Teachers As Scholars: Course Preference Registration Matching

Authors: Palak Prashant, Saahil Bhatia, Bradley Chang

Team Members: Shubham Shrivastava, Aneesh Khilnani, Arth Maindarkar, Jonathan Buchanan

College of Science, Purdue University

Dr. Steven J Miller (Williams College)

October 8th, 2021

Abstract

This paper reviews the advances of the linear programming-based approach to schedule classes based on teacher or student preferences. Given available courses, their capacities, and the ratings of the course sections, we show how to use a simple linear programming problem to determine an optimal class schedule. This paper delves into optimization methods by applying them to matching problems and explores the entire process of quantitative modeling, including problem definition, data collection, model formulation, model solution, implementation, and testing.

Introduction

Class scheduling has been a challenging problem faced by academic institutions across the world, especially in the context of utilizing existing resources and tools efficiently and economically. It refers to fitting or allotting all courses to meet the preferences of students or faculty while also incorporating the optimization of the available facilities in terms of operating cost. Several techniques can be employed to develop a solution for the scheduling and matching problem. These methods range from mathematical programming, analytical methods, graph coloring approaches, and artificial intelligence techniques to heuristics and metaheuristics.

We address the scheduling problem for Teachers As Scholars, a program in which K-12 teachers participate in small, two-day seminars led by university faculty in different fields of study. In this problem, we reconfigure how four-hundred teachers sign up for thirty-six classes through the program's website and how a matching tool can be developed to maximize teacher satisfaction. Therefore, this paper discusses the combination of two sub-problems: teacher assignment and course scheduling. It is important to note that this matching tool can also be used in a comparable situation wherein students opt for courses.

Problem Description

We had already undertaken other mathematical approaches to devise an algorithm that processed teacher enrollment based on three ranked course choices from a given course list. These included the Hungarian algorithm, the Gale Shapley algorithm, and the Evolutionary algorithm. However, they could not effectively determine the existence of an optimal solution, especially for large-scale timetabling problems. Linear programming, on the other hand, could not only tackle large-scale timetabling challenges but could also help maximize teacher happiness by allocating more credit to solutions that had more classes running. Consequently, this approach was applied to the project.

Although each academic institution may encompass a unique set of conditions for its scheduling process, our model can accommodate the most common requirements of every institution. In this project, we account for the following specifications of Teachers As Scholars:

1. The number of teachers that can opt for a seminar must be limited.
2. The website must have the ability to:
    1. Enter seminar codes and seminar names
    2. Enter the names of the teachers with their choices (this step should be ideally completed with the help of an Excel or .csv file since manually entering 400 teachers is a tedious process.
    3. Change the minimum or maximum number of teachers on a per-seminar basis.
    4. Display those teachers that have not been assigned to any seminar.
    5. Decide not to have a class so the algorithm won't assign teachers to it.
    6. Display the results of the matching problem, i.e., a list containing the names of teachers and the seminars they have been assigned to. This step must also be completed using an Excel or .csv file.
    7. Display the number/percentages of teachers assigned to their first, second, and third choices. The algorithm must be optimized such that the seminars will fill up to their maximum capacities.

Mathematical Approach

Under the linear programming approach, the fundamental strategy behind solving the matching problem was the following:

$$\begin{array}{ll} \text{Find a vector} & \mathbf{x} \\ \text{that maximizes} & \mathbf{c}^T\mathbf{x} \\ \text{subject to} & A\mathbf{x} \leq \mathbf{b} \\ \text{and} & \mathbf{x} \geq \mathbf{0}. \end{array}$$

This approach aims to maximize teacher satisfaction and ensure that they are placed in seminars that will run. The variables mentioned in this approach represent the following:

1. **x**: This vector of Booleans (integers 0 and 1) describes the schedule solution. It labels whether or not a student is assigned to a seminar. For instance, if there are ten seminars, the first ten values describe if the first teacher is allotted to any of those seminars. The next ten values of this vector describe if the next teacher is assigned to those seminars, and so on. The end of this vector consists of values specifying whether or not the seminars will run. This vector requires ten bits per teacher in the tool. Therefore, for large datasets, there is a possibility that the tool implementation would become too large. To solve this problem, incorporating sparse matrices may be a wise idea. In our case, this challenge did not arise as 400 teachers with 36 seminars use 0.0018 megabytes.

2. **cTx:** ‘c’ is the happiness function ranging from integers 0 through 5, a vector similar to x. It describes the value preference of all the seminars for each teacher. Integers 1 through 5 mark how preferable the class is, where 5 represents most preferred, 1 represents least preferred, and 0 denotes that the teacher did not opt for the seminar. Therefore, if there are ten seminars, the first ten values will describe how the first teacher ranks those seminars, and the next ten values will describe how the next teacher ranks those seminars, and so on. We also add an entry for each class at the end of the c vector, all these entries represent: (number of teachers) *5/ (number of seminars).
3. **Ax:** This vector ensures that every seminar-instructor is allotted to only one seminar. We can say that A is somewhat like an identity matrix. In our approach, we attribute Ax to ‘1,’ wherein 1 is a vector describing the number of classes each teacher *should be* assigned to. T(x)=Ax is a transformation that turns x into a vector, specifying the number of seminars each teacher *is* assigned to. For instance, for two courses and three students, the matrix would look like [110000, 001100, 000011].

Each matching problem has a variety of constraints, such as the instructors' constraints, class capacity constraints, or time constraints, which may involve extra operational expenses. Therefore, it would be crucial to find tools or techniques that would handle such situations. In our case, we tackle class capacity constraints, consisting of the minimum and the maximum number of teachers that the seminar can accommodate.

For a teacher ‘t’:

1. ‘S’ is the total number of classes in the system
2. x [t, s] is a Boolean variable equal to 1 if the teacher ‘t’ is taking seminar ‘s’
3. $\sum_{s=1}^{S} x[t, s]$ is the number of seminars that teacher ‘t’ is taking

$$\sum_{s=1}^{S} x\,[t, s] \leq 1$$

For each seminar ‘s’:

1. ‘T’ is the total number of teachers in the system
2. x [t, s] is a Boolean variable that is 1 if teacher ‘t’ is taking seminar ‘s’
3. w[s] is a Boolean variable that is 1 if the seminar will run (assuming that the seminar meets all the requirements)
4. $\sum_{t=1}^{T} x[t, s]$ is the number of teachers taking the seminar ‘s’
5. m[s] is the minimum class size for seminar ‘s’ to run
6. M[s]is the maximum class size for seminar ‘s’
7. S is the total number of seminars in the system
8. r [t, s] is the ranking/preference value that teacher ‘t’ has for seminar ‘s’ (currently either 5,3,1,0)

9. R is the objective value of ensuring that a course meets the requirements, which currently is equal to 5T/S

Using these variables, we prepare the minimum and maximum constraints as the following:

Minimum constraint C1: $\sum_{t=1}^{T} x\ [t,s] - M[s] * w[s] \leq 0$

Maximum constraint C2: $\sum_{t=1}^{T} x\ [t,s] - m[s] * w[s] \geq 0$

To maximize teacher happiness using the given constraints, we will use the equation:

$$\sum_{s=1}^{S}\left(\sum_{t=1}^{T} r[t,s] * x[t,s] + R * w[s]\right)$$

The following tables depicts different scenarios in this matching approach. The underlined values represent those scenarios in which both the constraints have been satisfied:

Scenario 1 - The number of teachers lies between the minimum and maximum capacities of the seminar:

Minimum	**Number of Teachers**	Maximum	Seminar will run	C1 satisfied?	C2 satisfied?	C1 LHS	C2 LHS
5	10	20	1	Yes	Yes	-10	5
5	10	20	0	No	Yes	10	10

Scenario 2 - No teacher has been assigned to a seminar:

Minimum	**Number of Teachers**	Maximum	Class will run	C1 satisfied?	C2 satisfied?	C1 LHS	C2 LHS
5	0	20	1	Yes	No	-20	-5
5	0	20	0	Yes	Yes	0	0

Scenario 3 - The number of teachers falls below the minimum capacity of the seminar:

Minimum	**Number of Teachers**	Maximum	Class will run	C1 satisfied?	C2 satisfied?	C1 LHS	C2 LHS
5	4	20	1	Yes	No	-16	-1
5	4	20	0	No	Yes	4	4

Scenario 4 - The number of teachers falls above the maximum capacity of the seminar:

Minimum	**Sum of Students**	Maximum	Class will run	C1 satisfied?	C2 satisfied?	C1 LHS	C2 LHS
5	21	20	1	No	Yes	1	16
5	21	20	0	No	Yes	21	21

The Matcher: Proposed Tool for Teachers As Scholars

The matcher takes two files for input describing students and courses. For solving the problem, input files require the following input data:

1. Teacher's first choices
2. Teacher's second choices
3. Teacher's third choices
4. Minimum number of teachers in a seminar
5. Maximum number of teachers in a seminar

The matcher uses this data to arrive at a solution. It goes through the following steps to do so:

1. Create and implement the model:
    a. Initialize a model and a solver
    b. Initialize variables
    c. Create constraints from the variables and input .csv or Excel data
    d. Create an objective equation from the variables and input .csv or Excel data
    e. Add the constraints and the objective function to the model
    f. Use the solver to solve the linear programming model
2. Output the required statistics, seminar data with corresponding sizes, teacher data with assignment information, and data on unassigned teachers. To output the formatted results, we needed the variables listed under point 1 along with the following:
    a. Teacher first name
    b. Teacher last name
    c. Teacher ID
    d. Seminar name
    e. Seminar ID

The matcher.py module allows us to instantiate a matcher, tell it which excel files to use, which sheet number in the excel file to use, and which columns to read data from before solving and outputing results.

Currently, the column names should be passed as python dictionaries. The matcher should be called in this order:

1. matcher = HardConstraintMatcher(students_fileLocation, students_SheetName, students_Columns, courses_FileLocation, courses_SheetName, courses_Columns)
2. matcher.solve()
3. matcher.outputResults()

Example usage of the matcher is as follows:

```
from matcher import HardConstraintMatcher

matcher = HardConstraintMatcher(
    students_FileLocation = 'Students.xlsx',
    students_SheetName = 0,
    students_Columns = {
        "First_Name": "First Name",
        "Last_Name": "Last Name",
        "P1": "Preference 1",
        "P2": "Preference 2",
        "P3": "Preference 3",
    },
    courses_FileLocation = 'Courses.xlsx',
    courses_SheetName = 0,
    courses_Columns = {
        "Name": "Course Name",
        "Min": "Minimum Size",
        "Max": "Maximum Size"
    }
)
matcher.solve() #this is the step that takes a long time
matcher.outputResults()
```

We can specify the matcher with a JSON file as follows:

```
from matcher import HardConstraintMatcher
from json import load

with open('config.json') as config_file:
    config = load(config_file)

    matcher = HardConstraintMatcher(
        students_FileLocation = config["students_FileLocation"],
        students_SheetName = config["students_SheetName"],
        students_Columns = config["students_Columns"],
        courses_FileLocation = config["courses_FileLocation"],
        courses_SheetName = config["courses_SheetName"],
        courses_Columns = config["courses_Columns"]
    )
matcher.solve() #this is the step that takes a long time
matcher.outputResults()
```

config.json:

```
{
   "students_FileLocation":"data\\MOCK_Students.xlsx",
   "students_SheetName":0,
   "students_Columns":{
      "First_Name":"first_name",
      "Last_Name":"last_name",
      "P1":"Preference_1",
      "P2":"Preference_2",
      "P3":"Preference_3"
   },
   "courses_FileLocation":"data\\MOCK_Courses.xlsx",
   "courses_SheetName":1,
   "courses_Columns":{
      "Name":"Course Name",
      "Min":"Test Min",
      "Max":"Test Max"
   }
}
```

This matcher requires the following while being implemented on Python:

1. Pulp
2. Pandas
3. Google OR-Tools

Install these by running:

1. pip install pulp (for Pulp)
2. pip install pandas (for Pandas)
3. python -m pip install --upgrade --user ortools (for Google OR-Tools)

The primary steps for our problem are the following: getting data from the Excel or .csv files, setting up the matrices and formatting the data, and then sending the data to a solver algorithm to get the results.

With that in mind, here is a breakdown of what each line of the python code does:

Lines 5-21: Read data from the input files and puts each column from the spreadsheets into lists/arrays that we can use.

Lines 22-23: These are short hands for the number of teachers and the number of seminars.

Lines 25-26: Initialize the model object and the solver. The model here is an object that contains all the variables, the constraints, and the objective equation. The solver does not have to be initialized here, it could be initialized later. We kept it here so I could change the type of algorithm it uses without scrolling down.

Lines 29-32: Initialize variables. ClassWillRun is a one-dimensional array that is the size of the number of seminars, since it is meant to describe whether or not each seminar will run. StudentAssignments is a 2d array that is accessed in the form 'studentAssignments[s][c]' where 's' is the specific student and 'c' is the specific class. The entry at this point describes whether or not student 's' is taking class 'c' (Boolean values, 0 and 1).

Lines 35-38: Set up a shorthand for the summation term in each of the equations.

Lines 41-49: Set up extra constraints suggested by Dr. Miller, such as ensuring that a teacher is assigned to a seminar from one of the three preferences or no seminar.

Lines 52-59: Set up the constraints for each class

Lines 61-67: Setup for the constraints limiting the maximum number of classes each student can be assigned to.

To give context for the next few sets of lines, 69-99 deal with setting up the objective equation. Lines 69-92 setup the preferences of each student for each class, similar to the student assignments array. In hindsight, I probably should have put this at the start, to organize all the array setup in one place.

Lines 69-78: We have commented out the set up for old method of assigning preferences, since they incentivize bullet voting instead.

Lines 80-92: A better way to set the preferences each student has to each class.

Lines 94-99: Set up the objective equation.

Lines 102-111: Add all the equations to the model object

Lines 114-117: Ask for the model to be solved with the specified solver algorithm and print out basic statistics on if it found an optimal solution and its objective value.

Lines 120-122: Store the variable objects in the studentAssignments arrays. To make outputting results easier, we just convert these into actual numbers (Boolean values so just 0 and 1).

The rest of the code deals with outputting results.

We use the Spyder IDE to run the script, which stores all the variables/arrays that the script created in a variable explorer window. The script itself outputs to the console some statistics on how it found a solution, notably the objective value of the solution and the time it took to solve it in milliseconds. It also outputs the number and percentage of teachers who got their first, second, third preferences, and no assignments.

Conclusion

In essence, this course assignment project applies linear programming to perform effective quantitative modeling and schedule classes based on student and teacher preferences. The goal of this tool is to find a schedule that satisfies the fundamental rules of any academic institution yet optimizing the use of existing facilities. This tool has been tested on several cases with varying sizes and has demonstrated promising results. Therefore, it is complete and will find the best possible solution.