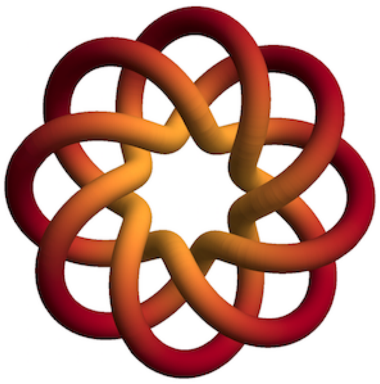
The Minnesota Journal of Undergraduate Mathematics

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

Samuel Asher, Trevor Chan , and Jesús De Loera

University of California, Davis



The Minnesota Journal of Undergraduate Mathematics

Volume 7 (2021-22 Academic Year)

Sponsored by
School of Mathematics
University of Minnesota
Minneapolis, MN 55455

The Minnesota Journal of Undergraduate Mathematics

Volume 7 (2021-22 Academic Year)

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

Samuel Asher, Trevor Chan* , and Jesús De Loera

University of California, Davis

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 modified two classical mathematical models for assignment problems, namely the Gale-Shapley algorithm and integer programming. Both of our models produced a pairing of graduate students to classes while attempting to maximize the satisfaction of the graduate students. We also created a website interface that gathers preference data from the graduate students and allows the staff to easily perform the matching.

1. Introduction

The MAA's 2004 Committee on the Undergraduate Program in Mathematics Curriculum Guide states that *"every course should present key ideas and concepts from a variety of perspectives; employ a broad range of examples and applications to motivate and illustrate the material; promote awareness of connections to other subjects (both in and out of the mathematical sciences) and strengthen each student's ability to apply the course material to these subjects; introduce contemporary topics from the mathematical sciences and their applications, and enhance student perceptions of the vitality and importance of mathematics in the modern world."* It is in our view very important to respond to this call with examples of real life applications solved by real mathematics undergraduates. In this article we present a wonderful example of the synergy that exists between what students learn in basic undergraduate courses in optimization and how they can use it to solve problems existing in their own home department. The often heard question "When am I going to use what I learned in this course?" has a clean answer: Now.

Every term, mathematics departments must go through the process of assigning each discussion section to a graduate student in the department. This process is usually carried

* *Corresponding author*

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. It should be noted that in our algorithm we don't address the important problem of TAs who want to teach certain classes, but may not be qualified to teach those classes. 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, 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.

Two groups of undergraduate math and applied math majors took on the task of designing and implementing two mathematical models 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 work that is quite practical that has direct connections to work that can be traced back to two separate Nobel Prizes in Economics.

The first approach incorporates the graph algorithms of Gale and Shapley that used a model of marriage stability to achieve some notion of "satisfaction" with the assignment. A matching or a "marriage" is called *stable* if an unsatisfied partner cannot find another "cheating" partner that would be willing to break their matching to do a switch of partners. We recount the method in the next section and then we explain how to modify it for our purposes.

The second approach incorporates 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.

Our 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, we created mathematical models of the TA assignment problem and implemented software to solve the problem for any department. Not surprisingly, the methods achieve different answers to the problem of finding the best matching. It is important to note that this is one of many applications of both methods: Shapley's stable marriage approach has been employed to match kidneys donors to medical residents and students to medical schools [6] while Koopmans and Kantorovich's integer programming approach has been used in solving similar scheduling problems (see [13]).

2. The Stable Marriage Approach to the Assignment Problem

Our problem is one of a multitude of applications that resemble the *assignment problem* (see [7] for an introduction), in which a number of agents must be assigned to an equal number of tasks as dictated by preference lists. In this paper, we use the terms *assignment* and *matching* interchangeably to refer to a solution to such a problem. Our problem is a modification of the *stable marriage problem* (SMP), in which marriages between a set of men and women are determined by each sex's list of preferences of their potential partners. A stable assignment is one in which there does not exist a man and woman who would prefer to be matched together instead of with their current partners. In our problem, we explore how to assign teaching assistants (TAs) to discussion sections. The TA course assignment problem requires us to allow for incomplete, partially ordered preference lists as well as account for different qualification criteria to find the best possible assignment. On one hand, we have the preferences of the TAs, but on the other we have departmental preferences of who should teach which class, as determined by TA seniority and feedback from TA evaluations. It goes without saying that the assignments take potential time conflicts and individual qualifications into consideration as well. Additionally, it should be noted that TA preference is not indicative of TA teaching effectiveness. As such, this work may in the future be extended to account for a TA's teaching capability when assigning courses.

We will introduce the algorithm that will be used for our program and explain what factors are taken into account given the problem we are dealing with. This will be followed by a description of what our matching does and how it improves the matching process.

Every matching problem has a common general form. There are two distinct sets, X and Y, of agents and tasks, respectively, where elements in one set must be matched or assigned to one or more elements in the other set. Each element of each set has a preference list of the elements of the other set. This concept can be expressed as a bipartite graph; that is, a graph that can be partitioned into two mutually exclusive sets of vertices X and Y such that for all $x \neq x' \in X$ the edge (x, x') does not exist, and for all $y \neq y' \in Y$ the edge (y, y') does not exist (i.e., we cannot have two points from the same set connected to one another by an edge). We show an example below:

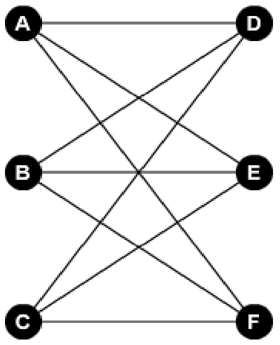


FIGURE 1. A bipartite graph. Here the two sets are $\{A, B, C\}$ and $\{D, E, F\}$.

The stable marriage problem (SMP) is a specific type of the matching problem by Gale and Shapley [7]. The referred authors present both the problem and a solution to the stable marriage problem which can also be termed a stable matching. The title stable marriage comes from their informal description of the problem, as seen in [7]: *"A certain community consists of n men and n women. Each person ranks the opposite sex in accordance with his or her preferences for a marriage partner. We seek a satisfactory way of marrying off all members of the community. Imitating our earlier definition, we call a set of marriages* unstable *(and here the suitability of the term is quite clear) if under it there are a man and a woman who are not married to each other but prefer each other to their actual mates.*

A *stable matching* is an assignment for which, given any two pairs (x,y) and (x',y'), none of the following are true:

(1) x prefers y' over his current partner y

(2) y' prefers x over her current partner x'

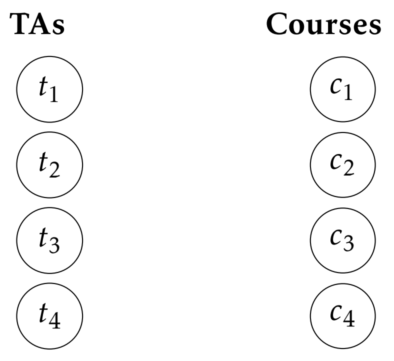
Our goal is to assign teaching assistants to discussion sections in an unbiased manner such that an *acceptable* matching is achieved under as many constraints as given by the user. More specifically, we want a stable matching – a matching that would be most preferable for both parties involved (the department and the TAs) To obtain this matching, we turn to a well-known algorithm for solving SMPs: the Gale-Shapley Algorithm.

The Gale-Shapley Algorithm is a famous algorithm developed in 1962 by mathematicians David Gale and Lloyd Shapley to solve the Stable Marriage Problem in polynomial time (see [7]). Later, with Dr. Alvin Roth, Lloyd Shapley won the 2012 Nobel Prize in Economics for his work. It takes equal-sized sets of men and women as the input, each with their own strictly-ordered and complete preference lists. The algorithm works as follows: In an iterative process, each member of the proposing set proposes to his most preferred partner. If the proposee is not engaged, she accepts. If she is engaged, she accepts only if she prefers the proposer to her current fiancé. If the proposer is still not accepted by the proposee, he will propose to the next preferred proposee. This process continues until everyone is engaged. Below follows a simple example of the algorithm.

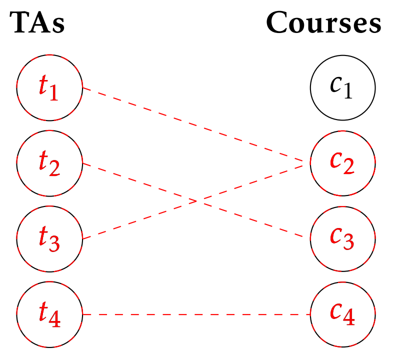
For example we can consider a particular collection of four TAs (denoted t_1, t_2, t_3, t_4) and four courses (denoted c_1, c_2, c_3, c_4) and their particular preference lists (courses themselves have preferences as dictated by the department):

TA's Preferences					**Course's Preferences**				
$t_1:$	c_2	c_4	c_1	c_3	$c_1:$	t_2	t_1	t_4	t_3
$t_2:$	c_3	c_1	c_4	c_2	$c_2:$	t_4	t_3	t_1	t_2
$t_3:$	c_2	c_3	c_1	c_4	$c_3:$	t_1	t_4	t_3	t_2
$t_4:$	c_4	c_1	c_3	c_2	$c_4:$	t_2	t_1	t_4	t_3

We want to find a stable matching given this data. We begin with a bipartite graph with no edges, illustrated below, and then we will run the Gale-Shapley algorithm to determine a stable matching in this graph.



We will now consider the first few steps to help illustrate the Gale-Shapley algorithm and how it utilizes the preference options. We start by drawing a line between each TA with his prospective first choice. This gives us the following graph:



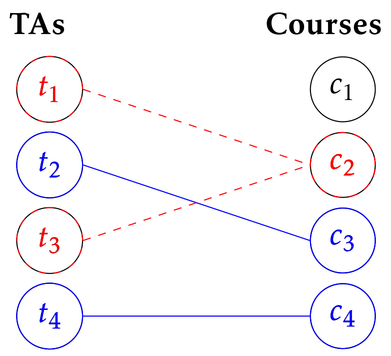
TA's Preferences

t_1	$\mathbf{c_2}$	c_4	c_1	c_3
t_2	$\mathbf{c_3}$	c_1	c_4	c_2
t_3	$\mathbf{c_2}$	c_3	c_1	c_4
t_4	$\mathbf{c_4}$	c_1	c_3	c_2

Course's Preferences

c_1	t_2	t_1	t_4	t_3
c_2	t_4	t_3	t_1	t_2
c_3	t_1	t_4	t_3	t_2
c_4	t_2	t_1	t_4	t_3

We notice that both t_1 and t_3 have c_2 as their first choice. When a situation like this occurs, where there exists more than one proposal to the same course, we consult to the preference list of the course that is proposed to (in this case c_2) to see which of the two TAs (t_1 and t_3) the department prefers more for that course. This process is done to establish an *engagement* or a *tentative matching* between a TA and course after going through a TA's preference list, meaning that there is the possibility that the matchings may change.



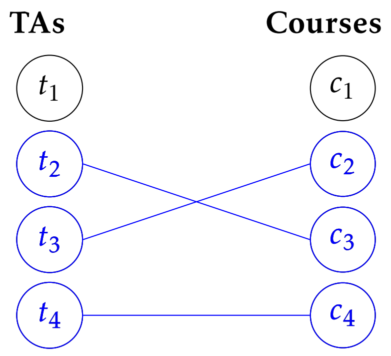
TA's Preferences

t_1	$\mathbf{c_2}$	c_4	c_1	c_3
t_2	$\mathbf{c_3}$	c_1	c_4	c_2
t_3	$\mathbf{c_2}$	c_3	c_1	c_4
t_4	$\mathbf{c_4}$	c_1	c_3	c_2

Course's Preferences

c_1	t_2	t_1	t_4	t_3
$\mathbf{c_2}$	t_4	$\mathbf{t_3}$	$\mathbf{t_1}$	t_2
c_3	t_1	t_4	t_3	t_2
c_4	t_2	t_1	t_4	t_3

Since the department prefers course c_2 for TA t_3 over t_1, it accepts the proposal from t_3. After the first round we have the following matches (where solid blue lines denote established engagements):



TA's Preferences

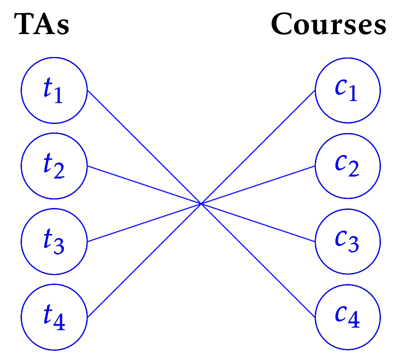
t_1	$\cancel{c_2}$	$\mathbf{c_4}$	c_1	c_3
t_2	$\mathbf{c_3}$	c_1	c_4	c_2
t_3	$\mathbf{c_2}$	c_3	c_1	c_4
t_4	$\mathbf{c_4}$	c_1	c_3	c_2

Course's Preferences

c_1	t_2	t_1	t_4	t_3
c_2	t_4	$\mathbf{t_3}$	t_1	t_2
c_3	t_1	t_4	t_3	$\mathbf{t_2}$
c_4	t_2	$\mathbf{t_1}$	$\mathbf{t_4}$	t_3

Since TA t_1 was rejected, we go to his next highest choice (denoted by the dashed red line, c_4) and see that that choice conflicts with the current engagement between t_4 and c_4. This brings us again to the situation where we must consider the departments preference for the course with regards to both TAs.

If we continue in this manner, we find a *stable* matching between these sets of TAs and courses does exist, as seen below in blue.



This is essentially the Gale-Shapley algorithm. We begin by going through the preference lists of each of the proposers and attempt to match them to courses they prefer, dealing with conflicting preferences by consulting the departmental preferences of the multi-matched courses.

There are three particular theorems (see [8]) that provide both a motivation and reasoning as to: (1) why this algorithm always provides a stable engagement, (2) why the stable solution is unique, and, (3) why the solution we get depends on which gender does the proposing.

Proposition 1.1. For any given SMP, the Gale-Shapley algorithm terminates and the ending engagements are stable.

Proof. First we note that since the number of men is equal to the number of women, everyone will eventually become engaged. If a man is not engaged, then there is at least one single woman to whom he can propose. If a woman is not engaged, there is at least one single man who will propose to her. Therefore, the Gale-Shapley algorithm will terminate. At the terminus, a matching is made. Assume that this matching is not stable. Then there are pairs that would be willing to *cheat* or rather, swap partners. But, then the pairings willing to cheat would have been matched as a result of the algorithm. But this is a contradiction since those pairings are not currently matched. Therefore, the marriage the Gale-Shapley algorithm output is stable. □

Proposition 2.1. With males proposing to females, every ordering of the male proposals results in the same stable solution. There is not a stable solution that each man would be happier with.

Proof. Suppose there is a stable matching A, and for contradiction, another stable matching A'. Say that a particular man x was matched to a woman y in the matching A, but actually prefers woman y'. So it follows that x must have been rejected by y' in A. We suppose without loss of generality that this was the first time in A that a woman rejected a stable partner, and that the rejection happened because y' got engaged to x' since she preferred him over x. Then x' can have no stable partner that he prefers to y', since no woman had previously rejected a stable partner. So x' must prefer y' to whatever partner he has in A', and the stable matching A' cannot occur since x' is paired with y'. Therefore, each man is paired with their favorite stable partner, that is, their most preferred partner they can have without anyone wanting to cheat. From the previous theorem, the output is the same stable set regardless of the order of proposals. □

Notice we mention in the above proof that the men get their favorite stable partners in the sense that each man is paired to their most preferred partner in a way that cheating would not be beneficial. This does not necessarily hold for the women. For the remainder of the paper, we will refer to this as the *man-optimal* version since the men are the ones proposing. If the women are proposing, then we will refer to this version as *woman-optimal*. Unfortunately one can prove:

Proposition 3.1. In the man-optimal stable matching, each woman is paired with the worst partner she is willing to accept in a stable matching.

Throughout this section, we have assumed numerous implicit conditions on our data. For instance, we have been working with the idea that the number of men and women are the same, that each person has strict complete preferences and a full ranking of who they favor, and that every pair is possible. But, what if these assumptions are not valid? If we consider any particular real-life application of this algorithm, we often find ourselves with two sets that are not of equal size. The TA-course assignment problem is exactly one such application, as the number of sections is often larger than the number of available TAs, and TAs are often required to teach more than just one section.

In the next section we discuss how to apply the Gale-Shapley algorithm to our TA matching problem in order to resolve these issues.

3. The TA problem as a Stable Marriage Problem

When approaching the problem of assigning teaching assistants to classes, we used the ideas illustrated in Section 2. We can simply think of the problem as a matching problem in a bipartite graph.

In the usual formulation of the Gale-Shapley algorithm, the sets of males X and females Y form a bipartite graph. More specifically, the bipartite graph is *complete*, meaning that for all $x \in X$ and $y \in Y$, the edge (x, y) is in the graph. In the context of our problem, this means that every man can be paired with every woman and vice versa.

For our problem, let X be the set of TAs and Y be the set of sections that need to be taught. Then these sets form a bipartite graph, but it is not complete. For example, a given TA x may be unable to teach a section y due to a time conflict or lack of necessary experience and/or seniority, in which case the edge between x and y is not present in the graph.

In this case, it may be impossible to match every TA to a section or vice versa; consider the simple example where two TAs x and x' only have an edge with section y. However, such a matching is still stable, as the Gale-Shapley algorithm is guaranteed to produce a stable matching for any bipartite graph (see [3]). The idea is that unmatched vertices are too "picky" or "undesirable" and that no vertex would rather have an edge with them.

Another assumption that the Gale-Shapley algorithm makes is that the matching from X to Y should be bijective; that is, every x needs to be paired with exactly one y and vice versa. Of course, this requires X and Y to have the same number of elements. As such, two problems arise when attempting to model our real-world problem: first, there are several TAs that need to teach more than one section; second, the number of TAs will very seldom (if ever) be equal to the number of sections. In our experience, there is most often a shortage of teaching assistants.

We solve these problems by adding in what we call “duplicate TAs”, “phantom TAs”, and “phantom sections.” Adding multiple copies of a TA to X allows us to simulate assigning a TA to several sections at once. Adding one or more “phantom TAs” allows us to even out the sizes of X and Y in the event that $|X| < |Y|$. When a section is assigned to a phantom TA, we know that we need to hire another TA to teach it. Conversely, adding one or more “phantom sections” allows us to even out the sizes of X and Y in the event that $|X| > |Y|$. If a TA is assigned to a phantom section, we know we have too many TAs and we need to cut down on some of their responsibilities. Preferences of both phantom TAs and phantom sections are assigned such that they are only paired with their respective sections and TAs in the event that a TA or section has nothing else to pair to.

These modifications allow us to run our model through the Gale-Shapley algorithm without changing the algorithm. This means that we can get a stable matching of TAs and sections in a fraction of a second. In fact, the Gale-Shapley algorithm runs in polynomial time, bounded by $O(p^2)$ for p the number of TAs (or the number of sections, if that happens to be larger) (see [10]).

Unfortunately, modeling our problem using a bipartite graph keeps us from expressing some of the key features of TA happiness. In particular, TAs like teaching sections from the same course and they generally prefer to teach sections that are close to each other temporally and spatially. This means that even if TA x prefers section y over sections z and z', but z and z' are in the same course, x would likely rather be assigned to z and z' than y and z. These are preferences which cannot be expressed by edges on a bipartite graph, which means the Gale-Shapley algorithm cannot account for them.

We also considered another combinatorial algorithm, the Hungarian algorithm, which offers a bound of $O(p^2q)$, where p is the number of TAs and q is the number of sections (see [10]). This algorithm suffers from the same problems as the Gale-Shapley algorithm because it uses a weighted bipartite graph as input.

Though it has no guarantees on complexity, integer linear programming allows us to develop a more expressive and robust model of TA happiness. Therefore, after revisiting the history of integer linear programs and assignment problem, we reformulate our problem as an integer program.

4. Integer Linear Optimization for Assignment Models

We now turn our attention to integer optimization, and its application to the assignment problem. Alexander Schrijver offers a comprehensive history of these topics [12], which we will summarize below.

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 [11] 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 transportation problem was fundamental to linear programming. In 1947, the simplex method was discovered by George Dantzig [9], allowing linear programs to be solved for an optimal solution by examining a k-dimensional triangle or tetrahedron. This method is widely used today, and has been implemented into software that can be used by the public. Such software is at the heart of our integer programming model, as we will describe below.

Before we discuss our model in detail, we will take a closer look into the computational technique we use to solve it. This is called integer linear programming.

Simply put, integer linear programming is a method to solve optimization problems, that range from the allocation of resources to production planning, that are described by linear constraints with non-negative integer valued variables.

A successful integer linear program will yield a solution that maximizes or minimizes some linear function (for a gentle introduction to this topic see [9]). E.g., the maximization of goods produced by machines in a meat packing company, or the minimization of cost from assigning product delivery routes. Integer linear programming is typically solved by solving a sequence of easier problems, namely *linear programs*, where the variables are no longer required to be integral. These easier problems are called after fixing the values of some variables to integral values. If the linear program returns an integer value that is often the optimal solution, otherwise we need to branch to fix different variables. This is organized in the branch-and-bound process. We give a few more details of the process below.

The most popular method to solve linear programs is the simplex method. The set of possible solutions of a linear program is a convex polyhedron described by a system of linear inequalities. Suppose we have a system of linear inequalities. If this system is two dimensional, the result of plotting this system will be a two dimensional shaded region, which we call a 2-polytope (polytope of dimension two, also known as a polygon). Examples such as these are easy to solve, however, linear systems in higher dimensions prove more difficult. Note that these linear inequalities are formed from the constraints applied our objective function, or what we want to maximize/minimize.

To give a formal definition of a polytope, several definitions are required. We first define a hyperplane as a set $\{x \in \mathbb{R}^n : A \cdot x = \alpha\}$, for any given $A \in \mathbb{R}^n$, $A \neq 0$ and $\alpha \in \mathbb{R}$, and where $\cdot$ is the dot product. For $n = 2$, this is simply a line, and for $n = 3$, this takes the form of a plane. We can think of an n-dimensional hyperplane as dividing $\mathbb{R}^n$ into two parts, which we call half-spaces. For instance, in dimension two the x-axis is a hyperplane which divides $\mathbb{R}^2$ into two half-spaces, one above the x-axis and one below it. Formally, we define a half-space as a set $\{x \in \mathbb{R}^n : A \cdot x \leq \alpha\}$ for any given $A \in \mathbb{R}^n$ and $\alpha \in \mathbb{R}$. Now we can define a polyhedron as a finite intersection of half-spaces, and a polytope as simply a bounded polyhedron. The constraints of a linear program can each be realized as half-spaces, and so their intersection forms a polytope. All feasible solutions are encapsulated by the polytope, and the simplex method provides an algorithm to examine such polytopes to find an optimal solution [5]. The simplex method methodically examines the value at each vertex to find the value that yields the highest objective value for the objective function we are trying to maximize. This is done by moving from vertex to vertex of the polytope, and checking each adjacent vertex. If every neighboring vertex decreases the objective value or does not increase the objective value, the process finishes and you have attained an optimal solution. The objective function will attain a maximum at some face of the polytope because polytopes are convex and the objective function is linear. Note that a polytope 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.

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 he should buy to raise on his farm. He wants to buy cows and chickens, and would like to purchase as many total animals as he can with the money he 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 his coop will hold no more than 20 chickens.

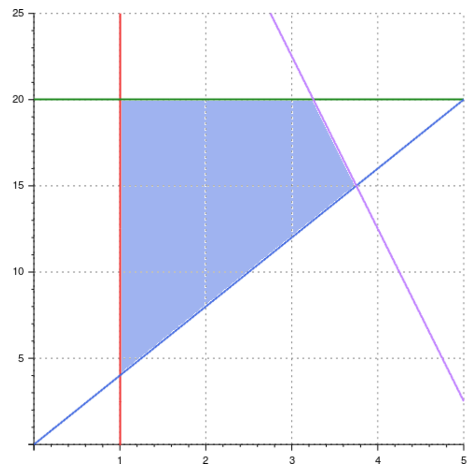
We can visualize solutions to this farmer's dilemma as points (x, y) in the Cartesian plane, where x is the number of cows and y the number of chickens he will purchase. The farmer wants as many animals as possible, which means he is trying to maximize the objective function

$$x + y.$$

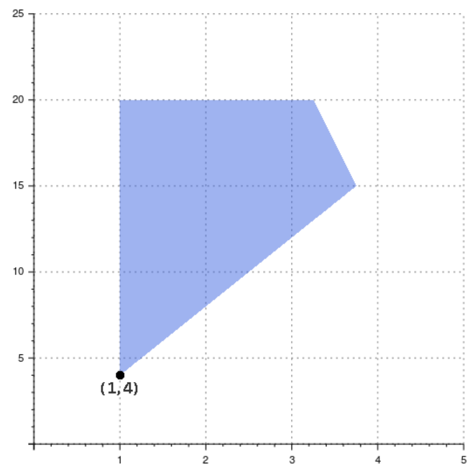
The farmer's financial and practical constraints can be represented as the following inequalities:

$$\begin{gathered} 200x + 20y \leq 1050, \\ x \geq 1, \\ y \geq 4x, \\ y \leq 20. \end{gathered}$$

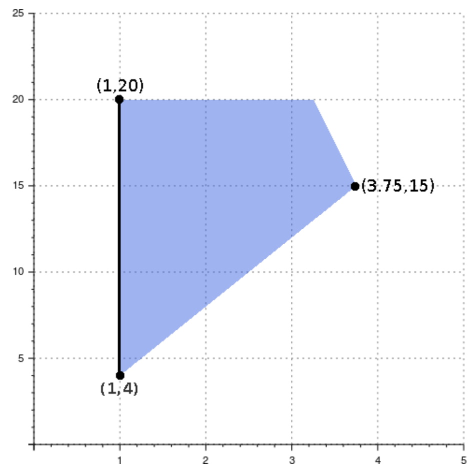
The first constraint captures the cost constraint, the second constraint ensures at least one chicken, the third constraint ensures at least four times as many chickens as cows, and the final constraint ensures that the coop will hold no more than 20 chickens. We graph the constraints below:



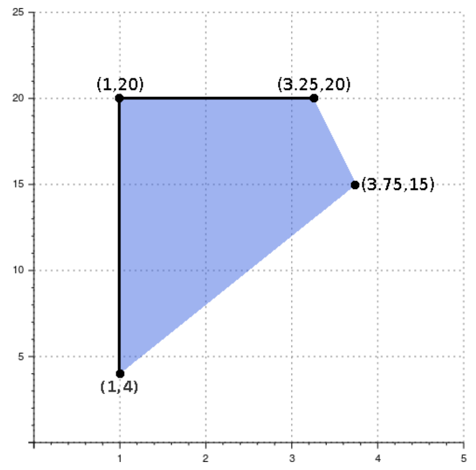
Notice that these lines cut out a polygon, which is a 2-dimensional polytope. Every point inside the polygon is a valid solution to the farmer's problem. The question that remains is which of these points is the optimal solution. The answer can be found using the simplex method. As we said before, the central idea behind the simplex method is that optimal solutions are found at the faces of polytopes. Thus, we can find the optimal solution by traveling between vertices until we cannot improve our objective value by traveling. In this our 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.



Now 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 $3.25 + 20$, so we move to that point.



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 basic 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.

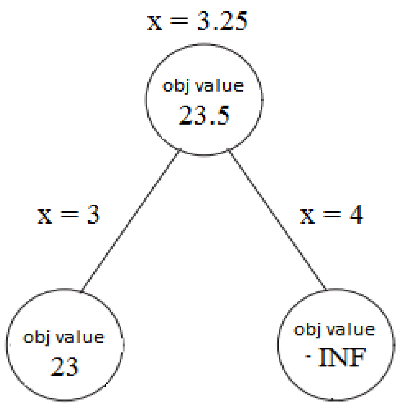
From the example above, we observe that the farmer should purchase 3.25 cows and 20 chickens. It is clear that this is impossible. We must find some way to extract a feasible integer solution. This can be done using the *branch and bound method*. The branch and bound method gives a solution within an optimal bound. Unlike integer programming, which provides the best possible integer solution given a set of linear constraints, the branch and bound algorithm provides a solution that is “reasonably close” to an optimal solution, and can also conveniently be applied to solve integer programming problems. First developed by Alisa Land and Alison Doig in 1960 [14], the branch and bound method is a divide and conquer method that can be used to solve some integer programs in polynomial time. 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 approach an optimal integer solution using real-valued linear programs.

To describe this process in detail, we would first break the LP 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 does, 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 following system of linear constraints:

$$\begin{gathered} 200x + 20y \leq 1050 \\ x \geq 1 \\ y \geq 4x \\ y \leq 20 \end{gathered}$$

and get the solution $x = 3.25$ cows, $y = 20$ chickens, and an objective function value of $x + y$ is 23.25.
Seeing this as a non-integer solution, we now apply branch and bound. Bounding our solution, we will now branch into two subcases: $x = 3, y = 20$ and $x = 4, y = 20$.
With $x = 3$ we apply substitute into the objective function to get a value of 23. This value is feasible, and we hold it as our current optimal integer solution while we test other cases.
Likewise with $x = 4$ we substitute to get an objective function value of 24. We test x and y and find that these values do not adhere to the constraints (the total cost is \$1200, which is more than \$1050), and therefore cannot be an optimal solution.



The branching stops at $x = 4$, as this gives an infeasible solution.
We have thus used the branch and bound method to "relax" the problem such that it gives us an integer solution $(3, 20)$. It should be noted that in larger examples where it is not feasible to explore every branch of the tree that the branch and bound algorithm will try and avoid searches that are "useless." For example, consider that the algorithm can prove that setting $x = 0$ for a sufficiently large binary-valued tree will lead to an optimal

value n, and setting $x = 1$ will lead to an optimal value of at least m. There is no reason to consider setting $x = 0$ if $n < m$, so the algorithm is able to avoid searching the space where $x = 0$.
Both the branch and bound and the simplex methods are tools that will allow us to solve integer programs for an optimal solution. We will now discuss integer programming and how we implemented it into our model.

5. A Happy Marriage vs a Stable Marriage: Modeling Happiness in TA Assignments

A natural application of integer programming is the assignment problem. We revisit our marriage example that we used to explain the Gale-Shapley method, 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

- $\forall i \in X, \ \sum_{j \in Y} x_{ij} \leq 1,$
- $\forall j \in Y, \ \sum_{i \in X} x_{ij} \leq 1.$
- $x_{ij} \geq 0, \ \forall i \in X$ and $j \in Y$

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 usually 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.

6. A Happy Marriage vs a Stable Marriage: Modeling Happiness in TA Assignments

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, as opposed to the "marriage language" of earlier sections. First let T be 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.

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 Katherine, Luis, Margaret, and Nicholas and our courses Calculus, Differential Equations, Ergodic Theory, and Functional Analysis. 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.

Katherine	**Luis**	**Margaret**	**Nicholas**
$\star$ Differential Equations	$\star$ Ergodic Theory	$\star$ Differential Equations	$\star$ Functional Analysis
$\star$ Functional Analysis	$\diamond$ Differential Equations	$\diamond$ Functional Analysis	$\star$ Ergodic Theory
$\diamond$ Ergodic Theory	$\otimes$ Functional Analysis	$\diamond$ Calculus	$\star$ Differential Equations
$\otimes$ Calculus	$\otimes$ Calculus	$\otimes$Ergodic Theory	$\otimes$ Calculus

Now we give each TA-course pair a numerical value γ_{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, Katherine will give 100 to Differential Equations, 50 to Ergodic Theory, and 0 to Calculus. She gives 75 to Functional Analysis because that is the halfway point between 100 and 50. On the other hand, Nicholas gives Ergodic Theory a 83.3 and Differential Equations 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, $\Phi_{K,D} = 1$ and $\phi_{K,F} = 1$. Then to account for indifference and normalizing, we let L_i be the set of courses liked by i, D_i the set of courses disliked by i, and then formally define γ_{ik} as follows:

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

This looks unnecessarily complicated, but is just a method to convert likes/dislikes into normalized numerical values. Since every section belongs to a course, the course preference γ_{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.

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 $\star$ denoting preferred times to teach, $\otimes$ denoting disliked times to teach, and $\diamond$ denoting indifference, we present the time preferences of our above TAs:

Katherine	**Luis**	**Margaret**	**Nicholas**
$\star$ Afternoon	$\star$ Morning	$\star$ Afternoon	$\star$ Evening
$\star$ Morning	$\diamond$ Evening	$\diamond$ Evening	$\star$ Afternoon
$\diamond$ Evening	$\otimes$ Afternoon	$\otimes$ Morning	$\otimes$ 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 ρ_{it} as follows:

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

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

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 γ_{ik} and ρ_{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:

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

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 $p_{aj} = \frac{2\cdot 100+50}{3} = 83.3$. 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.

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 teaches, the less time that 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.

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. As 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.

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 $v_{jj'}$ 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.

6.1. **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. This differs from the previously proposed Gale-Shapley assignment model, which simply attempted to give a ranking that was considered “stable”. To see in detail how this “fairness” factor was implemented, we will examine the objective function.

6.2. **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:

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

6.3. **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:
  $$\forall(i,j) \in (L_T \cup U_T) \times G_S, x_{ij} = 0.$$

6.4. **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.

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 $|T| \cdot (|S| + |C| + |D| + |S|^2)$ variables. In an average run, our program has about 20,000 variables.

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.
- $|T| \cdot |S|^2$ constraints ensure that no TA is assigned classes that occur at the same time.
- $|L_T| \cdot (|U_S| + |G_S|)$ constraints ensure no TA unfit to teach Upper division and graduate level courses does so. Bound this above by $|T| \cdot |S|$.
- $|U_T| \cdot |G_S|$ constraints ensure no TA unfit to teach graduate courses does so. This can also be bounded above by $|T| \cdot |S|$.
- 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 $|T|+|S|+|T|\cdot|S|^2+|T|\cdot|S|+|T|\cdot|S|+Q$. In our experience, an average run contains about 500,000 constraints.

To help illustrate the size and scope of the problem, there were approximately 80 TAs being assigned to 2-3 sections each per quarter.

7. 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. The system was ultimately implemented and employed by both the UC Davis Mathematics and Chemistry departments.

To solve the integer program we have explained thus far, we utilized the integer program solver SCIP (Solving Constraint Integer Programs) (see [4] 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 ([15] 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.

References

[1] picture of david gale. Accessed: 2018-02-24.

[2] picture of lloyd shapley. Accessed: 2018-02-24.

[3] H. Abeledo and G. Isaak. A characterization of graphs that ensure the existence of stable matchings. *Mathematical Social Sciences*, 22(1):93–96, 1991.

[4] T. Achterberg. SCIP: Solving constraint integer programs. *Mathematical Programming Computation*, 1(1):1–41, July 2009. `http://mpc.zib.de/index.php/MPC/article/view/4`.

[5] R.G. Bland. The allocation of resources by linear programming. *Scientific American*, 244:126–144, 1981.

[6] O.M. Carducci. The mathematics behind lifesaving kidney exchange programs. *Math Horizons*, 18(1):26–29, September 2010.

[7] D. Gale and L.S. Shapley. College admissions and the stability of marriage. *Amer. Math. Monthly*, 69:9–15, 1962.

[8] D. Gusfield and R.W. Irving. *The Stable Marriage Problem: Structure and Algorithms*. MIT Press, 1989.

[9] J. Matoušek and B. Gärtner. *Understanding and Using Linear Programming*. Universitext. Springer Berlin Heidelberg, 2007.

[10] M.D. Plummer and L. Lovász. *Matching Theory*. Elsevier, 1986.

[11] L. Rüschendorf. Monge-Kantorovich transportation problem and optimal couplings. *Jahresbericht der DMV*, 3:113–137, 2007.

[12] A. Schrijver. On the history of combinatorial optimization (till 1960). In *Handbooks in Operations Research and Management*. Elsevier, 2005.

[13] 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.

[14] R.J. Vanderbei. *Linear Programming: Foundations and Extensions*. International Series in Operations Research & Management Science (Book 114). Springer, 1996.

[15] R. Wunderling. *Paralleler und Objektorientierter Simplex-Algorithmus*. PhD thesis, Technische Universität Berlin, 1996. `http://www.zib.de/Publications/abstracts/TR-96-09/`.

Student biographies

Samuel Asher: graduated from the University of California, Davis in 2016 with a B.S. in Mathematics and a B.S. in Computer Science. He is currently working at Google as a Technical Writer.

Trevor Chan: (*Corresponding author:* tchchan@ucdavis.edu) graduated from the University of California, David in 2017 with a B.S. in Applied Mathematics and Computer Science. After graduating, Trevor continued to pursue graduate studies in Computer Science at UC David in the Genome and Biomedical Sciences Lab.