Math Majors Using Math to Help Math Departments: Two Models for Assigning Teaching Assistants to Courses

By

SAMUEL ASHER

SENIOR THESIS

Submitted in partial satisfaction of the requirements for Highest Honors for the degree of

BACHELOR OF SCIENCE

in

MATHEMATICS

in the

COLLEGE OF LETTERS AND SCIENCE

of the

UNIVERSITY OF CALIFORNIA,

DAVIS

Approved:

Jesús A. De Loera

March 2016

ABSTRACT. Every year, graduate students are chosen to assist in the teaching of classes. Traditionally, staff uses the graduate students' course preferences to manually match the graduate students to their preferred classes. Unfortunately, this is a time-consuming process and designated assignments generally fail to produce an optimal solution. In order to remedy this issue, we modify a classical mathematical model for assignment problems, namely integer programming. The model produces a pairing of graduate students to classes while attempting to maximize the satisfaction of the graduate students. In conjunction with this paper, I also created a website interface that gathers preference data from the graduate students and allows the staff to easily perform the matching.

Contents

CHAPTER 1

Introduction

Every quarter, university students flock to discussion sections led by graduate students in the hopes of better understanding the material from lecture. But who decides which graduate students lead which discussion sections? In a process largely taken for granted by many at the university, teaching assistants (hereafter referred to as TAs) are assigned by hand. We view this as a problem that applied mathematics can solve, and thus tackle the conundrum of automatically producing a fair, optimal matching of TAs to discussion sections.

First, we must understand the status quo of manual assignments. The process is usually carried out in two steps. First, the staff surveys all graduate students to find out what classes they would prefer to teach and when they are unavailable to teach. Next, they match the graduate students to the available classes, trying to produce a fair and balanced assignment. Some of the disadvantages of a manually generated schedule include: the time spent by the staff creating the matching, the lack of consistency from year to year, the inability to quickly predict the need of TAs from outside the department, and the introduction of human bias. A mathematical model can produce an impartial, consistent assignment in seconds, which is an excellent resource for the staff responsible for performing the matching.

This paper details the task of designing and implementing a mathematical model to solve the TA-section assignment problem. This is the story we tell here, of producing a tool to better perform the assignment. As we tell this story, we will encounter not just beautiful mathematics but also practical work that has direct connections to Economics Nobel prize winning work.

Our approach will incorporate integer programming and assignment theory, building on the original studies in assignment problems laid out by Koopmans and Kantorovich. In this model, we define binary *assignment variables* for every TA-section pair and then use integer programming to set those variables in a way that maximizes our notion of TA happiness.

My contribution is to modify the traditional theory in order to pair all graduate students to courses while attempting to maximize the satisfaction of the graduate students. To

this end, I created a mathematical model of the TA assignment problem and implemented software to solve the problem for any department. It is important to note that Koopmans and Kantorovich's integer programming approach has been used in solving similar scheduling problems (see [**8**]).

CHAPTER 2

Integer Linear Optimization for Assignment Models

We first recall the basics of integer optimization, and its application to the assignment problem. Alexander Schrijver offers a comprehensive history of these topics [**7**], which we will summarize below.

2.1. A Brief History of Linear Optimizaton

In 1939, a Russian mathematician named Leonid Kantorovich was approached by an engineer who asked for his help with a problem concerning the output of production of his meat-cutting machines. The engineer presented Kantorovich with a list of five machines and a list of eight meats and posed the question of how to optimally assign meats to machines based on productivity rates of the machines. Kantorovich quickly realized that an evaluation of all possible combinations would require searching through a billion or so systems of linear equations, so Kantorovich devised a new way to maximize linear functions under linear constraints. See [**6**] for a detailed explanation. As this discovery was made during World War II, its significance was immediately recognized in many areas of industry, and Kantorovich's work was utilized to "fulfill the needs of the USSR", in areas like the organizing and planning of production.

Several years later in 1942, a Dutch mathematician and economist named Tjalling Koopmans was appointed as statistician of the British Shipping Mission. His task was to analyze the optimal number of shipments needed to account for changes in demand. To do this, Koopmans created a method of the assignment problem that lead to an optimal shipment schedule. He also investigated the economic implications of this method, realizing its significance in resource transfer. Both Koopmans and Kantorovich received the Nobel Prize in Economics in 1975 for their contribution to the field of normative economic theory.

The development of the assignment problem was fundamental to the more general theory of linear programming. In 1947, the simplex method was discovered by George Dantzig (see [**4**] for details), allowing arbitrary linear programs to be solved for an optimal solution. The simplex method is widely used today, and has been implemented into software that can be used by the public. Linear programming and matching theory are at the heart of our model. It is important to note that assignment problems are special linear programs, in that when input data is integral, optimal solutions of linear programs are also integral.

FIGURE 1. Leonid Kantorovich (left), Tjalling Koopmans (middle), and George Dantzig (right)

Essentially, assignment problems were the first examples of *integer* linear programs, which we will describe below.

2.2. The Simplex Method: An Example

Before we discuss the model in detail, we will take a closer look into the computational technique we used to solve it, called *integer linear programming*. An integer linear program is comprised of a set of integer-valued variables, a system of linear equations and inequalities which constrain the variables, and an objective function of the variables, to be maximized or minimized. For a detailed introduction to this topic see [**5**]. Some examples of integer linear programs include the maximization of goods produced by machines in a meat packing company, or the minimization of cost when assigning product delivery routes. To solve an integer linear program one typically solves a sequence of easier problems called *linear programs*, where the variables are no longer required to be integral. We describe linear programs and their solutions in more detail below before returning to the larger problem of solving an integer linear program.

The set of possible solutions of a linear program forms a geometric object known as a polyhedron, or a shape cut out from a system of linear equalities. For example, consider a system of linear inequalities in two variables, so that every inequality can be visualized as a half-plane in the Cartesian plane. The result of plotting this system will then be a two dimensional shaded region, which we call a polyhedron of dimension two or simply a polygon. Formally, we define a half-space as a set $\{x \in \mathbb{R}^n : Ax \leq \alpha\}$ for any given $A \in \mathbb{R}^n$ and $\alpha \in \mathbb{R}$. Then we define a polyhedron as a finite intersection of half-spaces.

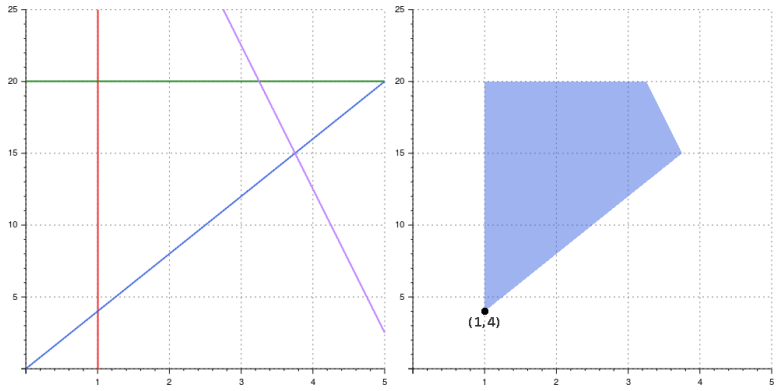
The constraints of a linear program can each be realized as half-spaces, and so their intersection forms a polyhedron. All feasible solutions are encapsulated by the polyhedron, and the simplex method provides an algorithm to traverse such polyhedra to find an optimal

solution (see [**2**] for a conceptual introduction). The simplex method methodically examines the value at each vertex to find the value that yields the highest objective value. This is done by moving from vertex to vertex of the polyhedron, and checking each adjacent vertex. If every neighboring vertex decreases the objective value or does not increase the objective value, the process finishes and the current vertex is the optimal solution. Note that a polyhedron may give multiple optimal solutions, and the process by which it chooses a optimal solution is dependent on the way it chooses which vertices to examine. These are called pivot rules. See [**4**] for a more complete explanation.

To help understand how the simplex method works, we present a simple, two-dimensional example. Consider a farmer who is trying to decide what animals she should buy to raise on her farm. She wants to buy cows and chickens, and would like to purchase as many total animals as she can with the money she has. Say cows cost \$200 each and chickens cost \$20 each, and the farmer has \$1050 to spend. Assume our farmer wants at least one cow and at least four times as many chickens as cows, but her coop will hold no more than 20 chickens.

We can visualize solutions to this farmer's dilemma as points (x, y) in the Cartesian plane, where x is the number of cows and y the number of chickens she will purchase. The farmer wants as many animals as possible, which means she is trying to maximize the objective function $x + y$.

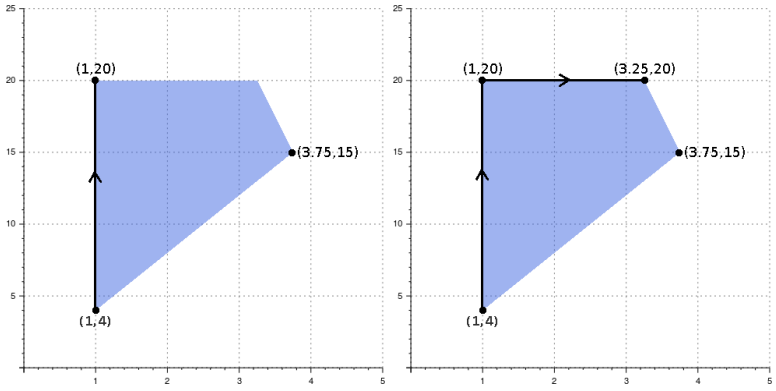
The farmer's financial and practical constraints can be represented as the set of inequalities $\{200x + 20y \leq 1050, x \geq 1, y \geq 4x, y \leq 20\}$. We graph the constraints below:



Notice that these lines cut out a polygon, which is a 2-dimensional polyhedron. Every point inside of the polygon is a valid solution to the farmer's problem. The question that remains is which of these points is the optimal solution. The answer can be found using the simplex method. As we said before, the central idea behind the simplex method is that optimal solutions are found among the vertices of the polyhedron. Thus, we can find the optimal solution by traversing the vertices until we cannot improve our objective value by traveling to a new vertex. In this example, our objective is to maximize $x + y$, the total number of animals. To begin the simplex method, we need to start at an arbitrary vertex

that gives a feasible solution. It is easy to check that $(1, 4)$ is a feasible solution, so we begin there.

Then we look to the neighboring vertices to see if we can improve our objective value, the total number of animals. We can move directly up to attain an objective value of $1 + 20 = 21$ or to the right diagonal to reach $3.75 + 15 = 18.75$. We choose to move up to reach the higher value. We again look to our neighbors to see if we can improve. We see that we can move directly to the right to attain 23.25.



Now we look to our neighbors and see that they have strictly lower objective values. Thus, we have found our optimal solution, $(3.25, 20)$. Though the simplex method gets much more complicated with higher dimensions and more constraints, the idea remains the same. As long as our objective function and constraints are linear, we are guaranteed to find an optimal solution if one exists. This is the fundamental theorem of linear programming, proved in [**9**].

From the example above, we observe that the farmer should purchase 3.25 cows and 20 chickens. As one might expect, it is unrealistic for the farmer to purchase one quarter of a cow. Instead we must find some way to extract a feasible *integer* solution. This can be done using the *branch and bound method*.

2.3. Example Continued: Branch and Bound

First developed by Alisa Land and Alison Doig in 1960 (see [**9**]), the branch and bound method is a divide and conquer method that can be used to solve integer linear programs (this is complemented with other techniques, see [**3**]). By applying a linear program relaxation (LP-relaxation) to an integer program, in which we allow the variables to take real values instead of integer values, we can arrive at an optimal integer solution by solving a sequence of real-valued linear programs.

To describe this process, we would first break the ILP into multiple subproblems and then apply an LP-relaxation to each subproblem to see if the LP is feasible (has an integer solution). If it is, we compare it to the optimal bound we currently have. If we find a

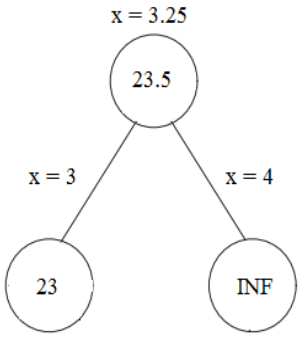
solution that gives a better objective value than our optimal solution, but is not integer, we repeat the process. All subproblems that yield worse objective values than the current optimal integer solution will be eliminated. In this way, we eventually reach an optimal integer solution.

We revisit the fledgling farmer problem, and will now use branch and bound to attain an optimal integer solution. As we did above, we solve the system of linear constraints $\{200x + 20y \leq 1050, x \geq 1, y \geq 4x, y \leq 20\}$ and get the solution $x = 3.25$ cows, $y = 20$ chickens, and an objective function value of $x + y = 23.25$.

Seeing this has a non-integer solution, we now apply branch and bound. Bounding our solution, we will now branch into two subcases: $x = 3$ and $x = 4$.

With $x = 3$ we apply the simplex method again to get an objective function value of 23 with y = 20. Since both x and y are integral, we hold 23 as our current optimal integer solution while we test other cases.

Likewise with $x = 4$ we apply the simplex method and find that the solution is infeasible. This is because when the farmer purchases 4 cows, she must also purchase at least 16 chickens, which costs her $1120 total, exceeding her budget.



Since we have found that one branch leads to an integer solution and the other branch leads to an infeasible solution, $x = 3, y = 20$ is the optimal integral solution. We have thus used the branch and bound method to "relax" the problem such that it gives us an integer solution $(3, 20)$.

2.4. The Assignment Problem

A natural application of integer programming is the assignment problem. We present a common example in which we have a set of males X and a set of females Y who are seeking engagement. To formulate this in terms of an integer program, we define a set of assignment variables x_{ij} such that

$$x_{ij} = \begin{cases} 1; & \text{male } i \text{ and female } j \text{ are engaged,} \\ 0; & \text{else.} \end{cases}$$

Now we need to put some constraints on the problem. For instance, each male i should only be engaged to at most one female, and each female j should only be engaged to at most one male. In terms of mathematical constraints, we write

- $\forall i \in X, \quad \sum_{j \in Y} x_{ij} \leq 1,$
- $\forall j \in Y, \quad \sum_{i \in X} x_{ij} \leq 1.$

Next, let us assume that we can represent the happiness of a couple with a numerical value $h_{ij} \in \mathbb{R}_{\geq 0}$ for $i \in X$ and $j \in Y$. Of course, this is a gross oversimplification of representing happiness; we will return to the problem of mathematically modeling happiness shortly when we discuss the TA assignment problem.

Now we would like a matching of males and females that yields the greatest amount of happiness among all couples. In terms of integer programming, we would like to maximize the following linear objective function:

$$\sum_{i \in X, j \in Y} h_{ij} x_{ij}.$$

Since the x_{ij} are our variables, and the h_{ij} are strictly positive, maximizing this function would mean setting as many x_{ij} to 1 as possible. But our constraints give us a limit on how many variables we can set to 1, forcing us to choose x_{ij} strategically. It is clear that even in this simple example, solving an integer program by inspection can be extremely difficult. Fortunately, a problem formulated in this way can be very quickly optimized with a good IP solver.

CHAPTER 3

The TA Assignment Model

3.1. Modeling "Happiness"

We now begin modeling our TA assignment problem as an integer program. Note that we will be using "TA pairings" to describe matchings in this section. First let T the set of TAs and S the set of sections. Similar to the above example, we define the binary assignment variables x_{ij} to represent whether TA $i \in T$ is assigned to section $j \in S$. In order to define TA happiness, we considered five main components:

1. Which courses the TA prefers to teach,
2. Which times of the day the TA prefers to teach,
3. Whether or not the TA prefers to teach sections from the same course,
4. Whether or not the TA prefers to teach sections on the same day,
5. Whether or not the TA prefers to teach back-to-back sections.

Each of these components factors into our objective function in a different way, but contributes to the overall satisfaction of the participants.

3.1.1. Course Preference. Each TA has their own mathematical interests that lead them to find certain undergraduate courses more enjoyable to teach than others. Most often, a TA will have a small set of courses that she particularly likes, another small set of courses she particularly dislikes, and a large portion of remaining courses that she is indifferent towards. We can thus think of a TA's preference in courses to teach as a ranked list in descending order. For example, let us introduce four TAs and four courses. Call our TAs Alice, Bob, Charlie, and Diane, and our courses Calculus, Algebra, Analysis, and Combinatorics. Then we show their ranked lists below. A $\star$ denotes a course that is preferred by that TA, a $\otimes$ denotes a course that is not preferred, and a $\diamond$ indicates indifference.

Alice	**Bob**	**Charlie**	**Diane**
⋆ Algebra	⋆ Analysis	⋆ Algebra	⋆ Combinatorics
⋆ Combinatorics	⋄ Algebra	⋄ Combinatorics	⋆ Analysis
⋄ Analysis	⊗ Combinatorics	⋄ Calculus	⋆ Algebra
⊗ Calculus	⊗ Calculus	⊗Analysis	⊗ Calculus

Now we give each TA-course pair a numerical value p_{ik} on a scale from 0 to 100 that represents how much TA i likes course k. The score of 100 is awarded to the TA's favorite class, 0 is given to the TA's least favorite class, and 50 is given to all courses towards which the TA is indifferent. We assign a normalized value to every other course based on the total number of courses. In our example, Alice will give 100 to Algebra, 50 to Analysis, and 0 to Calculus. She gives 75 to Combinatorics because that is the halfway point between 100 and 50. On the other hand, Diane gives Analysis a 83.3 and Algebra a 66.6, since these two numbers evenly split the interval between 50 and 100 into three equal parts.

Formally, we define Φ_{ik} the (positive) rank that TA i gives course k if i likes k, and ϕ_{ik} the (negative) rank i gives k if i dislikes k. In our example, if a is Alice, A is Algebra, and C is Combinatorics, then $\Phi_{aA} = 1$ and $\phi_{aC} = 1$. Then to account for indifference and normalizing, we let L_i the set of courses liked by i, D_i the set of courses disliked by i, and then formally define p_{ik} as follows:

$$p_{ik} = \begin{cases} 100 - \frac{50}{|L_i|}(\Phi_{ik} - 1); & i \text{ likes } k; \\ 50; & i \text{ is indifferent towards } k; \\ 0 + \frac{50}{|D_i|}(\phi_{ik} - 1); & i \text{ dislikes } k. \end{cases}$$

This looks unnecessarily complicated, but is just a method to convert likings/dislikes into normalized numerical values. Since every section belongs to a course, the course preference p_{ik} is a factor towards how much a TA prefers a given section. Next we consider another factor: the time of day during which the section is held.

3.1.2. Time of Day Preference. Discussion sections are held at different times throughout the course of the day. Some can start as early as 7:00 a.m. while some begin as late as 7:00 pm. As with courses, TAs have (often strong) preferences about what times of day they prefer to hold discussion sections. We can use our ranking model from above to assign numerical values to these preferences as well.

In our model, we divided a day into hours, so that each TA could rank each hour of the day. For simplicity, we will use the broader divisions of Morning, Afternoon, and Evening to continue our example of TA preferences. Again, with ⋆ denoting preferred times to teach, ⊗ denoting disliked times to teach, and ⋄ denoting indifference, we present the time preferences of our above TAs:

Alice	**Bob**	**Charlie**	**Diane**
⋆ Afternoon	⋆ Morning	⋆ Afternoon	⋆ Evening
⋆ Morning	⋄ Evening	⋄ Evening	⋆ Afternoon
⋄ Evening	⊗ Afternoon	⊗ Morning	⊗ Morning

Similar to our course ranking model, we define Ψ_{it} the ranking TA i gives to time t if i likes t, and ψ_{it} the negative ranking i gives to t if i dislikes t. These are calculated in exactly the same way as the Φ and ϕ above. We also account for indifference and normalizing the same way: given T_i the set of times liked by i, and τ_i the set of times disliked by i, we define the time preference p_{it} as follows:

$$p_{it} = \begin{cases} 100 - \frac{50}{|T_i|}\Psi_{it}; & i \text{ likes } t; \\ 50; & i \text{ is indifferent towards } t; \\ 0 + \frac{50}{|\tau_i|}\psi_{it}; & i \text{ dislikes } t. \end{cases}$$

Now we have a measure of a TA's time of day preference p_{it}. We will next show how to combine this with the previously defined course preference p_{ik} to produce an overall section preference.

3.1.3. Section Preference. A given section j has both a corresponding course k and a corresponding time of day t. So for a TA i, we have two values to look at when considering section preference, namely p_{ik} and p_{it}. These two values have different weights to different TAs. For instance, some TAs care only about the course material and not when they hold discussion. Thus to define p_{ij}, the preference of TA i for section j, we have implemented a weight system with the following five choices:

$$p_{ij} = \begin{cases} p_{ik}; & i \text{ is only concerned with the course material of a section;} \\ \frac{2*p_{ik}+p_{it}}{3}; & i \text{ values the course material of a section higher than the time of day;} \\ \frac{p_{ik}+p_{it}}{2}; & i \text{ values the time of day and course material of a section equally;} \\ \frac{p_{ik}+2*p_{it}}{3}; & i \text{ values the time of day of a section higher than the course material;} \\ p_{it}; & i \text{ is only concerned with the time of day of a section..} \end{cases}$$

This allows a more personalized preference of a given section. For instance, in our running TA example, say Alice has the second weight preference; that is, she values the course material of a section higher than the time of day. Then say we have a section j which is a Algebra discussion in the Evening. Then for Alice as a, Algebra as A, and Evening as E, we have $p_{aA} = 100$, $p_{aE} = 50$, and so $p_{aj} = \frac{2*100+50}{3} = 83.3$. In this way, we can gather p_{ij} for every TA i and section j. This is only part of the data we need; as we will see, it is not just the individual courses but rather the combination of courses that makes a TA teaching schedule good or bad.

3.1.4. Sections from the Same Course. One of the most important components to the happiness of several TAs is being able to teach sections from the same course. Presumably, the fewer distinct courses among sections a TA is teaching, the less time that TA must spend preparing material from different topics. The wish of many TAs is to teach as many sections as possible from the same course. To model this, we aggregate on the x_{ij} as follows. Let C the set of courses, $i \in T$ a TA, $j \in S$ a section, $k \in C$ a course, and $l \in C$ the course of section j. Then we define δ_{kl} a simple binary indicator of whether l and k are the same section. Formally, we write

$$\delta_{kl} = \begin{cases} 1; & l = k; \\ 0; & l \neq k. \end{cases}$$

Then we can define q_{ik} a binary variable such that q_{ik} is 1 if TA i is assigned to at least one section of course k and 0 otherwise. To force this, we write:

$$q_{ik} \geq \sum_{j \in S} \frac{x_{ij}}{m} \delta_{kl}.$$

Here m is the maximum number of units a TA can teach. We include it here as a normalizer to ensure that q_{ik} does not exceed 1. The q_{ik} is forced into its proper value as follows: if TA i is assigned to teach section j and section j is from course k, then $l = k$ so we have that $\delta_{kl} = 1$ and $x_{ij} = 1$. So the right-hand side is strictly greater than zero and since q_{ik} is binary, this means it must be set to 1.

Now we can define z_i the total number of distinct courses that TA i is teaching sections from. Given that we already have the q_{ik} set for every course k, we can simply sum up the q_{ik} to get the total number of distinct courses. We write:

$$z_i = \sum_{k \in C} q_{ik}.$$

An important remark here is that the more sections a TA teaches, the more difficult it is to ensure that z_i is low. That is, every TA wants a z_i of 1, but that is much more difficult for a TA who is teaching four sections versus one who is only teaching one section. We see that we need to account for the total number of sections that TA i is teaching, which we denote μ_i. What we want is a function f that rewards the LP for giving few courses to TAs with many sections. It turns out that a linear function serves our purpose well:

$$f(\mu_i, z_i) = \mu_i - z_i + 1.$$

We add the 1 to ensure that f never gives a value of zero. Thus the lowest value f can return is 1, which happens precisely when $\mu_i = z_i$. Of course, as the number of sections taught μ_i increases, we would like to keep z_i, the number of distinct courses, low. Since μ_i is unchangeable data, but z_i depends on the assignments that the LP makes, the LP will attempt to keep z_i small; that is, prevent TAs from teaching sections from too many distinct courses.

3.1.5. Sections on the Same Day. Next we want to define a measure for the number of days a TA must teach on. The goal here varies by TA; some prefer to teach all of their sections on the same day, while others would rather spread their sections throughout the week.

To enumerate the total distinct days on which a TA i is teaching, let D the set of days of the week, j a section and $d \in D$ a day of the week. Then we define υ_{jd} the binary indicator of whether section j is held on day d. Given υ_{jd}, we can calculate the binary indicator y_{id} of whether TA i teaches on day d. Similar to above, we define:

$$y_{id} \geq \sum_{j \in S} \frac{x_{ij}}{m} \upsilon_{jd}.$$

If TA i is teaching section j, and section j is held on day d, then $\upsilon_{jd} = 1$ and $x_{ij} = 1$, so the right hand side is strictly greater than zero, which forces y_{id} to be 1 since it is binary.

Now we'd like to calculate the total number of days on which TA i must teach, which we call v_i. As there are only five days in the week, we only have five y_{id} for a given TA i. To obtain v_i, we need only sum these up:

$$v_i = \sum_{d \in D} y_{id}.$$

As with number of courses, we need to normalize v_i over the number of sections TA i is teaching. We can use the same linear function as before:

$$g(\mu_i, v_i) = \mu_i - v_i + 1.$$

Here, as before, we want g to take a higher value when a TA with many sections has few days. We will see later how to change the interpretation of g based on whether TA i prefers to teach courses on the same day or not.

3.1.6. Back-to-back Sections. Finally we would like a measure of how many back-to-back sections a TA has. This is another divided issue, as some TAs prefer to have their sections temporally close while others would rather have them spaced out. Notice that this is not the same as having sections on the same day: a TA could prefer to have sections on the same day but not immediately back-to-back.

First, for a TA i and sections j,j', we define $\nu_{jj'}$ the binary indicator of whether sections j and j' are back-to-back. This can be calculated ahead of time simply using section data. Next we want to identify the back-to-back sections that a given TA is assigned to. We define $\beta_{ijj'}$ the binary indicator of whether TA i is teaching back-to-back sections j and j'. To force β to take proper values, we write:

$$\frac{1}{2}\nu_{jj'}(x_{ij} + x_{ij'}) \geq \beta_{ijj'} \geq \frac{1}{2}\nu_{jj'}(x_{ij} + x_{ij'} - 1).$$

If TA i is not teaching both j and j', then the left hand side can be no more than 1/2 and the right hand side can be no more than -1/2. Since β is binary, this forces β to be zero.

If TA i is teaching j and j', but they are not back to back, then $\nu_{jj'}$ will be zero, forcing β to zero. The case where we want β to be 1 is when TA i is teaching j and j', and they are back to back. In this case, $\nu_{jj'} = 1$, $x_{ij} = 1$, and $x_{ij'} = 1$, making the left hand side 1 and the right hand side 1/2, which forces $\beta_{ijj'}$ to be 1 as desired.

Now we want to calculate the total number of pairs of back-to-back sections that are assigned to TA i, which we call w_i. We can simply add up all the $\beta_{ijj'}$ to capture every possible back-to-back section pair:

$$w_i = \sum_{(j,j') \in SxS} \beta_{ijj'}.$$

Once again, we normalize over the number of sections a TA is teaching. After all, if a TA is only teaching one section, it is impossible for her to have back-to-back sections. We chose another linear normalizing function based on μ_i, the total number of sections TA i is teaching:

$$h(\mu_i, w_i) = \mu_i - w_i.$$

This function h works much the same way as the function f (the function that rewards the assignment of few courses to TAs with many sections) in that h returns a higher value when a TA has many sections but few back-to-back sections. We no longer need to add 1 here since μ_i can never equal w_i so we do not need to worry about h returning zero.

3.2. The "Fairness" Factor

Now that we have an idea of how to model "happiness", we have to incorporate fairness into our model. What exactly is meant by "fairness"? What we refer to as "fairness" can be described in terms of whether or not a TA deserves to be assigned to his or her preferred section. The staff of the Math department assign each TA a numerical rank R_i, ranging from zero to five. When assigning these ranks, the staff takes into account factors like prior performance, ability, and seniority.

To see why fairness is an important piece of our model, consider two TAs, Edgar and Frank, both of whom prefer to teach the same section, section y. Frank has a high ranking and Edgar a low ranking. Frank prefers course y over every other course except course y' while Edgar prefers course y to every other course. However, Frank's schedule for the quarter will not allow him to teach course y'. If the model assigned TAs to sections based solely on happiness, this would force Frank to teach some other course simply because the course he preferred had a time conflict while giving his second favorite course to Edgar, who, based on his ranking, did not deserve to teach course y as much as Frank. This simple scenario is an example of a case where considering happiness is not sufficient. To maximize the satisfaction of TAs we needed to prioritize fairness to ensure that the TAs who were "more deserving" of sections they preferred were assigned those sections.

This gives this assignment model great power, in the sense that the staff can choose whether or not a TA deserves to be "happy" based on performance factors and any other

criteria the faculty may evaluate. To see in detail how this "fairness" factor was implemented, we will examine the objective function.

3.3. The Objective Function

The objective function describes exactly what we attempt to optimize. For this project, we wanted to maximize the number of TAs that are assigned to sections they prefer given their rank.

As described above, we model TA happiness with four main components: the section preference p_{ij}; teaching sections from the same course, represented by $f(\mu_i, z_i)$; teaching sections from the same day, represented by $g(u_i, w_i)$; teaching back-to-back sections, represented $h(\mu_i, v_i)$. As we alluded to before, some TAs prefer to teach sections on the same day or back-to-back, while others prefer not to. We account for these differences by including constants a_i, b_i, and c_i which represent the preferences of TA i regarding teaching sections of the same course, teaching sections in the same day, and teaching back-to-back sections, respectively.

- a_i is 1 if TA i prefers to teach sections of the same course, -1 if i would rather not, and 0 if i is indifferent. Most if not all TAs choose a_i to be 1, which means that the objective function receives a bonus for assigning these TAs sections from the same course.
- b_i is 1 if TA i prefers to teach sections on the same day, -1 if i would rather not, and 0 if i is indifferent. If a TA chooses b_i to be 1, then the objective function will receive a bonus for assigning that TA sections in the same day; however, if a TA chooses b_i to be -1, then the objective function will receive a penalty if it assigns that TA sections in the same day.
- c_i is 1 if TA i prefers to teach sections that are back-to-back, -1 if i would rather not, and 0 if i is indifferent. As with b_i, the objective function will receive a bonus if it gives TA i back-to-back sections as long as c_i is positive, and will receive a penalty for doing so if c_i is negative.

Given all of the preferences above, the objective function will attempt to find courses that match what the TAs prefer. We have now defined every piece of the objective function, and so are now ready to present it:

$$\sum_{i \in T} R_i * (a_i * f(\mu_i, z_i) + b_i * g(u_i, w_i) + c_i * h(\mu_i, v_i) + \sum_{j \in S} x_{ij} p_{ij}).$$

3.4. Constraints

Not just any set of values for x_{ij} is a feasible solution for our problem. We need to introduce some constraints on the variables, for instance:

- For every section j, we have the condition that
$$\sum_{i \in T} x_{ij} = 1,$$
which simply means that every section requires exactly one TA.
- For every TA i, we require that there is no overlap between the sections each TA teaches. First, given two sections j and j', we define a binary indicator $\Omega_{jj'}$ that represents whether j and j' temporally overlap. We can determine the Ω ahead of time using data provided by the university or an independent authority (often the times and classroom corresponding to a course is a decision taken outside departments) .

  Now we define a binary indicator of whether TA i is teaching overlapping sections j and j', which we call $\xi_{ijj'}$. We can force the ξ to represent this by enforcing:
$$\frac{1}{2}\Omega_{jj'}(x_{ij} + x_{ij'}) \geq \xi_{ijj'} \geq \frac{1}{2}\Omega_{jj'}(x_{ij} + x_{ij'} - 1).$$
This is very similar to the way we priorly forced the β value to behave in our back-to-back sections formulation. If TA i is not teaching one of the sections j or j', or if j and j' do not overlap, then $\xi_{ijj'}$ is forced to be zero. Otherwise, the left hand side is 1 and the right hand side is $1/2$, so $\xi_{ijj'}$ is forced to be one.

  Now that we have indicators of when a TA takes overlapping classes, we need to ensure this situation never happens. We can enforce this by ensuring that no ξ is ever 1. Formally, for every TA i, we require that
$$\sum_{j,j' \in S \times S} \xi_{ijj'} = 0.$$
- We also require that every TA, i, meets their teaching duties. Recall that μ_i represents the number of sections a TA needs to teach, which is data provided by the staff. To ensure that each TA i teaches exactly this many sections, we write
$$\sum_{j \in S} x_{ij} = \mu_i.$$

We account for the teaching restrictions of the TAs as well. During their first few quarters, TAs are often restricted to teaching lower-division sections until they have adequately prepared for upper-division and graduate-level material. Denote by L_T the set of TAs only qualified to teach lower-division undergraduate sections, U_T the set of TAs only qualified to teach undergraduate sections, U_S the set of upper-division undergraduate sections, and G_S the set of graduate sections:

- TAs who were only allowed to teach lower division courses were subject to the following constraint:
$$\forall (i,j) \in L_T \times (U_S \cup G_S), x_{ij} = 0,$$

- and TAs who were not allowed to teach graduate courses were subject to the following constraint:

$$\forall(i,j) \in (L_T \cup U_T) \times G_S, x_{ij} = 0.$$

3.5. The Size of the Model

Linear programs are most often measured by the number of variables they introduce and the number of constraints they enforce. By examining these numbers, one can get an idea of the scale of our model and how it might perform in practice. Recall that we defined T the set of TAs, S the set of sections, C the set of courses, and D the set of days of the week.

3.5.1. The Number of Variables in a Solution. In a given solution we must calculate the following:

- x_{ij} $\forall(i,j) \in T \times S$,
- q_{ik} $\forall(i,k) \in T \times C$,
- y_{id} $\forall(i,d) \in T \times D$,
- $\beta_{ijj'}$ $\forall(i,jj') \in T \times (S \times S)$.

So a solution has to assign values to $|T| * (|S| + |C| + |D| + |S|^2)$ variables. In an average run, our program has about 20,000 variables.

3.5.2. The Number of Constraints.

- $|T|$ constraints to ensure each TA meets his or her teaching duties.
- $|S|$ constraints to ensure each section has one TA.
- $|T| * |S|^2$ constraints ensure that no TA is assigned classes that occur at the same time.
- $|L_T| * (|U_S| + |G_S|)$ constraints ensure no TA unfit to teach Upper division and graduate level courses does so. Bound this above by $|T| * |S|$.
- $|U_T| * |G_S|$ constraints ensure no TA unfit to teach graduate courses does so. This can also be bounded above by $|T| * |S|$.
- We also have to take into account constraints due to TA schedule conflicts. Let this number of conflicts be Q.
- Then the total number of constraints is about $|T| + |S| + |T| * |S|^2 + |T| * |S| + |T| * |S| + Q$. In our experience, an average run contains about 500,000 constraints.

3.6. Conclusions

A theoretical model is excellent on paper but does not help the TAs or the staff unless it is presented in an accessible form. To convert our theory into a usable product, we designed a user-friendly, self-contained system that can solve the TA assignment problem set up by our model. We provided a simple and clean web interface that abstracted the advanced optimization at work. For the underlying database, we chose to use SQLite. PHP was our programming language of choice because it allowed us to access both the database

and the ILP solver from inside the website code. We developed two main websites: one for TAs to enter preference data and one for staff to manage the data and run the solver. Without this purely engineering element the contribution would not have been as practical as it turned out to be.

To solve the integer program we have explained thus far, we utilized the integer program solver SCIP (Solving Constraint Integer Programs) (see [**1**] for documentation). SCIP is itself an solver that takes in an integer program and returns an optimal solution. It performs state-of-the-art methods, including a sophisticated branch-and-bound and more, to solve integer programs. To solve the underlying linear program it uses the software SOPLEX ([**10**] provides more information), which runs a variant of the simplex algorithm. We chose SCIP because it is free, fast, reliable, and integrates nicely with C++ and PHP, the languages we used to program the interface. Both the software and interface streamline the TA assignment process greatly, allowing the staff to reach an optimal assignment with less than 15 minutes of work.

The utilization of integer linear programming to solve the TA assignment problem gives one an idea of the power and versatility of integer linear programs. Not only does it provide us an efficient way to solve very complicated problems, but it allows us to model problems such that we can achieve optimality, which we simply cannot hope to do using other methods.

Bibliography

1. T. Achterberg, *SCIP: Solving constraint integer programs*, Mathematical Programming Computation **1** (2009), no. 1, 1–41, `http://mpc.zib.de/index.php/MPC/article/view/4`.
2. R.G. Bland, *The allocation of resources by linear programming*, Scientific American **244** (1981), 126–144.
3. M. Conforti, G. Cornuejols, and G. Zambelli, *Integer programming*, Graduate Texts in Mathematics, Springer, 2014.
4. J. Matouek and B. Gärtner, *Understanding and using linear programming*, Universitext, Springer Berlin Heidelberg, 2007.
5. G. Nemhauser and L. Wolsey, *Integer and combinatorial optimization*, Wiley-Interscience Series in Discrete Mathematics and Optimization, John Wiley & Sons, Inc., 1999.
6. L. Rüschendorf, *Monge-Kantorovich transportation problem and optimal couplings*, Jahresbericht der DMV **3** (2007), 113–137.
7. A. Schrijver, *On the history of combinatorial optimization (till 1960)*, Handbooks in Operations Research and Management, Elsevier, 2005.
8. A.S. Schulz, *From linear programming relaxations to approximation algorithms for scheduling problems: A tour d'horizon*, Surveys in Operations Research and Management Science (To Appear).
9. R.J. Vanderbei, *Linear programming: Foundations and extensions*, International Series in Operations Research & Management Science (Book 114), Springer, 1996.
10. R. Wunderling, *Paralleler und objektorientierter Simplex-Algorithmus*, Ph.D. thesis, Technische Universität Berlin, 1996, `http://www.zib.de/Publications/abstracts/TR-96-09/`.