A New Approach to Preplacement for CS Electives at Mudd

Shaurya Pednekar

7th May 2021

Abstract

In this paper, we look at a new approach to preplacement for CS electives at Harvey Mudd College, using Binary Integer Programming. We describe the mathematical model being used to represent the situation and the simplifying assumptions made. The code implementation is in Julia. The runtime of the implementation is then analyzed as a function of the number of students and of the solver being used. Currently, the code runs at a reasonable rate (~ 1 minute) for around 100 students while using the Cbc solver, but grows at an exponential rate. Finally, the paper describes possible future work that can be used to extend the project.

1 Introduction to Binary Integer Programming (BIP)

An integer linear programming model is an optimization model where the objective function and constraints and are linear in terms of the decision variables. Binary Integer Programming is a sub field of Integer Linear Programming, where the decision variables are constrained to be either 0s or 1s. In canonical form, an ILP can be expressed as

$$\begin{aligned} \max \quad & c^T x \\ \text{s.t.} \quad & Ax \leq b \\ & x \geq 0 \end{aligned}$$

where A is a real m by n matrix, c is an n dimensional vector, and b is an m dimensional vector. In an ILP (integer linear program), x is an n dimensional vector of integers. With binary integer programming, there is an additional constraint that $0 \leq x \leq 1$.

It has been proved that integer programming is NP-hard [1].

2 Preplacement Model

Currently, the CS preplacement model that is used at Mudd works in the following way:

1. The college decides which CS courses to offer.
2. The students give their preferences for the courses being offered.

3. The ILP finds the optimal assignment of students to the courses being offered.

With the approach described in this paper, the goal is to take into account the students preferences for courses (CS electives for now) when deciding which courses to offer itself. The reason for this is because professors might offer multiple different electives over different semesters/years, and with this new approach, the elective they choose to offer will be better suited to the current batch of students' preferences. Therefore, the preplacement approach described in this paper is as follows:

1. Each professor gives the courses they are willing to offer (along with details such as course timings and number of seats available).
2. Students distribute a fixed amount of points towards the potential courses.
3. The BIP then decides which courses should be offered from the set of potential courses AND the assignment of students to the courses being offered.

2.1 Simplifying Assumptions

For this final project, the simplifying assumptions made are stated below. Many of these were made since access to real world data (on how the college/professors actually decide which courses to offer) was not available. These can be modified if needed, without many changes, in order to more accurately reflect real world constraints and requirements.

The simplifying assumptions are:

1. This preplacement model deals only with CS electives (not courses that are requirements for a CS major). So, in the context of this paper, *course* is synonymous with *CS elective*.
2. Each professor should offer exactly one course.
3. Each student gets assigned to at least one and at most two courses (since these are electives). This sometimes leads to students being assigned to courses they are not interested in (represented by a negative objective value) - which could be an indication that the college might want to offer more sections of a course or offer different courses.
4. All potential courses offered by a professor take place at the same time. This means that each professor gets a single time slot for all of their potential offerings (which can overlap with other professors' time slots). This was done for convenience of code and can be easily modified if required.

3 Mathematical Representation of the Model

This section describes the variables, objective function, and constraints used to represent this BIP model.

3.1 Decision Variables

These are variables whose value is found by the optimization algorithm in order to maximize the objective function (which is described in the *Objective Function* section).

The binary decision variables used in this model are described below.

- Let m be the total number of potential courses. Then we have one decision variable for each potential course. These is represented as

$$c_j \text{ for } j \in \{1, \dots, m\}$$

where

$$c_j = \begin{cases} 1, & \text{if course } j \text{ should be offered} \\ 0, & \text{otherwise} \end{cases}$$

- Let n be the number of students. We have one decision variable for each assignment of student to course. These is represented as

$$s_{ij} \text{ for } i \in \{1, \dots, n\}, j \in \{1, \dots, m\}$$

where

$$s_{ij} = \begin{cases} 1, & \text{if student } i \text{ is assigned course } j \\ 0, & \text{otherwise} \end{cases}$$

An important point to note is that many of the constraints described in the *Constraints* section would need to include the term $c_i \cdot s_{ij}$. Since $c_i \cdot s_{ij}$ is not a linear term, this could cause problems with convexity and might not be solvable by a linear model. However, since both c_i and s_{ij} are binary, we can linearize the model by adding another set of decision variables z_{ij}. By adding the following four linear constraints, we can ensure that z_{ij} always equals $c_i \cdot s_{ij}$. The four constraints are:

$$z_{ij} \leq s_{ij} \tag{3.1}$$

$$z_{ij} \leq c_j \tag{3.2}$$

$$z_{ij} \geq c_j - (1 - s_{ij}) \tag{3.3}$$

$$z_{ij} \geq 0 \tag{3.4}$$

So,

$$z_{ij} = \begin{cases} 1, & \text{if course } j \text{ is offered AND student } i \text{ is assigned course } j \\ 0, & \text{otherwise.} \end{cases}$$

With this linearization, the model can now be represented using only linear terms!

3.2 Other Variables

In addition to the decision variables, the model also requires the two sets of variables described below. Their values are known before the model runs.

- First, we have a set of variables representing the number of points a student gave for a course (where a higher number of points means a student is more interested in the course). This is represented as r_{ij} where

$$r_{ij} = \text{ student } i\text{'s points for course } j.$$

In order to disincentivize the program from assigning students to courses that they gave zero points, we set r_{ij} equal to -1000 when student i gives course j zero points.

- Next, we have a set of variables representing whether a course j takes place at a time-step k. (An example of a time-step is *M1330*, which represents Mondays at 1:30pm.) This set of variables is represented as t_{jk} where

$$t_{jk} = \begin{cases} 1, & \text{if course } j \text{ takes place at time-step } k \\ 0, & \text{otherwise} \end{cases}$$

- We also have a set of variables for the number of seats available in a course. This is represented as n_j where

$$n_j = \text{ number of seats available in course } j \text{ .}$$

3.3 Objective Function

The objective function for this model is to maximize the total happiness of the students. This is represented as shown below:

$$\max \sum_{i \in n} \sum_{j \in m} r_{ij} \cdot z_{ij}$$

3.4 Constraints

We now describe the constraints of the model.

3.4.1 Time Conflict Constraint

This constraint ensures that no student is assigned to two different courses being offered that take place at the same time, to reflect the constraint that a student cannot be enrolled in two courses that take place at the same time.

This is represented as shown below:

For each time-step k:
For each student i

$$\sum_{j \in m} z_{ij} \cdot t_{jk} \leq 1$$

3.4.2 Bounds on Number of Courses Assigned to a Student

This constraint reflects the simplifying assumption that each student is assigned at least one course and at most two courses.

This is represented as shown below:

For each student i:

$$1 \leq \sum_{j \in m} z_{ij} \leq 2$$

3.4.3 Cap on Number of Students in a Course

This constraint ensures that the number of students assigned to a course is at most equal to the number of seats available in that course.

This is represented as shown below:

For each course j:

$$\sum_{i \in n} z_{ij} \leq n_j$$

3.4.4 Each Professor Offers Exactly 1 Course

This constraint also reflects one of the simplifying assumptions - each professor is constrained to offer exactly one course (CS elective).

This is represented as shown below:

For each professor p

$$\sum_{q \in \text{Prof } p\text{'s courses}} c_q = 1$$

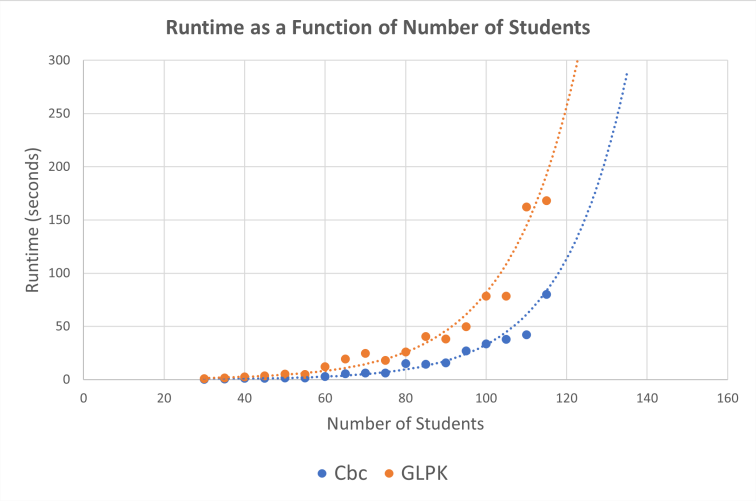
4 Code Implementation

The code for this model was implemented in *Julia* using *Jump Dev* as the optimization package and *Cbc* [2] and *GLPK* as the open source solvers. For the purposes of this final project, the model included 17 potential courses by 7 professors. The course timings were selected based on past course timings at Mudd, as were the number of available seats in a course. The students' distribution of points among the potential courses was done by randomly distributing 10 points among the potential courses for each student.

5 Results

The model ran successfully within a reasonable amout of time when the number of students was around 100. Sometimes, the optimal value was negative, which indicated that in order to satisfy the constraints, some students were assigned to courses that they gave zero points. As mentioned earlier, this is an indication that the college might want to include more sections of a certain course or include different potential courses.

Figure 1: Runtime Analysis Graph



The graph above shows how the runtime of the model varies as a function of the number of students. The runtime at each point was the average of five runs of the model (since the

randomness of the students preferences causes a variation in the runtime). As can be seen, the model is exponential in the number of students. Since Mudd is a small college, this model can still be feasible. However, at bigger colleges, this model would quickly become infeasible when using it for the whole student population. It could, however, still be useful when used with relatively small subsets of the students. Additionally, we see that the *Cbc* solver runs a lot faster than the *GLPK* solver, possibly due to heuristics employed by the *Cbc* solver.

6 Future Work

Possible additions to this project could include

- Modifying the assumptions in order to more accurately reflect real world data.
- Conducting surveys to collect students actual preferences for courses rather than randomly generating it.
- Comparing the runtime with other solvers such as *SCIP* and *CPLEX*.

Bibliography

1. Papadimitriou, Christos H., and Kenneth Steiglitz. *Combinatorial Optimization: Algorithms and Complexity.* Dover Publications, Inc., 2014.

2. Lougee-Heimer, R. "The Common Optimization INterface for Operations Research: Promoting Open-Source Software in the Operations Research Community." *IBM Journal of Research and Development*, vol. 47, no. 1, 2003, pp. 57–66., doi:10.1147/rd.471.0057.