, and the subjective difficulty of courses based on FCEs. Each class is assigned a specific number of units, representing the weekly workload required, and FCEs provide insights into the perceived difficulty by students. Through the formulation of constraints and objective functions, our linear program aims to identify an optimal course selection for students aiming to complete their undergraduate degree in four years, meaning eight semesters. Additionally, we used a Greedy algorithm to figure out which class to take in which semester, and finally a scheduling heuristic in SIO to make sure that our schedule was feasible in the end. By considering the constraints imposed by prerequisites and semester-specific requirements, our model provides a comprehensive solution that minimizes conflicts and

maximizes the efficiency of degree completion. Throughout this process we found it necessary to make changes along the way. For example, we had to replace classes that we thought would be offered but weren’t with classes that were offered. At the conclusion of our research, we reached an optimal and feasible solution that returned us a schedule with 362 unit hours over 8 semesters, with an FCE count of only 299.

Introduction

Students at CMU have a lot of courses they need to take before they can graduate. Some are mandatory classes while others are elective courses that they can choose from a predetermined list. But a lot of these courses are not easy. CMU defines each class with a certain number of units which indicates how many hours a student is expected to spend on that class each week, including lectures and recitation times. But of course, this is not a perfect estimate. This is why CMU sends out a faculty course evaluation survey at the end of each semester for students to fill out to determine the true number of hours they spent per week on each course. This is what is known as the FCE hours.

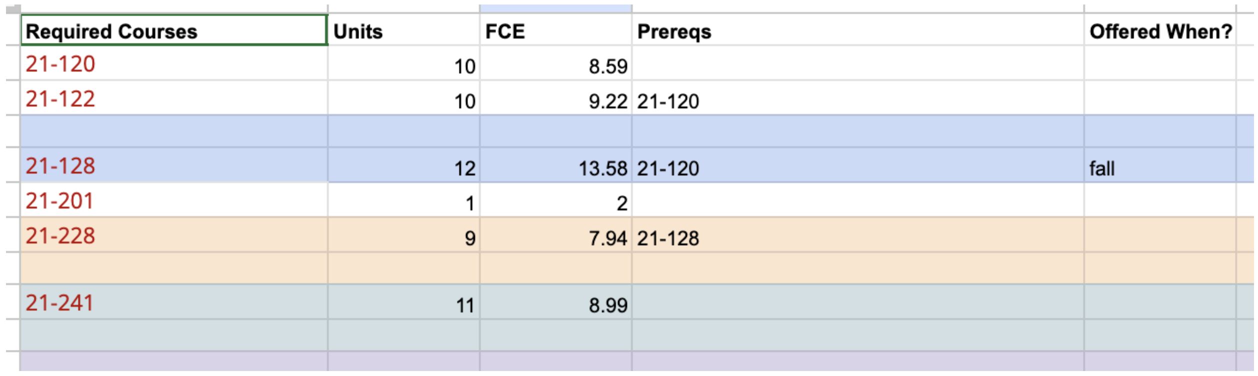
For our project, we are going to focus on finding a schedule that minimizes the total FCE hours for a student who is a math major with a Operations Research and Statistics concentration. This concentration requires the student to take eleven mathematical science courses, five statistics courses, and five economics, business, and computer science courses. Also as a student in the Mellon College of Science, there are other core requirements that they need to take. These are: first year writing, freshman seminar, junior seminar, five ENGAGE courses, five depth electives, one life science, one physical science, one mathematics, statistics, and computer science, one STEM, one cultural and global understanding, and four humanities and social sciences. Each of these groups have a predetermined list of courses the student can choose from. These courses also can not double count for more than one of these categories. Finally, the student needs a total of three hundred sixty units to graduate. To complete our project, we decided to code in Microsoft Excel and Google Colab, utilizing Python and the GurobiPy software.

Assumptions

To make our computation simpler, we made several assumptions. When defining what the Faculty Course Evaluation (FCE) number was for each course, we used the CMU Courses website, cmucourses.com. The CMU Courses website gets the FCE value as the average of the previous two semesters and is dependent on the response rate on the FCE surveys at the end of each semester. If there was no FCE data for a course, we assumed the FCE was the number of units of the course. This is because the number of units a course is an estimate of the amount of time that course would take per week. Another assumption we made was for some courses that had substitute classes, we chose the class that a "typical" math major would take. For example, we chose to keep 21-128 over 21-127 since 21-128 is the math major version of Mathematical Concepts and Proofs. We also required that the student take 21-325 instead of 36-225 for Probability since that is the new requirement for incoming math majors. We also removed the math honors version of courses. For example, for introductory linear algebra we chose to take 21-241 over 21-242 which is the math honors version. We also removed elective courses that were only for students of other majors. Since we are choosing a schedule for a student who is a math major only, they cannot take 10-315 which is machine learning for computer science majors only. We removed graduate courses since we assumed that the student is not considering doing the 5+1 program. We excluded language classes since there are so many of them with different units to make the code simpler. Additionally, to make figuring out the timing of courses easier, we assumed that the student would take 21-241 with Professor Howell only since that is what most math majors did our freshman year.

Data

To solve our LP, we need some data from CMU. In particular, we need the list of required courses and the list of courses that count for each technical and non-technical elective requirement. For each class, we also collected data on their number of units, FCE hours, prerequisites, and when they are offered. We imported all the data into an Excel spreadsheet where each of these variables were columns. If a class was offered in both semesters, we left the column labeled “Offered When?” blank, only indicating if they were fall-only or spring-only classes. We got this data through the CMU website and CMU Courses website. In the Excel spreadsheet, we color coded classes that fulfill the same requirement the same color. We then omitted the classes that a “typical” math major would not take, in order to fulfill our assumptions. Our spreadsheet ended up looking like this:



Required Courses	Units	FCE	Prereqs	Offered When?
21-120	10	8.59		
21-122	10	9.22	21-120	
21-128	12	13.58	21-120	fall
21-201	1	2		
21-228	9	7.94	21-128	
21-241	11	8.99		

We also had different tabs for different subsections of requirements. Those tabs consisted of Required Courses (Academic), In Depth Electives, General Education Requirements, and Non-Technical Breadth Requirements. So all of our data was compiled and organized in an understandable way in one Google Sheet file.

To get the schedule times for lecture and recitation for each course, we looked at the schedules for Fall 2023 and Spring 2024 as these were the only semesters available to us.

Method

To solve for a schedule for an Operations Research and Statistics math major, we created an integer linear program. Our objective function was to minimize the overall total FCE hours the student would have for all eight semesters at CMU. We had variables X_{ij} which equal 1 if we took class i in semester j and 0 otherwise. We also included multiple constraints for our IP based on requirements on when we need to take specific classes, or how many units of a category we need to take. We made sure that we didn't retake a course and that the student satisfied the full time student unit status each semester without overloading. Another constraint was that we made sure the prerequisites were satisfied when taking a course. If there were substitute classes for the same requirement, we made sure that the student only took one. Finally, we made sure the student took all the required classes and enough elective classes to graduate after four years.

Since this integer LP is hard to solve, we decided to split this up into three steps. For the first step, we created a new LP with fewer constraints to just find the list of all the classes the student needs to take in their four years. The constraints would be that the student must take all the required classes and the necessary categories for technical and non-technical electives. In the second step, we have a greedy algorithm to assign each class to a semester such that it satisfies another set of constraints. These constraints include which semester these courses must be taken by, make sure prerequisites are satisfied, and that each semester is between 36 and 54 units. The last step is to make sure that this schedule is feasible by checking whether the course times overlap through a heuristic. We order the courses per semester and attempt to create these schedules on SIO using the Fall 2023 and Spring 2024 schedules. If we are unable to make a schedule, we will swap classes around until we get a feasible solution.

New Integer Linear Programming Implementation

Solve this new IP to get the list of courses the student must take within their four years.

1. Take all the required classes
2. Take at least 45 units of in-depth elective courses
3. Take at least 9 units of life science courses
4. Take at least 9 units of physical science courses
5. Take at least 1 class from Math/Statistics/Computer Science courses
6. Take at least 1 class from STEM courses
7. Make sure the courses chosen for Math/Statistics/Computer Science and STEM are different
8. Take at least 1 class from Science and Society category
9. Take at least 9 units of cultural/global understanding courses
10. Take at least 36 units of humanities/arts courses (done by us afterwards)
11. Take all ENGAGE courses
12. Take C@CM

Greedy Algorithm Implementation

Schedule the courses given by our LP according to the following constraints:

1. Between 36 and 54 units every semester
2. Only take a class if you satisfy the prerequisites
3. Do not retake courses
4. Fulfill the one course in physical sciences, life sciences, stem, math/cs/stat (three in freshman year, i.e. 3 in semesters 0 and 1)

5. First year writing must be taken in the first year (semester 0 or 1)
6. Even semesters can only take courses from the fall set of courses, odd semesters from the spring set
7. Eureka! Freshman fall (semester 0)
8. Take 38-230 - Looking Inward sophomore spring (semester 3)
9. Take 38-330 - Looking Outward junior fall (semester 4)
10. Take Propel/Junior seminar course junior spring (semester 5)
11. Take 38-430 - Looking Forward senior fall (semester 6)
12. All Engage courses must be completed in or before senior fall (semester 7)
    a. 38-110 and 38-220
13. Each course is taken

Time Heuristic Implementation

Check that our schedule is feasible

1) $E_i < S_j$ where class i is before class j and E_i is the end time of class i and S_j is the start time of class j (Done through our inputting into SIO)
2) If not feasible, we moved classes around to other semesters where the class would fit and it still satisfied the prerequisites

Code

1. Course decision IP

It is important to note that the specific indices reflect the ordering of classes on our courses csv file.

```
from gurobipy import GRB
import gurobipy as gp
import pandas as pd
import numpy as np
classes = pd.read_excel("or2project.xlsx", "Sheet1")

model = gp.Model()

x = pd.Series(model.addVars(len(classes), vtype=GRB.BINARY))

model.setObjective(classes['FCE'].dot(x), GRB.MINIMIZE)


#no repeating classes and total units requirement
cons1 = model.addConstrs(x[j]<=1 for j in range(len(classes)))
cons2 = model.addConstr(gp.quicksum(classes['Credits'][j]*x[j] for j in
range(194)) >= 324)
#required math/stat/business classes
cons3 = model.addConstrs(x[j]==1 for j in range(20))
cons4 = model.addConstr(x[20]+x[21]>=1)
#non technical breadth requirements
cons5 = model.addConstr(x[90]==1)
cons6 = model.addConstr(x[96]==1)
cons7 = model.addConstr(gp.quicksum(classes['Credits'][j]*x[j] for j in
range(91,96)) == 9)
cons8 = model.addConstr(gp.quicksum(x[j] for j in range(97,122))==1)
cons9 = model.addConstrs(x[j]==1 for j in range(122, 127))
cons10= model.addConstr(gp.quicksum(classes['Credits'][j]*x[j] for j in
range(127,194)) >= 9)
#in depth electives
cons11= model.addConstr(gp.quicksum(classes['Credits'][j]*x[j] for j in
range(22,47)) >= 45)
#general education requirements
```

```
cons12= model.addConstr(gp.quicksum(classes['Credits'][j]*x[j] for j in 
range(47,62)) >= 9)
cons13= model.addConstr(gp.quicksum(classes['Credits'][j]*x[j] for j in 
range(62,81)) >= 9)
cons14= model.addConstr(gp.quicksum(classes['Credits'][j]*x[j] for j in 
range(81,90)) >= 9)
cons15= model.addConstr(gp.quicksum(classes['Credits'][j]*x[j] for j in 
range(47,81)) >= 36)
#prereqs
model.addConstr(x[22]<=x[2])
model.addConstr(x[22]<=x[23])
model.addConstr(x[22]<=x[11])
model.addConstr(x[23]<=x[86])
model.addConstr(x[24]<=x[23])
model.addConstr(x[24]<=x[2])
model.addConstr(x[25]<=x[23])
model.addConstr(x[25]<=x[24])
model.addConstr(x[26]<=x[0])
model.addConstr(x[27]<=x[1])
model.addConstr(x[27]<=x[2])
model.addConstr(x[28]<=x[2])
model.addConstr(x[29]<=x[35])
model.addConstr(x[31]<=x[30])
model.addConstr(x[32]<=x[31])
model.addConstr(x[34]<=x[26])
model.addConstr(x[36]<=x[26])
model.addConstr(x[38]<=x[34])
model.addConstr(x[46]<=x[44])
model.addConstr(x[47]<=x[50]+x[51])
model.addConstr(x[48]<=x[86])
model.addConstr(x[58]<=x[50]+x[51])
model.addConstr(x[60]<=x[50]+x[51])
model.addConstr(x[63]<=x[62]+x[64])
model.addConstr(x[66]<=x[63])
model.addConstr(x[66]<=x[72]+x[74])
model.addConstr(x[67]<=x[62]+x[64])
model.addConstr(x[68]<=x[63]+x[64])
model.addConstr(x[69]<=x[63]+x[64])
model.addConstr(x[70]<=x[62]+x[64])
model.addConstr(x[70]<=x[72]+x[74])
model.addConstr(x[70]<=x[62]+x[64])
model.addConstr(x[71]<=x[62]+x[64])
model.addConstr(x[73]<=x[72]+x[74]+x[76])
model.addConstr(x[70]<=x[62]+x[64])
model.addConstr(x[75]<=x[72]+x[74]+x[76])
model.addConstr(x[77]<=x[76])
model.addConstr(x[78]<=x[73]+x[75]+x[77])
model.addConstr(x[79]<=x[72]+x[74]+x[76])
model.addConstr(x[80]<=x[73]+x[75]+x[77])
model.addConstr(x[87]<=x[86])
model.addConstr(x[88]<=x[86])
model.addConstr(x[89]<=x[87])
model.addConstr(x[98]<=x[62]+x[64])
model.addConstr(x[98]<=x[72]+x[74]+x[76])
model.addConstr(x[101]==0)
model.addConstr(x[102]<=x[62]+x[64])
model.addConstr(x[115]<=x[73]+x[75]+x[77])


model.optimize()
```

2. Greedy Algorithm

It is important to note that the specific indices reflect the ordering of classes on our chosen courses csv file.

```
from gurobipy import GRB
import gurobipy as gp
import pandas as pd
import numpy as np
classes = pd.read_excel("Courses.xlsx", "Chosen Courses")
model2 = gp.Model()
s=8
y = pd.Series(model2.addVars(len(classes),s, vtype=GRB.BINARY))

#only take classes once and credit requirements
model2.addConstrs( (gp.quicksum(y[i][j]*classes['Credits'][i] for i in
range(len(classes))) <= 54)for j in range(0,8))
```

```
model2.addConstrs( (36 <= gp.quicksum(y[i][j]*classes['Credits'][i] for i
in range(len(classes))))for j in range(0,8))
model2.addConstrs(gp.quicksum(y[i][j] for j in range(0,8)) == 1 for i in
range(len(classes)))

#prereqs
model2.addConstrs(y[1][j] <= (gp.quicksum(y[0][k] for k in range(j))) for
j in range(8))

model2.addConstrs(y[2][j] <= (gp.quicksum(y[0][k] for k in range(j))) for
j in range(8))

model2.addConstrs(y[4][j] <= (gp.quicksum(y[2][k] for k in range(j))) for
j in range(8))

model2.addConstrs(y[6][j] <= (gp.quicksum(y[1][k] for k in range(j))) for
j in range(8))
model2.addConstrs(y[6][j] <= (gp.quicksum(y[5][k] for k in range(j))) for
j in range(8))

model2.addConstrs(y[7][j] <= (gp.quicksum(y[1][k] for k in range(j))) for
j in range(8))

model2.addConstrs(y[8][j] <= (gp.quicksum(y[1][k] for k in range(j))) for
j in range(8))
model2.addConstrs(y[8][j] <= (gp.quicksum(y[5][k] for k in range(j))) for
j in range(8))
model2.addConstrs(y[8][j] <= (gp.quicksum(y[4][k] for k in range(j))) for
j in range(8))

model2.addConstrs(y[9][j] <= (gp.quicksum(y[16][k] for k in range(j))) for
j in range(8))
model2.addConstrs(y[9][j] <= (gp.quicksum(y[6][k] for k in range(j))) for
j in range(8))
model2.addConstrs(y[9][j] <= (gp.quicksum(y[5][k] for k in range(j))) for
j in range(8))
model2.addConstrs(y[9][j] <= (gp.quicksum(y[7][k] for k in range(j))) for
j in range(8))
```

```
model2.addConstrs(y[10][j] <= (gp.quicksum(y[4][k] for k in range(j))) for
j in range(8))
model2.addConstrs(y[10][j] <= (gp.quicksum(y[8][k] for k in range(j))) for
j in range(8))

model2.addConstrs(y[11][j] <= (gp.quicksum(y[6][k] for k in range(j))) for
j in range(8))

model2.addConstrs(y[12][j] <= (gp.quicksum(y[11][k] for k in range(j)))
for j in range(8))

model2.addConstrs(y[13][j] <= (gp.quicksum(y[12][k] for k in range(j)))
for j in range(8))
model2.addConstrs(y[13][j] <= (gp.quicksum(y[5][k] for k in range(j))) for
j in range(8))

model2.addConstrs(y[14][j] <= (gp.quicksum(y[13][k] for k in range(j)))
for j in range(8))

model2.addConstrs(y[15][j] <= (gp.quicksum(y[11][k] for k in range(j)))
for j in range(8))

model2.addConstrs(y[19][j] <= (gp.quicksum(y[18][k] for k in range(j)))
for j in range(8))

model2.addConstrs(y[20][j] <= (gp.quicksum(y[6][k] for k in range(j))) for
j in range(8))
model2.addConstrs(y[20][j] <= (gp.quicksum(y[18][k] for k in range(j)))
for j in range(8))

model2.addConstrs(y[21][j] <= (gp.quicksum(y[2][k] for k in range(j))) for
j in range(8))

model2.addConstrs(y[23][j] <= (gp.quicksum(y[13][k] for k in range(j)))
for j in range(8))

model2.addConstrs(y[24][j] <= (gp.quicksum(y[29][k] for k in range(j)))
for j in range(8))
```

```
model2.addConstrs(y[25][j] <= (gp.quicksum(y[24][k] for k in range(j)))
for j in range(8))

model2.addConstrs(y[28][j] <= (gp.quicksum(y[0][k] for k in range(j))) for
j in range(8))

model2.addConstrs(y[29][j] <= (gp.quicksum(y[0][k] for k in range(j))) for
j in range(8))

#specific semesters
model2.addConstr(gp.quicksum(y[34][j] for j in range(0,7)) == 1 ) #38-110
by last sem
model2.addConstr(gp.quicksum(y[35][j] for j in range(0,7)) == 1 ) #38-220
by last sem
model2.addConstr(gp.quicksum(y[3][j] for j in range(2,4)) == 1)
model2.addConstr(gp.quicksum(y[31][j] for j in range(0,2)) == 1)
model2.addConstr(y[30][0] == 1) #38-101 (EUREKA!) first sem
model2.addConstr(y[32][0] == 1) #99-101 first sem
model2.addConstr(y[36][3] == 1) #38-230 spring 2nd year
model2.addConstr(y[37][4] == 1) #38-330 fall third year
model2.addConstr(y[33][5] == 1) #propel class (19-429)
model2.addConstr(y[38][6] == 1) #38-430 fall 4th year
model2.addConstr(gp.quicksum(y[16][j] + y[26][j] + y[27][j] + y[28][j] +
y[29][j] for j in range(0, 2)) >= 3)

#spring vs fall
model2.addConstr(y[2][1]+y[2][3]+y[2][5]+y[2][7]==0)

model2.addConstr(y[8][0]+y[8][2]+y[8][4]+y[8][6]==0)

model2.addConstr(y[10][1]+y[10][3]+y[10][5]+y[10][7]==0)

model2.addConstr(y[12][0]+y[12][2]+y[12][4]+y[12][6]==0)

model2.addConstr(y[13][1]+y[13][3]+y[13][5]+y[13][7]==0)

model2.addConstr(y[14][0]+y[14][2]+y[14][4]+y[14][6]==0)

model2.addConstr(y[15][0]+y[15][2]+y[15][4]+y[15][6]==0)
```

```
model2.addConstr(y[23][0]+y[23][2]+y[23][4]+y[23][6]==0)


model2.addConstr(y[28][1]+y[28][3]+y[28][5]+y[28][7]==0)



model2.optimize()
```

Results

We solved the first IP, finding the list of required classes that students must take in their four years to minimize the number of FCE hours. Within this IP, we made sure that our program returned all the required classes and the necessary courses to fulfill technical and non-technical requirements. Our resulting list of classes is as follows:

Required Classes	**Technical Electives**	**Non-Technical Electives**
21-120	15-110	42-101
21-122	70-122	33-121
21-128	73-102	33-151
21-201	73-103	36-220
21-228	73-230	38-101
21-241	21-321	76-101
21-268	21-366	99-101
21-260	36-461	19-429
21-292	70-371	38-110
21-369	70-471	38-220
21-393		38-230
21-325		38-330
36-226		38-430
36-401		79-263
36-402		
36-410		

This table gives us a total unit count of 325 hours, with an FCE count of 286.86 hours. It is important to note that this table is without any of the non-technical elective classes, which bring us up to the required 360 units needed to graduate.

Our resulting schedule to minimize FCE from our Greedy algorithm was:

Semester 1: 21-241 15-110 42-101 38-101 99-101 42 units	Semester 2: 21-120 73-102 33-121 76-101 38-220 42 units	Semester 3: 21-122 21-128 70-122 33-151 43 units	Semester 4: 21-201 21-228 21-268 21-260 38-230 79-263 39 units
Semester 5: 21-325 21-321 21-366 36-220 38-330 37 units	Semester 6: 21-292 36-226 36-410 70-371 19-429 45 units	Semester 7: 21-393 36-401 73-103 73-230 38-110 38-430 38 units	Semester 8: 21-369 36-402 36-463 70-471 39 units

Next, we needed to include the four non-technical electives. Since there are hundreds of courses that satisfy this requirement, we just chose four non-technical electives that had low FCEs or ones that we took in previous semesters. The four non-technical electives we added are 82-137, 57-173, 57-374, and 85-211, resulting in 36 units as required.

Our resulting schedule after applying the time heuristic and adding in the non-technical electives was:

Sem 1: 54 u 21-241 15-110 42-101 38-101 99-101 21-295 57-209 FCE: 39 hr	Sem 2: 51 u 21-120 73-102 33-121 76-101 38-220 57-374 FCE: 37 hr	Sem 3: 43 u 21-122 21-128 70-122 33-151 FCE: 38 hr	Sem 4: 40 u 21-201 21-228 21-268 21-260 38-230 85-211 FCE: 36 hr
Sem 5: 46 u 21-325 21-321 21-366 36-220 38-330 57-173 FCE: 38 hr	Sem 6: 42 u 21-292 36-226 36-410 70-371 38-304 FCE: 38 hr	Sem 7: 38 u 21-393 36-401 73-103 38-110 38-430 82-137 FCE: 30 hr	Sem 8: 48 u 21-369 36-402 36-462 70-471 73-230 FCE: 43 hr

When applying the time heuristic, we ran into some time conflicts and some classes that did not exist or did not have a scheduled time in Fall 2023 or Spring 2024. For example, 21-366 and 21-321 did not have a scheduled time in the schedule data we received. But on the CMU website, it says that these courses are offered regularly. Therefore, we assumed that 21-366 was at 10am in junior fall since that is when it was offered before and it is supposed to be offered every fall. We also assumed that 21-321 had a scheduled time that would fit into junior fall since it says on the CMU website that it is offered every fall. Courses 79-263, 19-429, and 36-463 on the other hand, are stated to be offered intermittently. Therefore, we chose other courses to replace those. We looked for the next lowest FCE course that satisfied the same requirements. Thus, we replaced 79-263 with 57-209 and moved it from sophomore spring to freshman fall so the time heuristic was met. We replaced 19-429 with 38-304 in junior spring. This course satisfies the junior seminar requirement. One note is that 19-429 was 9 units and 38-304 is now 6 units. So we need to add 3 more units to make sure we meet the 360 units requirement for graduation. Lastly, we replaced 36-463 with 36-462 in senior spring. Next, we added our non-technical requirements. We placed 82-137 in senior fall, 57-173 in junior fall, 57-374 in freshman spring, and 85-211 in sophomore spring. We moved 73-230 to senior spring since it had a time conflict in senior fall. Then to account for the loss of 3 units from replacing 19-429 with 38-304, we added Putnam Seminar (21-295) to freshman fall since every math major is registered for that course their freshman fall semester. Now we are left with a feasible schedule with a minimum FCE.

Discussion

In our project, we made some assumptions to simplify our programs. In the future, we could include all possible courses that count and not only the ones that a typical math major would take. This will allow the program to pick the best course between all courses that satisfy the same requirement. We also assumed no double-counting, but if we were able to and included courses that, for example, a computer science student would take that count for both computer science and math majors, we could try to add a minor or a double major. Right now, we also split up our program into three steps. The last step we did by hand to check the time heuristic. In the future, we could try to code this up. We also chose by hand the non-technical electives since there are hundreds of courses that satisfy this requirement. Therefore, in the future, we could add these courses into our program and have it choose the best four non-technical electives to take. Since some of these semesters had below 54 units and low FCEs, we could also see if the student can complete all the requirements in less than four years. We also had some limitations for schedule data, having just Fall 2023 and Spring 2024 available to us. If we had more data, we could include courses that are not offered every semester or year as part of our final schedule. We could also expand our project by considering preferences to specific professors after a student takes one course with them and wants to take another course with that professor, or they've heard a lot about the professor.