Assigning Teaching Assistants to Courses: Mathematical Models

By

VICTOR KEESEY FUENTES

SENIOR THESIS

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

BACHELOR OF SCIENCE

in

APPLIED MATHEMATICS

in the

COLLEGE OF LETTERS AND SCIENCE

of the

UNIVERSITY OF CALIFORNIA,

DAVIS

Approved:

Jesús A. De Loera

June 2014

ABSTRACT. In this senior thesis, we discuss an implementation of three models to solve a specific application of the Stable Marriage Problem, namely matching graduate student teaching assistants to discussion sections. We consider 3 different models, one that utilizes the Hungarian Algorithm, one that uses a combination of graph theory and a modified formulation of the Gale-Shapley algorithm, and a third model that is implemented as an integer program. Various constraints and restrictions are considered in each model, either tying into the likelihood of a particular matching or preventing a matching entirely. The criterion considered for these constraints are factors such as a graduate students time schedule, their seniority (as identified by the department), and the need for the department to potentially match a single graduate student to multiple discussion sections/courses.

Contents

Chapter 1. Introduction 1
1.1. Premise and Motivation 1
1.2. Overview of Implementations 1

Chapter 2. Hungarian Algorithm: Linear Programming View 3
2.1. Definitions and Concepts 3
2.2. General Algorithm 3
2.3. Example 6
2.4. Conclusion 8

Chapter 3. Gale-Shapley Algorithm 11
3.1. Overview of Concept 11
3.2. Definitions 11
3.3. General Algorithm 12
3.4. Theory 12
3.5. Example 14

Chapter 4. Integer Program Model 19
4.1. Definitions and Concepts 19
4.2. The Basic Theory of Integer Programming 19
4.3. Example 21
4.4. Implementation 23

Appendix A. Selected Pieces of Code in both Python and Zimpl 27
A.1. Setup 27
A.2. Running the Code 28
A.3. Understanding the Output 29
A.4. Integer Program Implementation 30

Appendix. Bibliography 35

CHAPTER 1

Introduction

Every quarter the Mathematics Department at the University of California, Davis must go through the process of assigning each graduate student to one (or possibly two) of the mathematics courses provided at the time. This process has been carried out by first emailing all graduate students (teaching assistants) to find out what classes they would prefer to teach and then the Student Affairs Officer matches the graduate students to the available classes via heuristics. This assignment alone takes upwards of 8 hours to complete and generally fails to reach a solution in which both the department and graduate students find acceptable.

1.1. Premise and Motivation

Our goal was 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 wanted an optimal matching - a matching that would be most preferable for both parties involved (the department and the TAs) either by maximizing or minimizing an objective function. The Hungarian Algorithm was considered as the initial model, as it guaranteed optimality of the matching, although it was found that this criteria was not sufficient for our purposes. This led to our second model, the Gale Shapley Algorithm, which not only guaranteed optimality, but introduced a notion of stability to the generated optimal solution. The third model, born out of a desire to compare our modified Gale Shapley Algorithm (GSA), was a Binary Integer Program (BIP) implementation, as it is possible to take into account all of the intricacies in the problem (i.e., which courses are available to which individual, how many courses each individual has to teach, possible time conflicts, etc) via the construction of additional constraints to the general BIP formulation. In total, these three implementations provide varied approaches to generating solutions to similar problems, that is generating a matching constrained by a collection of specified conditions.

1.2. Overview of Implementations

The Hungarian Algorithm, named after two mathematicians J. Egeváry and D. Knig, is a special case of the Primal-Dual Algorithm in which it takes a bipartite graph and produces a maximal matching.

One option that is considered is the Gale-Shapley Algorithm, an algorithm developed in 1962 by mathematicians D. Gale and L.S. Shapley [**2**] to solve the Stable Marriage Problem in polynomial time. It requires equal-sized sets of men and women as the input, each with their own strictly-ordered and complete preference lists. As an iterative process, each member of the proposing set proposes to his or her most preferred partner. The proposees accept their respective offers unless their current fiancé is more preferred than the current proposer. The algorithm continues until the last member of the proposee set is engaged.

Another option under consideration is an integer program approach, where we introduce an objective function (which relates to the total satisfaction of the faculty members) and construct a collection of constraints. Such constraints that are taken into account: the number of courses a particular faculty member must teach, ensure all courses have a professor, ensure that each professor receives at least one course they like (which is done by extensive weighting of every course throughout the entire year), that graduate courses are taught by full-time faculty, that faculty do not teach small lower division courses (or that large courses are split up between faculty), and take into account individual requests (i.e., preferences not to work particular quarters, specific times to teach, specific courses, etc). As a binary integer program, an acceptable assignment between faculty member and course (i.e., a matching that meets all the constraints) is denoted by a 1, whereas a 0 denotes an assignment that either is not allowed due to some constraint being violated or because a better assignment has already been made.

With the Hungarian Algorithm acting as a starting point for our investigation, we wish to build up our understanding of the Assignment Problem and the many ways it can be represented.

CHAPTER 2

Hungarian Algorithm: Linear Programming View

2.1. Definitions and Concepts

Here are a collection of definitions that will aid in constructing the proper context for the Hungarian Algorithm:

Definition. The *Assignment Problem* is a class of problems where m *agents* must be assigned to n possible *tasks* such that the sum of the costs incurred by the individual assignments is minimized.

Definition. An *undirected graph* (or simply a *graph*) G consists of a finite non-empty set of elements $V(G)$ called *points* and a multi-set of unordered pairs of points $E(G)$ called *lines*. [**7**]

Definition. If the point set of a graph G can be partitioned into two disjoint non-empty sets, $V(G) = A \bigcup T$, such that all lines of G join a point of A to a point of T, we call G *bipartite* and refer to $A \bigcup T$ as the *bipartition* of G. [**7**]

Definition. A *matching for a bipartite graph* $G = (\{A,T\},E)$ is a subset M of E such that no two elements of M have a common vertex.

Definition. If $G = (\{A,T\},E)$ is a bipartite graph, set $\rho(G) = \max\{\bar{M} \mid M$ is a matching of $G\}$. A matching M such that $\bar{M} = \rho(G)$ will be called a *maximal matching.*

Definition. A set of vertices V' is said to be a *cover of a set of edges* E' if every edge in E' is incident on one or more of the vertices in V'. A set of vertices S will be called a *cover* of the bipartite graph $G = (\{A,T\},E)$ if every edge of G is incident on one or more of the vertices of S.

2.2. General Algorithm

The assignment problem is one of the fundamental combinatorial optimization problems in the branch of optimization or operations research in mathematics. It consists of finding a maximum weight matching in a weighted bipartite graph. Such a formulation begins

with a complete bipartite graph between the sets A, the agents, and T, the tasks, as seen below.

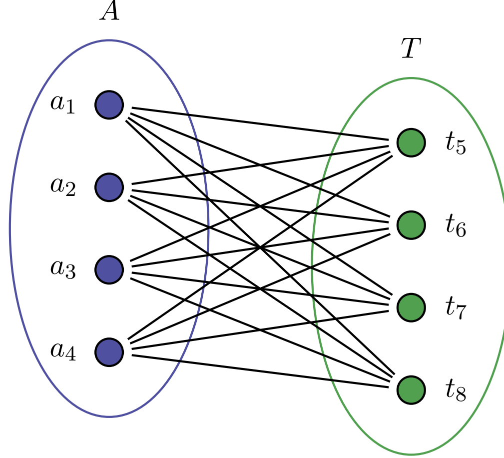


FIGURE 1. Complete bipartite graph

Each edge between the sets is provided a weight value, c, which represents the cost of assigning a particular node $a \in A$ to $t \in T$. Although not shown in the above complete bipartite graph, each c_{ij}, $i, j = 1, 2, 3, 4$, represents the appropriate cost of assigning agent a_i to task t_j. In cases where a particular agent a_{ij} cannot be assigned to a task t_{ij}, a cost $c_{ij} = 0$ is given, effectively removing that edge from the graph. It is required to perform all tasks by assigning exactly one agent to each task and exactly one task to each agent in such a way that the total cost of the assignment is minimized.

This can in turn be expressed as a standard linear program where, if given two sets A and T, of equal size, with a weight function $C : A \times T \to \mathbb{R}$, the following objective function, ζ is constructed:

$$\begin{aligned}
\text{minimize} \quad & \sum_{i \in A} \sum_{j \in T} C_{ij} x_{ij} \\
\text{subject to} \quad & \sum_{j \in T} x_{ij} = 1 \qquad , i \in A, \\
& \sum_{i \in A} x_{ij} = 1_i \qquad , j \in T, \\
& x_{ij} \geq 0 \qquad\qquad , i \in A, j \in T.
\end{aligned}$$

Where x_{ij} represents the assignment of agent i to task j, with value 1 if the assignment is made and 0 otherwise. The first constraint requires that every agent ($i \in A$) be assigned to a single task, and the second constraint requires that every task ($j \in T$) be assigned to

a single agent. The goal is to find a solution vector $x \in \mathbb{R}^n$ such that all the constraints are satisfied and the objective function ζ is minimized.

Definition. A solution to the linear program is called *optimal* if it both satisfies all of the constraints and attains the desired maximum (or minimum). [**8**]

With the goal being an optimal assignment of agents to tasks, the Hungarian Algorithm provides a solution to the problem in $O(n^4)$ time [**6**].

Hungarian Algorithm

Let A be an $n \times n$ cost matrix with $a_{ij} > 0$ representing the cost of assigning agent i to task j.

(1) For all $i \in \{1, 2, ..., n\}$ and a_{ij} such that $j \in \{1, 2, ..., n\}$, set $a_{ij} = a_{ij}$ - $min_j\{a_{ij}\}$. Similarly, for all $j \in \{1, 2, ..., n\}$ and a_{ij} such that $i \in \{1, 2, ..., n\}$, set $a_{ij} = a_{ij}$ - $min_i\{a_{ij}\}$. Let this new matrix be denoted as A'.
(2) 2. Find the number of lines, k, through both the rows and columns that "cover" all of the zeros of A'.
(3) If $k < n$, let a_0 be equal to the minimal element a_{ij} such that a_{ij} is not covered by any of the k lines. For all uncovered elements a_{ij}, let $a_{ij} = a_{ij}$ - a_0, whereas for all twice covered elements let $a_{ij} = a_{ij} + a_0$. With the revised matrix, repeat Step 2.
(4) When $k \geq n$, construct a set of n independent zeros, Γ, where for all a_{ij}, if $a_{ij} \in \Gamma$, $a_{ij} = 1$, else $a_{ij} = 0$. This assignment output, composed of the n independent zeros, is the new (and final) matrix.

Theorem. *Suppose that Step* (1) *or* (3) *in the assignment is implemented. Then the optimal solution of the assignment problem with the new cost matrix does not change. [**6**]*

Proof. For Step (1) note that the optimal solution x^* does not change if a constant is subtracted from the column. To see this suppose that c is subtracted from every element in the first row of the cost coefficient matrix. Then the problem becomes

$$\begin{aligned} \text{minimize} \quad & \sum_{i \geq 2, j} c_{ij} x_{ij} + \sum_j (c_{1j} - c) x_{1j} & \\ \text{subject to} \quad & \sum_{j \in T} x_{ij} = 1 & , i \in A, \\ & \sum_{i \in A} x_{ij} = 1 & , j \in T, \\ & x_{ij} \geq 0 & , i \in A, j \in T. \end{aligned}$$

But we note that

$$\sum_j (c_{1j} - c)x_{1j} + \sum_{i \geq 2,j} c_{ij}x_{ij} = \sum_{i,j} c_{ij}x_{ij} - \sum_j cx_{1j} = \sum_{i,j} c_{ij}x_{ij} - c.$$

So the optimal solution for the original problem is exactly the optimal solution for the perturbed problem. The same holds for all other rows and columns. So Step (1) of the algorithm does not change the optimal solution.

Now suppose c is added to each cost of a doubly covered entry and c is subtracted from each cost of an uncovered entry. This is equivalent to adding c to each covered column and subtracting c from each uncovered row. To see this, suppose c is added to the covered column 1 and subtracted from the uncovered row 1. Suppose column two is uncovered and row two is covered. Then we get the upper left 2×2 submatrix

	cov($+c$)	**uncov**(no action)
uncov($-c$)	$c - c = 0$	$-c$
cov(no action)	$+c$	0

which covers all four cases. So the action in Step (3) is a series of actions in Step (1) and the optimal solution does not change. □

2.3. Example

Here is an illustrative example of the Hungarian Algorithm, which walks through the above steps. We begin with the following:

$$A = \begin{bmatrix} 14 & 5 & 8 & 7 \\ 2 & 12 & 6 & 5 \\ 7 & 8 & 3 & 9 \\ 2 & 4 & 6 & 10 \end{bmatrix}$$

(1) For all $i \in \{1, 2, ..., 4\}$, the row minimum (RowMin) is calculated and each row a_{ij}, for $j = \{1, 2, ..., 4\}$ is reduced by its respective minimum.

$$\begin{bmatrix} & & & & \text{RowMin} \\ 14 & 5 & 8 & 7 & 5 \\ 2 & 12 & 6 & 5 & 2 \\ 7 & 8 & 3 & 9 & 3 \\ 2 & 4 & 6 & 10 & 2 \end{bmatrix}$$

and for $j \in \{1, 2, ..., 4\}$, the column minimum (ColMin) is also calculated and each column a_{ij}, for $i = \{1, 2, ..., 4\}$ is reduced by its respective minimum.

$$\begin{bmatrix} & 9 & 0 & 3 & 2 \\ & 0 & 10 & 4 & 3 \\ & 2 & 5 & 0 & 6 \\ & 0 & 2 & 4 & 8 \\ \text{ColMin} & 0 & 0 & 0 & 2 \end{bmatrix}$$

(2) A minimal covering of the zeros for the new matrix A' is found

$$\begin{bmatrix} \rightarrow & 9 & 0 & 3 & 0 \\ & 0 & 10 & 4 & 1 \\ \rightarrow & 4 & 5 & 0 & 4 \\ & 0 & 2 & 4 & 6 \\ & \uparrow & & & \end{bmatrix}$$

but with this cover $k = 3$ and $k < n$ $(n = 4)$, so we must continue to Step 3.

(3) The minimal non-covered element is $a_{24} = 1$, so our modified matrix is

$$\begin{bmatrix} \rightarrow & 10 & 0 & 3 & 0 \\ & 0 & 9 & 3 & 0 \\ \rightarrow & 5 & 5 & 0 & 4 \\ & 0 & 1 & 3 & 5 \\ & \uparrow & & & \uparrow \end{bmatrix}$$

which gives us the minimal number of covering lines, $k = 4$, as desired.

(4) To begin the construction of a set of zeros Γ, set $x_{33} = 1$ and delete the corresponding row and column.

$$\begin{bmatrix} & 10 & 0 & 3 & 0 \\ & 0 & 9 & 3 & 0 \\ \rightarrow & 5 & 5 & \underline{0} & 4 \\ & 0 & 1 & 3 & 5 \\ & & & \uparrow & \end{bmatrix}$$

This generates a 3×3 submatrix of A'

$$\begin{bmatrix} \rightarrow & 10 & 0 & 0 \\ \rightarrow & 0 & 9 & 0 \\ & \underline{0} & 1 & 5 \\ & \uparrow & & \end{bmatrix}$$

Set $x_{31} = 1$ (x_{41} in the original matrix A'). Again, this generates a smaller, 2×2 submatrix

$$\begin{bmatrix} \rightarrow & 0 & 0 \\ & 9 & \underline{0} \\ & & \uparrow \end{bmatrix}$$

Set $x_{22} = 1$ (x_{24} in the original A'. This forces the last assignment to be x_{12} in the original A'.

(5) Therefore with the original matrix we have the following assignment, with the appropriate zeros underlined

$$\begin{bmatrix} 10 & \underline{0} & 3 & 0 \\ 0 & 9 & 3 & \underline{0} \\ 5 & 5 & \underline{0} & 4 \\ \underline{0} & 1 & 3 & 5 \end{bmatrix}$$

which is equivalently expressed in the the assignment matrix

$$\begin{bmatrix} 0 & \underline{1} & 0 & 0 \\ 0 & 0 & 0 & \underline{1} \\ 0 & 0 & \underline{1} & 0 \\ \underline{1} & 0 & 0 & 0 \end{bmatrix}$$

2.4. Conclusion

The objective function considered for this system is $\zeta = \sum_i \sum_j A_{ij} x_{ij}$, with the desire to minimize the overall cost. This algorithm does just that with the following constraints:

$$\begin{aligned} \text{minimize} \quad & \sum_i \sum_j A_{ij} x_{ij} \\ \text{subject to} \quad & \sum_j x_{ij} = 1 \qquad , i \in \{1, 2, 3, 4\} \\ & \sum_i x_{ij} = 1 \qquad , j \in \{1, 2, 3, 4\} \\ & x_{ij} \geq 0 \qquad\quad , i, j \in \{1, 2, 3, 4\} \end{aligned}$$

.

As can be seen through the example from the previous section, this algorithm does indeed provide the optimal solution to this linear program. A natural extension to this problem would be to ask if, introducing some new concept of preference and instead maximizing upon that, would this change the solution to the system? That is to say, if a particular assignment had two such agents that prefer each others tasks (and maximizing overall satisfaction of each agents assignment was one of the criterion), then how would

this change the assignment found if just trying to minimize the overall cost of an assignment? This question takes us away from the linear program model and reintroduces the bipartite graph representation, along with a different algorithm, to handle this seemingly new question of finding an optimal solution.

CHAPTER 3

Gale-Shapley Algorithm

3.1. Overview of Concept

In the search for an optimal solution to the Assignment Problem, as seen in the Hungarian Algorithm in Chapter 2, there is also another criterion, stability, that serves a purpose in our search for a solution. A well-known example of the criterion, the Stable Marriage Problem, presents a situation in which there are two groups of equal size, one of men and the other women. Each individual in their respective group, in the desire for marriage, has a strict preference list of each potential spouse, from most to least desirable. Similar to the set up in Chapter 2, let us denote the two groups as sets of men, M, and women, W, and each individual in the set m and w, respectively.

3.2. Definitions

Definition. A *matching* for a bipartite graph $G = (\{M,W\},E)$ is a subset $\mathcal{M}$ of E such that no two elements of $\mathcal{M}$ have a common vertex. The set $\mathcal{M}$ consists of the two-tuples, where for each $m \in M$ and $w \in W$, $(m, w) \in \mathcal{M}$ denote a particular matching between the elements m and w.

Definition. A *stable matching* occurs when for all matchings (m, w) and (m', w') in an assignment set $\mathcal{M} \in E$, the following do not occur together:

(1) m prefers w' over his current partner w,
(2) w' prefers m over her current partner m'.

Definition. If man m and woman w are matched in $\mathcal{M}$, then m and w are called *partners* in $\mathcal{M}$. To specify the particular partners in matching $\mathcal{M}$, we write $m = p_{\mathcal{M}}(w)$ and $w = p_{\mathcal{M}}(m)$, where $p_{\mathcal{M}}(m)$ is the $\mathcal{M}$-partner of m, and $p_{\mathcal{M}}(w)$ is the $\mathcal{M}$-partner of w. [**3**]

Definition. A man m and a woman w are said to *block* a matching $\mathcal{M}$, or a *blocking pair* of $\mathcal{M}$, if m and w are not partners in $\mathcal{M}$, but m prefers w to $p_{\mathcal{M}}(m)$ and w prefers m to $p_{\mathcal{M}}(w)$. [**3**]

Definition. If there is at least one blocking pair for a matching $\mathcal{M}$, then this matching is considered to be *unstable*. [**3**]

The *Stable Marriage Problem* is a class of matching problems where each element of one set must be matched to exactly one element of another set of the same size such that the matching is stable.

3.3. General Algorithm

We start with the each of the men $m \in M$ proposing to one of the women $w \in W$, where each man and woman has a complete, strictly ordered preference list of the opposite set. Each man proposes to his top choice.

(1) If a woman does not receive a proposal, continue.
(2) If a woman receives only one proposal, she is engaged to the proposer.
(3) If a woman receives more than one proposal, she is engaged to the most preferable proposer.
(4) Rest are rejected.
(5) Each rejected man proposes to his next best choice.
(6) Repeat Steps 1 and 2 until each woman is engaged.

3.4. Theory

Here we introduce some of the theorems regarding the Gale-Shapley Algorithm that address the notion of stability (and its various forms) for the Stable Marriage Problem.

Theorem. *For any given Stable Marriage Problem, the Gale-Shapley algorithm terminates and the terminus engagements are stable.*

Proof. -First we show that no man can be rejected by all the women. A woman can reject only when she is engaged, and once she is engaged she never again becomes free. So the rejection of a man by the last woman on his list would imply that all the women were already engaged. But since there are equal numbers of men and women, and no man has two fiancées, all the men would also be engaged, which is a contradiction. Also, each iteration involves one proposal, and no man ever proposes twice to the same woman, so the total number of iterations cannot exceed n^2 time (for an instance involving n men and n women). Therefore the algorithm terminates. It is clear, at termination, the engaged pairs specify a matching, which we denote by $\mathcal{M}$. If man m prefers woman w to $p_{\mathcal{M}}$(m), which denotes the $\mathcal{M}$-partner of man m, then w must have rejected m at some point during the execution of the algorithm. But this rejection implies that w was, or became, engaged to a man she prefers to m, and any subsequent change of her fiancé brings her a still better partner. So w cannot prefer m to $p_{\mathcal{M}}$(w), and therefore (m, w) cannot block $\mathcal{M}$. It follows that there are no blocking pairs for $\mathcal{M}$, and therefore that $\mathcal{M}$ is a stable matching. [**3**] □

Theorem. *With males proposing to females, every ordering of the male proposals results in the same stable solution. There is no stable solution that each man would be happier in.*

PROOF. Suppose that an arbitrary execution E of the algorithm yields the matching $\mathcal{M}$, and that, in contradiction of the theorem, there is a matching $\mathcal{M}'$ and a man m such that m prefers w'=$p'_{\mathcal{M}}$(m) to w=$p_{\mathcal{M}}$(m). Then during E, w' must have rejected m. Suppose, without loss of generality, that this was the first occasion, during E, that a woman rejected a stable partner, and suppose that this rejection took place because of the engagement of w' to m' (so that w' prefers m' to m). Then m' can have no stable partner whom he prefers to w' (for no woman had previously rejected a stable partner). So m' prefers w' to his partner in $\mathcal{M}'$, and the supposed stable matching $\mathcal{M}'$ is blocked by (m', w'). Each man m is therefore matched in $\mathcal{M}$ with his favorite stable partner w, and since E was an arbitrary execution of the algorithm, it follows that all possible executions of the algorithm leads to this same stable matching. [**3**] □

Theorem. *In the man-optimal stable matching, each woman is paired with the worst partner she can have in a stable matching.*

PROOF. Suppose not. Let $\mathcal{M}_0$ be the man-optimal stable matching, and suppose there is a stable matching $\mathcal{M}'$ and a woman w such that w prefers m=$p_{\mathcal{M}_0}$(w) to m'=$p'_{\mathcal{M}}$(w). But then (m, w) blocks $\mathcal{M}'$ unless m prefers $p_{\mathcal{M}'}$(m) to w=$p_{\mathcal{M}_0}$(m), in contradiction of the fact that m has no stable partner better than his partner in $\mathcal{M}_0$. [**3**] □

Although the GS algorithm is effective on its own, there is a particular limitation, when the size of the sets of men and women are not equal, that calls for a very necessary extension of the algorithm. This brings us to the extended GS algorithm when taking into account sets of unequal size, which is defined as follows:

Let X and Y be the sets of men and women, respectively, and suppose that $|X| = n_x < n_y$ $= |Y|$. Then a matching M is unstable if there is a man m and a woman w such that:

(1) m and w are not partners in $\mathcal{M}$;
(2) m is either unmatched in $\mathcal{M}$, or prefers w to his partner in $\mathcal{M}$;
(3) w is either unmatched in $\mathcal{M}$, or prefers m to her partner in $\mathcal{M}$.

This variation on the concept of stability is necessary due to the fact that most systems are rarely ever equal in terms of the two sets (for example, in our particular context the number of TAs is rarely equal to the number of classes).

Let us consider a possible (and very much relevant) relaxation of the Gale-Shapley Algorithm, where each individuals preference list is no longer required to be strictly ordered. This relaxation in turn requires a necessary extension of the notion of stability. There are three such notions to consider:

(1) First Notion: Regard a matching as *unstable* if there is a man and a woman who are not partners, each of whom likes the other at least as well as his/her partner in the matching. If such a matching is stable under this criterion, the matching is

super-stable.
An example for which no super-stable matching exists would be when there is complete indifference, where no strict preference is expressed by anyone. [**3**]

(2) Second Notion: A more relaxed notion of stability would be to regard a matching as *unstable* if there exists a man and woman who are not partners, such that one strictly prefers the other to his/her partner and the other is at least indifferent. If such a matching is stable under this criterion, the matching is denoted as *strongly stable.*
An example in which no matching is strongly stable is to consider an instance of size 2 described by the following preference lists:

Men's Preferences			**Women's Preferences**		
m_1	w_1	w_2	w_1	m_2	m_1
m_2	(w_1)	(w_2)	w_2	m_2	m_1

where all preferences are strict except for those of m_2, which are denoted by parentheses around each woman in his list. Then the matching $\{(m_1,w_1),(m_2,w_2)\}$ is blocked by the pair (m_2,w_1) and the matching $\{(m_1,w_2),(m_2,w_1)\}$ is blocked by the pair (m_2,w_2). [**3**]

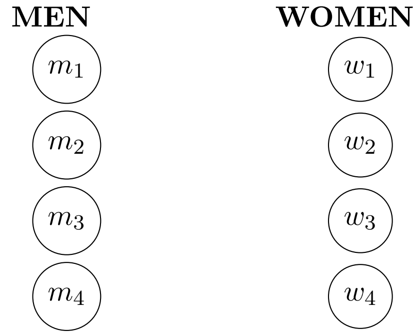
(3) Third Notion: A matching is *unstable* if there is a man and a woman who are not partners, each of whom strictly prefers the other to his/her partner. In this case, if an instance with strict preferences is created by breaking all ties arbitrarily, then any matching that is stable in that standard instance will also be stable in the original instance with ties. That is to say, for this weaker notion of stability, a stable matching can always be found by breaking ties arbitrarily and applying the Gale-Shapley algorithm. [**3**]

3.5. Example

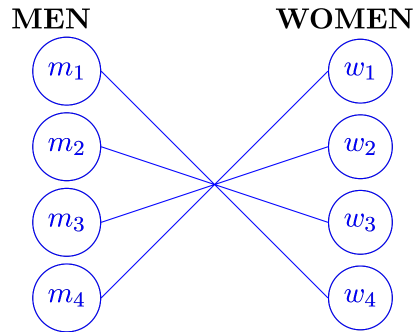
For example we can consider a particular collection of four men (denoted as m_1, m_2, m_3, m_4) and 4 women (denoted w_1, w_2, w_3, w_4) given their particular strict preference lists

Men's Preferences					**Women's Preferences**				
m_1	w_2	w_4	w_1	w_3	w_1	m_2	m_1	m_4	m_3
m_2	w_3	w_1	w_4	w_2	w_2	m_4	m_3	m_1	m_2
m_3	w_2	w_3	w_1	w_4	w_3	m_1	m_4	m_3	m_2
m_4	w_4	w_1	w_3	w_2	w_4	m_2	m_1	m_4	m_3

We start with a list both men and women's preferences, where the left most entry in the list is the most preferred match and the right most the least preferred match. This information can be depicted with a bipartite graph, that is two disjoint sets of nodes representing each of the men and each of the women, and we can utilize the Gale-Shapley algorithm along with the information. We then have the following graph:

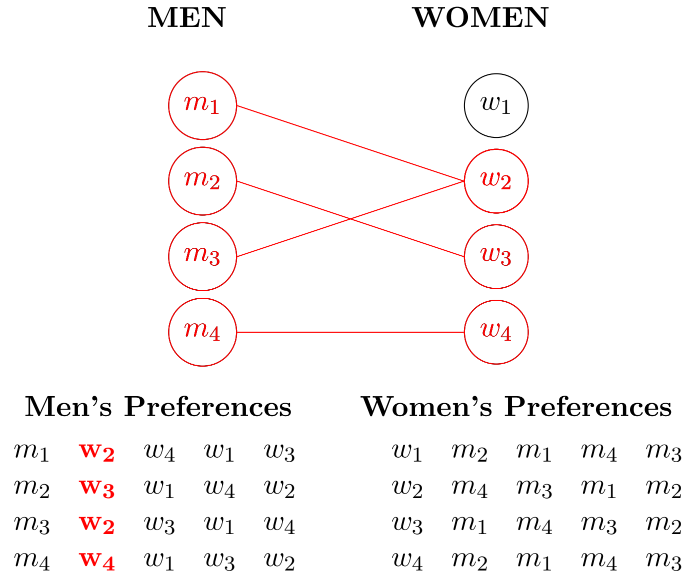


We find that the *stable* matching between these sets of men and women is as seen below in blue.



For sake of the example we will consider some of the first few steps to help illustrate the Gale-Shapley algorithm and how it utilizes the preference options.

1. We start with each man's first choice on their preference lists and draw a line between that man and their respective first choice to denote a proposal. This gives us the following graph:



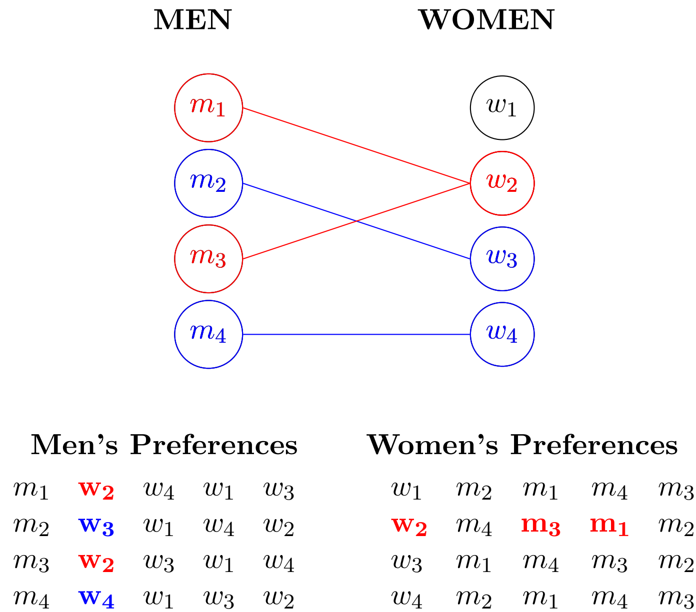
Men's Preferences

m_1	$\mathbf{w_2}$	w_4	w_1	w_3
m_2	$\mathbf{w_3}$	w_1	w_4	w_2
m_3	$\mathbf{w_2}$	w_3	w_1	w_4
m_4	$\mathbf{w_4}$	w_1	w_3	w_2

Women's Preferences

w_1	m_2	m_1	m_4	m_3
w_2	m_4	m_3	m_1	m_2
w_3	m_1	m_4	m_3	m_2
w_4	m_2	m_1	m_4	m_3

2. We notice that both m_1 and m_3 have w_2 as their first choice. When a situation like this occurs, where there exists more than one proposal to the same woman, we have to refer to the preference list of that particular woman (in this case w_2) to see which of the two men (m_1 and m_3) she prefers more. This process is done to establish an *engagement* or a *tentative matching* between a men and women after each round of going through each man's preference list, meaning that there is the possibility that the matching may change.



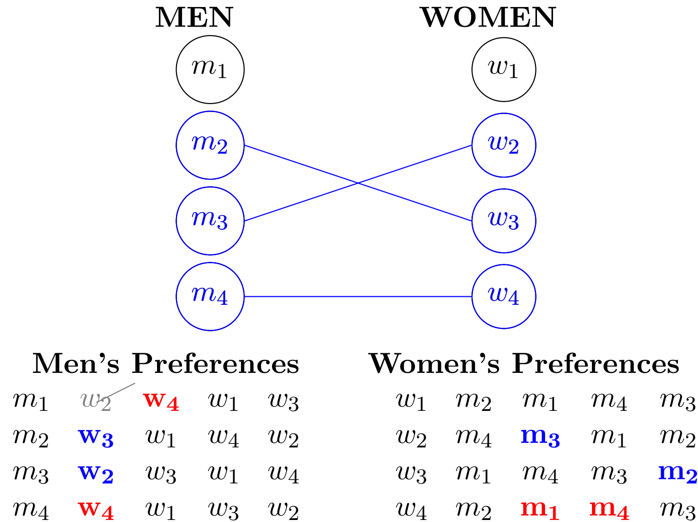
Men's Preferences

m_1	$\mathbf{w_2}$	w_4	w_1	w_3
m_2	$\mathbf{w_3}$	w_1	w_4	w_2
m_3	$\mathbf{w_2}$	w_3	w_1	w_4
m_4	$\mathbf{w_4}$	w_1	w_3	w_2

Women's Preferences

w_1	m_2	m_1	m_4	m_3
$\mathbf{w_2}$	m_4	$\mathbf{m_3}$	$\mathbf{m_1}$	m_2
w_3	m_1	m_4	m_3	m_2
w_4	m_2	m_1	m_4	m_3

3. Since women w_2 prefers man m_3 over m_1 she accepts the proposal from m_3 and after the first round we have the following matches (where blue denotes established engagements):



We see from the preference lists that since man m_1 was rejected, we go to his next highest choice (as denoted in red, w_4) and see that that choice conflicts with the current engagement between m_4 and w_4. This brings us again to the situation where we must consider the woman's preference with regards to both men. Since w_4 prefers m_1, a new tentative engagement is made between m_1 and w_4, leaving m_4 now without a partner. The last step would be to look at the preference list of m_4 and check his next highest choice, which is w_1. Since w_1 has not yet received a proposal, m_4 is engaged to w_1. Since all men and women are engaged, the algorithm terminates, leaving us with a stable matching between these particular sets of men and women.

This, in effect, is the Gale-Shapley algorithm, going through the preference lists of each of the proposers (traditionally the men, since this is reference to the Stable-Marriage Problem) and dealing with multiple men preferring the same woman by consulting that particular woman's preference list.

Although the Gale-Shapley Algorithm worked relatively well, even when modified via a "black list" to remove unsatisfactory potential matchings, there were some questions as to whether this implementation was the most effective one if a larger and/or more complex collection of constraints/restrictions were to be considered. This leads to the preliminary tests of an Integer Program Implementation of matching faculty members to courses, which will be discussed in greater detail in the following chapter.

CHAPTER 4

Integer Program Model

Now we are re-imagining this matching problem as an integer programming problem where our goal is to maximize the overall satisfaction given all matchings. Classes are still separated by time, but the entire set of classes can be (and is, in this case) partitioned into subsets such that there is a set of early morning classes, a set of late morning classes, a set of early afternoon classes, etc. Classes are also separated into sets containing lower division courses, upper division courses, and graduate level courses. Faculty members can also be separated in a similar manner, where different sets are constructed to denote which faculty members are cleared to teach which type of classes (for example, differentiating between full-time and part-time faculty members).

Furthermore, this set up allows for us to utilize constraints to dictate the number of units a faculty member must teach. A faculty's preferences can be taken into account by designating a weight (ranging from 100, which means the faculty member loves that course, to -100, which means the faculty member loathes that course). Any special conditions can be taken into account as an extra constraint for the problem, which includes time conflicts such as sabbaticals or particular times during the day.

4.1. Definitions and Concepts

Definition. The *simplex algorithm for linear programming* (LP) allows for the maximization or minimization of a linear function of several variables, subject to a collection of linear constraints. [**9**]

Definition. An *integer program* (IP), that is, an optimization or feasibility program in which the variables are restricted to be integers and can be further restricted to a binary case (denoted as BIP), with variables either 0 or 1. [**8**]

4.2. The Basic Theory of Integer Programming

Before jumping into the binary variant of an integer program, which is the desired model for the matching problem we are dealing with, let us motivate the concepts behind the Linear Programing (LP) Problem. With the goal being optimization, let us define *decision variables* x_j, $j = 1, 2, ..., n$ as variables whose values are to be decided in an optimal

fashion and a linear function of these variables, whose value we may want to maximize or minimize, the *objective function* denoted by $\zeta = c_1x_1 + c_2x_2 + ... + c_nx_n$. Furthermore any situation will introduce additional constraints on these decision variables, with these constraints consist of an equality or inequality associated with some linear combination of the decision variables:

$$a_1x_1 + a_2x_2 + ... + a_nx_n \left\{ \begin{array}{c} \leq \\ = \\ \geq \end{array} \right\} b.$$

Note that these constraints can be converted from inequalities to equalities with the introduction of a nonnegative variable w called a *slack variable*. There we go from an inequality such as

$$a_1x_1 + a_2x_2 + ... + a_nx_n \leq b$$

and convert it to an equality constraint with the addition of w:

$$a_1x_1 + a_2x_2 + ... + a_nx_n + w = b.$$

On the other hand, an equality constraint can be decomposed into two inequality constraints:

$$\begin{aligned} a_1x_1 + a_2x_2 + ... + a_nx_n &\leq b, \\ a_1x_1 + a_2x_2 + ... + a_nx_n &\geq b. \end{aligned}$$

The preferred presentation is pose the inequalities as less-thans and stipulate that all of the decision variables are nonnegative. Therefore the linear programming problem can be formulated as follows:

$$\begin{aligned} \text{maximize } & c_1x_1 + c_2x_2 + ... + c_nx_n \\ \text{subject to } & a_{11}x_1 + a_{12}x_2 + ... + a_{1n}x_n \leq b_1, \\ & a_{21}x_1 + a_{22}x_2 + ... + a_{2n}x_n \leq b_2, \\ & \vdots \\ & a_{m1}x_1 + a_{m2}x_2 + ... + a_{mn}x_n \leq b_m, \\ & x_1, x_2, ..., x_n \geq 0. \end{aligned}$$

A proposal for specific values of the decision variables is called a *solution*. With a LP there various types of solutions. Let us define a solution as *feasible* if the solution (x_1, x_2, ... , x_n) satisfies all constraints. The solution is optimal if the solution also attains the desired maximum. Some problems are just infeasible, so let us define a solution as *infeasible* if there exists no feasible solution. On the other end of the spectrum, there may be problems where there exist feasible solutions, but no upper bound on the value the objective function takes. These problems are called *unbounded*. [**8**]

For linear programming (LP), the simplex method is commonly utilized to find an optimal solution for the system in question. With Integer Programming (Integer Linear Programming, ILP) a similar approach can be used in which repeated calls to LP are made, along with the addition of new constraints each time. The basic idea for this ILP is called the *branch-and-bound* method, where the goal is to optimize a linear function subject to a set of linear constraints such that the solution only contains integer values. The process is as follows:

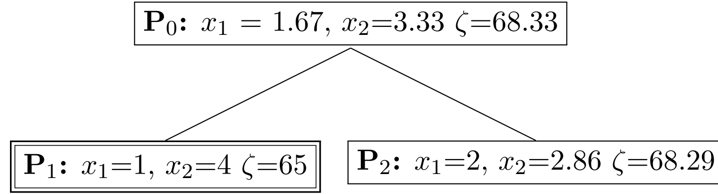
(1) First step is to use LP to solve the system, which for the sake of example shall be 2-dimensional, in the space of rational (or approximate real) numbers. If the LP solution occurs at integer values, the process terminates.
(2) If the solution from the initial LP is not integer, choose a variable x and set up two new LP problems, one with $x \geq m$, where $m = \lceil q \rceil$ (q being one of the non-integer solution of the initial LP), and the other with the constraint that $x \leq n$, where $n = \lfloor q \rfloor$.
(3) This process is repeated, creating a binary tree structure. The tree can be explored in either a depth-first (follow one branch in its entirety) or breadth-first (evaluate at each level of the tree, moving downwards) manner.
(4) The process terminates when the LP solution is integer, along with the objective function being either the minimum (or maximum) integer value. [**9**]

4.3. Example

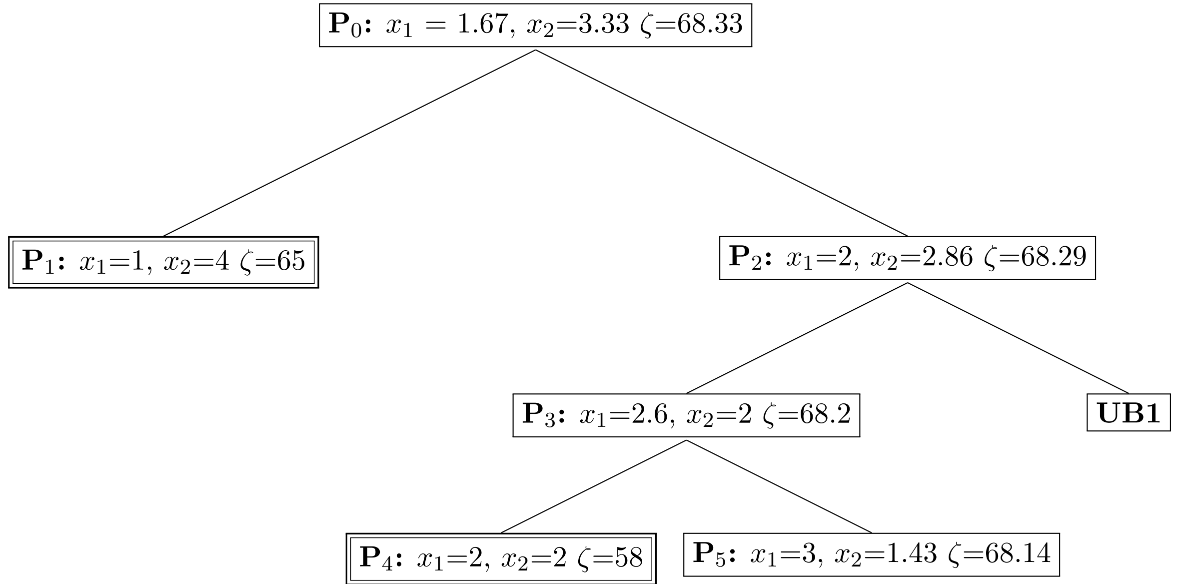
Let us consider the objective function $\zeta = 17x_1 + 12x_2$ subject to the constraints $x_1, x_2 \in \mathbb{N}$, $10x_1 + 7x_2 \leq 40$, $x_1 + x_2 \leq 5$, and $x_1, x_2 \geq 0$. This LP relaxation can be expressed in the following manner:

$$\begin{aligned} \text{maximize } & 17x_1 + 12x_2 \\ \text{subject to } & 10x_1 + 7x_2 \leq 40, \\ & x_1 + x_2 \leq 5, \\ & x_1, x_2 \geq 0, \\ & x_1, x_2 \in \mathbb{N}. \end{aligned}$$

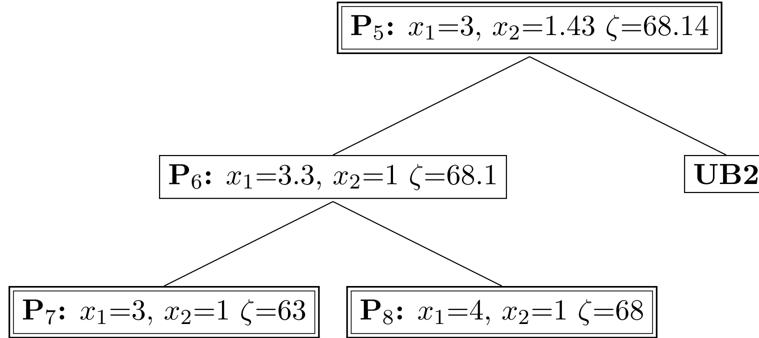
Using the simplex method, the LP relaxation finds the minimum value $\zeta = 68..33$ at $x = 1.67$ and $y = 3.33$ (let this point be denoted as $\mathbf{P}_0$). We then choose one of the variables, say x_1, and set up two new LP problems where if $x = 1.67$, one of the problems will have the additional constraint $x \geq \lceil 1.67 \rceil = 2$ and the other will have the additional constraint $x \leq \lfloor 1.67 \rfloor = 1$. This effectively pushes the value of x towards the nearest integer. In the following enumeration tree, the leftmost branch $\mathbf{P}_1$ represents the LP relaxation with the additional constraint $x_1 \leq 1$ and the rightmost branch $\mathbf{P}_2$ represents the LP relaxation with the additional constraint $x_1 \geq 2$.



Since the LP relaxation with $x_1 \leq 1$ gives us an integer solution, let us continue along the right most branch stemming from the original LP relaxation $\mathbf{P}_0$. With the additional constraint of $x \geq 2$ considered in the next implementation of the LP, the new minimum value generated for $\mathbf{P}_2$ is $\zeta = 68.29$. We again set up two new LP relaxation problems, moving the value for x_2 towards an integer value, one with the additional constraint $x_2 \geq 3$ (the right branch, which we will denote as **UB1**, but not evaluate just yet) and the other additional constraint $x_2 \leq 2$ (the left branch, which we will denote $\mathbf{P}_3$).



Now at $\mathbf{P}_3$ x_2 is integer, but $\zeta = 68.2$ and $x_1 = 2.6$ are not, so we continue down this branch. We add to further constrain the variable x_1, with P_4 adding the constraint $x_1 \leq 2$ and P_5 adding $x_1 \geq 3$. Notice above the P_4 is double boxed. This signifies that we have reached an integer solution (like what we saw in P_1), particularly one in which $x_1 = 2$, $x_2 = 2$ and $\zeta = 58$. Since we have not exhausted all possible branches (determining if we can find a better solution, or at least confirm infeasibility) we continue the process with P_5



With the above fragment of the enumeration tree for this Integer Program proces, we see that both $\mathbf{P}_7$ and $\mathbf{P}_8$ generate integer solutions, with the objective function $\zeta = 63$ for $\mathbf{P}_7$ and $\zeta = 68$ for $\mathbf{P}_8$. Therefore the branches for $\mathbf{P}_6$ have been accounted for and we may return to the two uncheck branches to see if they generate any integer solutions that will maximize the objective function. It turns out the two branches we left unchecked result in infeasible solutions, so the search is complete. The optimal integer solution for this LP relaxation problem is $x_1 = 4$, $x_2 = 0$, with the objective function generating $\zeta = 68$. [**8**]

4.4. Implementation

Imagining this matching problem as a Binary Integer Program (BIP), it is necessary to identify both an objective function and the collection of constraints of our given system. Before considering either construct, let us first define a number of terms relevant to our particular applied problem:

(1) Professors

$$\begin{aligned} Fa &= \text{auxiliary faculty member(s)}, \\ F &= \text{faculty}, \\ L &= \text{lecturers}, \\ I &= \text{instructors} := F \cup L. \end{aligned}$$

(2) Course Times

$$\begin{aligned} FC &= \text{fall courses}, \\ WC &= \text{winter courses}, \\ SC &= \text{spring courses}, \\ EMC &= \text{early morning courses}, \\ LMC &= \text{late morning courses}, \\ EAC &= \text{early afternoon courses}, \\ LAC &= \text{late afternoon courses}, \\ EvC &= \text{evening courses}, \\ CbyT &= \text{course by time} := EMC \cup LMC \cup EAC \cup LAC \cup EvC. \end{aligned}$$

(3) Course Days

$$\begin{aligned} mwfC &= \text{monday/wednesday/friday courses}, \\ trC &= \text{tuesday/thursday courses}, \\ CbyD &= \text{course by day} := mwfC \cup trC. \end{aligned}$$

(4) Course Types/Sizes

$$\begin{aligned} SmC &= \text{small lower division courses}, \\ BgC &= \text{big lower division courses}, \\ UpC &= \text{upper division courses}, \\ GrC &= \text{graduate courses}, \\ C &= \text{courses} := SmC \cup BgC \cup UpC \cup GrC. \end{aligned}$$

Since there are many possible assignments, IP algorithms are used to maximize a satisfaction function. Since the decision is binary, the variables x[p,c] take on a value of either 1 or 0, depending on whether instructor p is assigned to course c. Therefore maximization depends on a summation of weights, which can be denoted as d[p,c], whose values depend on the input instructor p and course c. The values of d[p,c] are integer values that are assigned by the user: d[p,c] is positive means "instructor p likes course c", zero means "indifferent", and a negative d[p,c] means "instructor p dislikes course c". The scale for the weights ranges from 100 to -100. Furthermore, the number of teaching units per instructor, which can be denoted by units(p), dictates both the number of courses and the types of courses (as there is a distinction between small and large courses).

$$\text{maximize} \sum_{(i,j)\in E} d[i,j] * x[i,j]$$

The following section will provide an example of how to take specific requests made by professors with regards to their teaching schedules and translate that into a functional set of constraints for the integer program.

(1) All courses need a professor
$\forall\ j \in \text{C}, \sum_{(i,j)\in E} x[i,j] = 1.$

(2) Instructors need to meet their unit duties
$\forall\ v \in I \backslash F, \sum_{(v,i)\in I\times K} x[v,i] + \sum_{(v,i)\in I\times BgC} 2x[v,i] = \text{units(v)},$
where $K = [SmC \cup UpC \cup GrC]$.

(3) Distribute big classes among the faculty
$\forall\ v \in I, \sum_{(v,i)\in F\times BgC} x[v,i] \leq 1.$

(4) Graduate courses taught by full-time faculty
$\forall\ v \in I \backslash F, \sum_{(v,i)\in I\times GC} x[v,i] = 0.$

(5) Faculty should not teach small lower division courses or too low
$\forall\ v \in \text{F}, \sum_{(v,i)\in F\times S} x[v,i] = 0,$
where S $= [SmC \cup \{"12-fall9am", "12-fall5pm", "B", "C"\}]$.

(6) Any individual faculty member gets a positive value of classes
$\forall\ v \in \text{F}, \sum_{(v,i)\in F\times C} -d[v,i] * x[v,i] \leq -1.$

As you can see, each set in question, be it faculty or courses, is further broken down into subsets to allow for more detailed specification of the intricacies of how each faculty member can be assigned to a particular course. Furthermore, this integer program only generates one of possibly many optimal feasible solutions, save for the very rare situation there is only one specific assignment of faculty members that will achieve the maximum of the objective (satisfaction) function. In all, this formulation allows a different perspective on the same problem, effectively generating a set of possible optimal binary solutions for the assignment problem.

APPENDIX A

Selected Pieces of Code in both Python and Zimpl

Gale Shapley Algorithm

These are the instructions for the product of our REU with Professor Jesus De Loera from the summer of 2012. We have written a Python script that implements a modified version of the Gale-Shapley Algorithm that provides a matching of people to time slots, be they teaching assistants (TAs) to discussion sections, professors to courses, or professors to committees.

These instructions will describe the process for matching TAs to discussion sections.

A.1. Setup

The first step is to install the necessary software to execute the script
1. First off, you must install Python (v2.7) on your computer. You can download it here: Python v2.7
2. Additionally, two modules to write and read in Excel files are required. You can download them here: XLRD XLWT
3. Keep everything in one folder: the Python script and the input Excel file

A.1.1. Formatting. After installing the prerequisite software, you must properly format the input excel file. You may follow the example file "Example.xlsx" as a guide.

There are a couple of important things to note:

- First and foremost, format all of your cells in each sheet of the input Excel file to be *Text*. This can be done by doing the following:
  (1) Press *Ctrl + A*
  (2) Right-click the current spreadsheet and select "Format cells"
  (3) Select "Text"

First Sheet:

- The first sheet will contain the information pertaining to the TAs. It will have the columns "Teaching Assistants", "Like", "Dislike", "Time Conflicts", "Number of Classes Taught", and "Ranking".

- The "Like" and "Dislike" columns are both in order of most to least. For example, if the list *16A, 22A, 16B, 250A* was in the "Like" column, then that TA's preferences will have 16A as his or her first choice, followed by 22A, etc. If it was in the "Dislike" column, then 16A would be his or her least preferable class to teach, 22A would be the next least preferable, etc. Also. make sure that the "Like" and "Dislike" columns have elements separated by exactly one (1) comma and one (1) space.
- In the "Time Conflicts" column, please do not put commas between times associated with the same group of days. You should model your entries after the following example: *MTWF 13.5-14.5 15-16, W 8-9, TR 10-11*. Each block of days and times is separated by a comma. Additionally, please use the 24-hour clock to represent the times, using the ending ".5" instead of ":30" for half-hour increments. In the previous example, 13.5-14.5 represents 1:30-2:30.
- The "Number of Classes Taught" column has the number of discussion sections each TA is teaching. This number is not the number of units the TA must teach, but the number of classes.
- The "Ranking" column has some number (integer) assigned to each TA representing a ranking of him or her. It should take into account seniority and past performance. These cells should not be left blank.

Second Sheet:

- The second sheet will have all of the information pertaining to the classes. It will have the the columns "CRN", "Class Name", "Time", and "Blacklist".
- The "CRN" column must be one number associated with one discussion section, and it may not be more than nine (9) digits long.
- The "Class Name" column has the name of the class associated with the corresponding CRN.
- The "Time" column must have at least one time slot for the corresponding discussion section.
- The "Blacklist" column is a list of TAs separated by one (1) comma and one (1) space that are not qualified to teach the corresponding class.

A.2. Running the Code

The next step after formatting the input Excel file is to run the script. Befor doing so, make sure that the file you want analyzed is in the same directory as *assignments.py*. Then, right-click on *assignments.py* and select "Open in IDLE". Next, click *Run* $\rightarrow$ *Run Module*, or alternatively press F5. Then in a separate window, you will be prompted for the name of the file you wish to have analyzed (including the file extension). Finally, press *Enter*, and the output Excel file will be automatically saved to your working directory.

A.3. Understanding the Output

There are two choices of optimal matchings that you can decide between, the first of which (on the first sheet in the outputted excel file) places more of an emphasis on the preferences provided by the TAs and the second one (second sheet on the same excel file) placing more emphasis on the individual rankings of the TAs. Each row in this Excel file corresponds to a matching between a TA and a discussion section with its corresponding CRN.

It is possible that not every TA will be matched to a class or vice versa. This can occur because of time conflicts ot restrictions from the blacklist. In such cases, it may be necessary to hire graduate students from other departments.

A.4. Integer Program Implementation

Here are some snippets from the integer program constructed by Dr. Jess De Loera in February of 2013 with the goal of generating acceptable assignments of faculty members to department courses (with the test data set being the Math Department Faculty and the courses taught by faculty from the department).

```
# MATHSCHEDULER: software to solve the instructor-to-course assignment problem at Dept. of MATH UCD.

# code and model developed February 2013 by Jesus De Loera.


# OVERVIEW: Instructors need to be assigned to courses.
# This is a binary decision to be made. In this software there
# are variables x[p,c] which take value 1 or 0 depending on whether
# instructor p is assigned course c.

# The assignments must satisfy a series of linear equations and inequalities.
# that enforce conditions: E.g., they do not go over their units or
# all courses are covered by an instructor.

# There are many possible assignments, we use IP algorithms
# to maximize a satisfaction function. These is the SUM of weights d(p,c)
# whose values depend on the input instructor p & course c.
# The values of d() are integer values that are assigned by the user
# (e.g., mso, chair,etc): d(p,c) is positive means "instructor p likes course c",
# zero means "indifferent", a negative d(p,c) means "c dislikes course p".
# Our scale is 100 to -100.

# NOTE: There is alot of freedom as to how the integers d(c,p) are assigned
# and they are the "subjective" part of the model. The d(c,p) are extracted
# the response of faculty to the call for courses. In the future faculty should
# provide this themselves.


# To construct the variables x[p,c] we first set
# the names of instructors and courses (with quarters+start time)

set faculty:=
```

```
{”babson”,”benham”,”biello”,”bremer”,”cheer”,”deloera”,”fannjiang”,”freund”,
”fuchs”,”gravner”,”guy”,”hass”,”hunter”,”kapovich”,”koeppe”,”kuperberg”,”lewis”’
”liu”,”mogilner”,”morris”,”mulase”,”nachtergaele”,”osserman”,”pizzo”,
”puckett”,”romik”,”saito”,”schilling”,”schultens”,”schwarz”,”shkoller”,”soshnikov”,
”strohmer”,”temple”,”thomases”,”thompson”,”tracy”,”vazirani”,”walcott”,
”waldron”,”xia”, ”newfaculty”};

set lecturers_and_kaps:=
{”daddel”,”kouba”,”marx”,”kap1”,”kap2”,”kap3”,”kap4”,”kap5”,”kap6”,”kap7”,
”kap8”, ”lect1”,”lect2”,”lect3”,”lect4”,”lect5”,”lect6”,”lect7”};

set visitors_and_occasional:=
{”chuchel”,”visitor1”,”visitor2”,”visitor3”,”fantasma”};

set instructors:=faculty union lecturers_and_kaps union
visitors_and_occasional;
```

Following the above segment of code, the courses for the entire year are separated by quarter, time of day, and size/level, with each denomination representing a particular subset of the whole set of courses. Furthermore, specific subsets are generated by the union of other subsets. For example, the subset *coursesbyquarter* := *fallcourses union wintercourses union springcourses*. In similar fashion, the courses separated by time and by day are grouped via the union of smaller subsets.

```
# Now we assign the official teaching units per instructor
param units[instructors] :=
<”babson”> 4, <”benham”> 2, <”biello”> 4,<”bremer”> 2,<”cheer”> 4,<”deloera”>
3,<”fannjiang”> 4, <”freund”> 4,<”fuchs”> 4,<”gravner”> 4,<”guy”> 4,<”hass”> 1,<”hunter”>
4,<”kapovich”> 1,<”koeppe”> 4,<”kuperberg”> 3,<”lewis”> 4,<”liu”> 1,<”mogilner”>
1,<”morris”> 4,<”mulase”> 4,<”nachtergaele”> 0,<”osserman”> 4,<”pizzo”> 0,<”puckett”>
4,<”romik”> 4,<”saito”>3,<”schilling”> 4,<”schultens”> 1,<”schwarz”> 3,<”shkoller”>
2,<”soshnikov”> 4,<”strohmer”> 3,<”temple”> 3,<”thomases”> 4,<”thompson”> 2,<”tracy”>
2,<”vazirani”> 4,<”walcott”> 4,<”waldron”> 4,<”xia”> 4,<”newfaculty”> 3,<”daddel”>
9,<”kouba”> 9,<”marx”> 9,<”kap1”> 3,<”kap2”> 4,<”kap3”> 4,<”kap4”> 4,<”kap5”>
4,<”kap6”> 4,<”kap7”> 4,<”kap8”> 4, <lect1”> 6,<”lect2”> 6,<”lect3”> 6,<”lect4”>
6,<”lect5”> 6,<”lect6”> 6,<”lect7”> 6, <”visitor1”> 2,<”visitor2”> 2,<”visitor3”> 2,<”chuchel”>
3,<”AuxiliaryFaculty”>1000;
```

NOW we are ready to set up the constraining equations, inequalities.

The variables are labelled by the pairs of instructors and courses. # A variable x[p,c]=1 if and only if professor p is assigned to teach course c.

set E := { <i,j> in (instructors cross courses)};

these pairs will be labeling our variables

var x[E] binary;

Here we enter the satisfaction function. This is read from the requests # of faculty and the needs of the department and the chair's decision.

Here is the procedure followed: # if a class c was starred (*) by professor p, it is very desirable, then # d(p,c)=100. d(p,c) is assigned a negative number if professor dislikes it # e.g., course c is too early for professor p.

WEIGHTED PREFERENCES HERE # satisfaction index (rankings go from 100 to -100).

HERE IS WHAT WE WANT TO MAXIMIZE

maximize total_satisfaction:
(sum <i,j> in E : d[i,j] * x[i,j]);

#-(sum <i,j> in ({"AuxiliaryFaculty"} cross courses): 10000*x[i,j]);

HERE ARE THE CONDITIONS THAT NEED TO BE TRUE!

#————-
All courses need to have a professor.

subto course_covering:
forall <j> in courses do

```
sum <i,j> in E : x[i,j] == 1;

#————–
# All professors received at least one course they like:

#subto something_to_like:
#forall <i> in faculty without {"newfaculty"} do
#sum <i,j> in E: d[i,j]*x[i,j] >=0;


#————–-
# Instructors need to meet their unit duties.

subto instructors_meet_teaching_units:
forall <v> in instructors without {"AuxiliaryFaculty"} do
(sum <v,i> in (instructors cross (smalllowcourses union uppercourses union gradcourses))
: x[v,i]) + (sum <v,i> in instructors cross biglowcourses : 2*x[v,i]) == units[v];


#—————
# This constraint aims to distribute big classes among many faculty (no more than one).

subto atmost_onebigclassfor_instructors:
forall <v> in instructors do
sum <v,i> in (faculty cross biglowcourses): x[v,i] <=1;

#—————-
# Graduate courses are taught by full-time faculty

subto gradcourses_restricted_to_faculty:
forall <v> in (instructors without faculty) do
sum <v,i> in instructors cross gradcourses: x[v,i]==0;

#—————-
# Faculty should not teach small lower division classes or too low

subto nosmalllowdivcourses_forfaculty:
forall <v> in faculty do
sum <v,i> in (faculty cross (smalllowcourses union {"12-fall9am","12-win5pm","B","C"})):
x[v,i]==0;
```

```
#—————————————————-
# Any individual faculty member gets a positive value of classes:


#subto satisfactionofeachfaculty:
#forall <v> in faculty do
#sum <v,i> in (faculty cross courses): -d[v,i] * x[v,i] <=-1;
```

Bibliography

[1] D. Bertsimas and R. Weismantel, *Optimization over integers*, Dynamic Ideas, 2005.

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

[3] D. Gusfield and R.W. Irving, *The stable marriage problem: Structure and algorithms*, The MIT Press, 1989.

[4] R. W. Irving, *Stable marriage and indifference*, Discrete Applied Mathematics **48** (1994), 261–272.

[5] R. W. Irving, D. F. Manlove, and S. Scott, *Stable marriage problem with master preference lists*, Discrete Applied Mathematics **156** (2008), 2959–2977.

[6] H.W. Kuhn, *The hungarian method for the assignment problem*, Naval Research Logistics Quarterly **2** (1955), 83–97.

[7] L. Lovasz and M.D. Plummer, *Matching theory*, Elsevier Science Publishers B.V., 1986.

[8] R. J. Vanderbei, *Linear programming: Foundations and extensions*, 2008.

[9] S. Wagon, *An algebraic approach to geometrical optimization*, Math Horizons (2012), 22–27.