Introduction to Linear Programming

A Simplex Implementation

Caleb, Gavin

September 19, 2024

1 Introduction

In this study, we aim to create our own implementation of the simplex method and apply the technique to a variety of linear programming problems. Such problems are often very complicated, featuring any number of linear relationships between variables that correspond to real world constraints, making it time consuming and often burdensome to solve by hand. The simplex tableau method provides a more efficient route to finding an optimal solution and provides an effective framework for addressing resource allocation, transportation, scheduling, and network flow problems. We begin the study with a brief observance of linear programming history before talking about the general problem of optimization. We then outline our specific implementation of the simplex tableau method and apply it to several relevant problems.

2 Linear Programming

In general, linear programming refers to a mathematical optimization technique used to find the best possible outcome given a set of linear constraints. These constraints are defined using equalities or inequalities and describe restrictions on the variables in objective function we are trying to optimize. It's feasible region, the intersection of the finitely many half planes defined by the constraints, is called a convex polytope. The objective function is real-valued and defined on this polyhedron. Alas, the solutions themselves are geometric, hinting at a possible approach utilizing this area of mathematics. In two-dimensions, this is in fact the common approach. We sweep the objective function across a polygon describing the feasible region until we reach the last intersection. Once we begin thinking in higher dimensions, this notion of a polytope is much more difficult to conceptualize. For these reasons we turn to the simplex method, which changes all of the inequalities to equalities through the introduction of slack variables (more on that later), and refines current solutions until we reach an optimum. An optimal solution implies maximum efficiency, highlighting a logical application in management situations where the goal is to maximize profit or minimize cost. We aim to apply this to technique to a wider range of problems that may fall outside of this original use.

2.1 History

As mentioned previously, linear programming problems are notoriously difficult to solve, which could be the reason for a limited history on the subject. The first person to really focus on the topic, Joseph Fourier, published a paper in 1827 with a way to solve these problems. Known as 'Fourier-Motzkin' method, the technique was a step in the right direction, but had a computational time that was exponential in the number of variables. A search for a better practice continued. It wasn't until 1939 that a Soviet mathematician, Leonid Kantorovich, developed a formulation for linear programs equivalent to today's general problem. The technique was meant to plan expenditures and returns in order to reduce costs in WWII, but ended up being neglected by the USSR. Around the same time, a Dutch-American economist Tjalling C. Koopmans modeled classical economics problems as linear programs.
The largest contribution in the field was made by George Dantzig, who developed the Simplex Method in 1947 to solve planning problems for the U.S. Airforce. This technique was the first to efficiently find a solution in most situations. Dantizig's original problem was to assign 70 people to 70 jobs. If a computer were to test all of the possible assignment permutations, it would run forever. However, when posed as a linear programming problem, the simplex method was able to find the optimal solution in moments.

2.2 General Linear Programming Problem

We denote an objective function, the function we are trying to maximize or minimize while subjected to linear constraints, as 'z'. It's a general linear combination of variables $x_1, x_2, ..., x_n$ such that

$z = c_1x_1 + c_2x_2 + + c_nx_n$. As mentioned earlier, the constraints can be either equality or inequality constraints and take the form...

$$a_1x_1 + a_2x_2 + a_nx_n \begin{Bmatrix} \leq \\ = \\ \geq \end{Bmatrix} b$$

where a and b are constants. Because all of the variables should be non-negative, we recognize the additional constraints $x_1, x_2, x_n \geq 0$. Thus we define a feasible solution to by any combination of these variables that satisfy the entire set of constraints. A feasible region is the set of all n-tuples that are feasible solutions. Any given solution is a half space, and thus the feasible region is the finite intersection of all half spaces described by the constraints, making the solution space (in the problems we will consider) a convex polytope. The full form of the general problem would then be:

$$\begin{aligned} \text{maximize} \quad & c_1x_1 + c_2x_2 + ... + c_nx_n = z \\ \text{subject to} \quad & a_{11}x_{11} + a_{12}x_{12} + ...a_{1n}x_{1n} \leq b_1 \\ & \vdots \\ & a_{m1}x_1 + a_{m2}x_2 + ... + a_{mn}x_n \leq b_n \\ & x_1 \geq 0 \quad i \in \{1, 2, ..., n\} \\ & b_j \geq 0 \quad j \in \{1, 2, ..., m\} \end{aligned}$$

Where n is the number of variables and m is the number of constraints (excluding the non-zero constraint for x and b). Notice that this 'standard' form consists of all less-than inequalities and is a maximization problem. Any equality can be converted into two simultaneous inequality constraints, where a greater-than equation is really a less-than equation multiplied by -1. One may wonder what happens if we want to minimize the objective function. Thankfully, this difference is arbitrary because maximizing z is equivalent to minimizing -z. The constraints and objective function are easier to work with in matrix form, thus:

$$\begin{aligned} \text{maximize} \quad & c^T x \\ \text{subject to} \quad & Ax \leq b \\ & x_i \geq 0 \end{aligned}$$

$$c = \begin{bmatrix} c_1 \\ \vdots \\ c_n \end{bmatrix}$$

$$x = \begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix}$$

$$A = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ & & \vdots & \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{bmatrix}$$

$$b = \begin{bmatrix} b_1 \\ \vdots \\ b_m \end{bmatrix}$$

It can be helpful to characterize the solutions for linear optimization problems. An infeasible problem is one with no solution at all. A feasible region is *bounded* if there exists an $M \in \mathbb{R}$ such that $M \geq x_i$ for every possible feasible solution. This is almost always the case when dealing with real world problems, otherwise the answer to every question would just be 'make more of it'.

2.3 Simplex Method

The simplex method was the first algorithm capable of solving large dimensional problems. It can be shown that if an extreme point is not the optimum, than the solution exists on an edge such that the objective function is strictly increasing as you move away from the original point. Of course, this idea hinges on the assumption that the feasible region is a convex polytope, and there are more nuanced parts of the program that handle unbounded solutions. In essence, the simplex method starts with a feasible solution and traverses the edges of the region until it reaches a maximum output.

Introducing the method is nearly impossible without and example problem, so we will consider the linear program described below:

$$\begin{aligned} \text{maximize} \quad & 3x_1 + 4x_2 \\ \text{subject to} \quad & x_1 + 2x_2 \leq 8 \\ & 2x_1 + x_2 \leq 6 \end{aligned}$$

where the non-negative variables are taken as an implicit constraint. The algorithm starts with the introduction of slack variables, a process that changes all of the inequalities to equalities. These slack variables, denoted s_1 and s_2 must also be positive. Thus the system of linear equations becomes:

$$\begin{aligned} x_1 + 2x_2 + s_1 = 8 \\ 2x_1 + x_2 + s_2 = 6 \end{aligned}$$

If we include the objective function, we can write the linear program in an augmented matrix form known as the simplex tableau.

$$\begin{array}{cccccc|c} \text{basis} & z & x_1 & x_2 & s_1 & s_2 & b \\ s_1 & 0 & 1 & 2 & 1 & 0 & 8 \\ s_2 & 0 & 2 & 1 & 0 & 1 & 6 \\ & 1 & -3 & -4 & 0 & 0 & 0 \end{array}$$

The first column is simply to remind us what basis we are in, it has no computational importance. Likewise, the top row is just for labelling purposes. From here on out when we refer to say 'row 1', we mean the numbers 0, 1, 2, 2, 0. We immediately notice that s_1 and s_2 are basic variables, and a trivial solution can be obtained by setting the non-basic variables (x_1 and x_2) equal to zero. This is the implicit starting point for most simplex tableaus of this form. Our goal is to force s_1 and s_2 to become the non-basic variables. How do we do this? We identify a pivot column and then select a single number to identify a pivot row. We transform this column into a pivot with respect to the selected point through elementary row operations, which now makes the corresponding variable a basic variable. We will walk through the finer details using this example defined above.

As mentioned before, a trivial solution is obtained by setting x_1 and x_2 equal to zero, then $z = 0$. Despite it being within the feasible region, it is clearly not the maximum. In order to traverse to the next solution, select the column that has the smallest value in the 'z' row. In this case, that value is -4 and thus we want the x_2 column to become a pivot. First we need to decide which entry in this column to make the pivot point. To do so, we take the value in each row and divide it into corresponding value in the 'b' column, and select the smallest result. Because $\frac{8}{2} = 4$ is smaller than $\frac{6}{1}$, the value 2 in row 1 will be our pivot value.

The first step is to divide $R1$ by 2, resulting in a value of 1 in our pivot point position. Next, we want to eliminate all other values in the column. Thankfully this is a simple example and there are only two other rows. Replace $R2$ with $R2 - R1_{new}$ and replace $R3$ with $R3 + 4R1_{new}$. Then we arrive at the second iteration of the simplex tableau:

$$\begin{array}{cccccc|c}
\text{basis} & z & x_1 & x_2 & s_1 & s_2 & b \\
x_2 & 0 & \frac{1}{2} & 1 & \frac{1}{2} & 0 & 4 \\
s_2 & 0 & \frac{3}{2} & 0 & -\frac{1}{2} & 1 & 2 \\
 & 1 & -1 & 0 & 2 & 0 & 16
\end{array}$$

We repeat the process but this time making x_1 the pivot column. The resulting matrix is:

$$\begin{array}{cccccc|c}
\text{basis} & z & x_1 & x_2 & s_1 & s_2 & b \\
x_2 & 0 & 0 & 1 & \frac{2}{3} & -\frac{1}{3} & \frac{10}{3} \\
x_1 & 0 & 1 & 0 & -\frac{1}{3} & \frac{2}{3} & \frac{4}{3} \\
 & 1 & 0 & 0 & \frac{5}{3} & \frac{2}{3} & 17\frac{1}{3}
\end{array}$$

Now both x_1 and x_2 are basic variables and we can read off the optimal solution from the tableau. The 'b' column states that the optimal solution is to take $x_1 = \frac{4}{3}$ and $x_2 = \frac{10}{3}$, resulting in an output of $17\frac{1}{3}$.

3 Computer Implementation

Using the process from Section 2.3, we can broadly outline the simplex method as Algorithm 1.

Algorithm 1 Naive Simplex Method

```
Put the objectives in the tableau with slack variables
while The objective has negative values do
    $j \leftarrow$ the most negative objective column          ▷ Pivot Column
    $i \leftarrow$ the smallest (positive) ratio test          ▷ Pivot Row
    Divide row $i$ by the value at $i, j$
    Row reduce to eliminate column $j$ from other rows
    Replace the basic variable in row $i$ with the variable from $j$
end while
```

This implementation, however, suffers from a number of key issues that make it unworkable. Solving these problems will require us to use a handful of related algorithms in our implementation.

3.1 Finding Feasible Regions

The first problem with the naive implementation is that the simple way to setup a tableau from Section 2.3 doesn't work in cases where the trivial zero solution is infeasible. To address this, we'll implement the Two Phase Method.

The Two-Phase Method, as you may have guessed, divides an LP into two phases. The first phase finds a feasible region and the second one finds the optimal solution in the feasible region. To do this, we begin by adding 'additional variables', a_i, to each of the infeasible constraints representing the distance between the current solution and the feasible solution. Then, ignoring the objective, we use the simplex method to minimize the sum of the additional variables. If this sum can be minimized to 0, we have a basic feasible solution (BFS) and can proceed to Phase 2.

If it can't, the problem is infeasible. Lastly, assuming we found a BFS, we plug the objective back into the tableau and use normal simplex iterations to find the optimal solution [4].

For example, given the linear program

$$\begin{aligned} \text{maximize} \quad & 3x_1 + 4x_2 \\ \text{subject to} \quad & x_1 + 2x_2 \leq 8 \\ & 2x_1 + x_2 \leq 6 \\ & x_1 + x_2 \geq 2 \\ & x_{1,2} \geq 0 \end{aligned}$$

the trivial, $x_1 = x_2 = 0$ solution doesn't meet the constraint that $x_1 + x_2 \geq 2$. Therefore, when we setup the simplex problem, we add the additional variable a_1 to this constraint to get

$$\begin{aligned} \text{minimize} \quad & a_1 \\ \text{subject to} \quad & x_1 + 2x_2 + s_1 = 8 \\ & 2x_1 + x_2 + s_2 = 6 \\ & x_1 + x_2 - s_3 + a_1 = 2 \\ & x_{1,2} \geq 0. \end{aligned}$$

Using a tableau, we have

basis	z	x_1	x_2	s_1	s_2	s_3	a_1	b
s_1	0	1	2	1	0	0	0	8
s_2	0	2	1	0	1	0	0	6
a_1	0	1	1	0	0	−1	1	2
z	1	0	0	0	0	0	1	0

Using the Simplex Iteration rules from Section 2.3 we get:

basis	z	x_1	x_2	s_1	s_2	s_3	a_1	b
s_1	0	1	2	1	0	0	0	8
s_2	0	2	1	0	1	0	0	6
a_1	0	1	1	0	0	-1	1	2
z	1	-1	-1	0	0	1	0	-2

Followed by

basis	z	x_1	x_2	s_1	s_2	s_3	a_1	b
s_1	0	0	1	1	0	1	-1	6
s_2	0	0	-1	0	1	2	-2	2
x_1	0	1	1	0	0	-1	1	2
z	1	0	0	0	0	0	1	0

At this point, since there are no negative values in the z row of the tableau, we check the b column of the row to see if either the value is zero, meaning we have a BFS, or the value is anything nonzero, meaning the problem is infeasible.

In this case, we get zero, meaning we can remove the a_1 columns and plug in the original objective to get

basis	z	x_1	x_2	s_1	s_2	s_3	b
s_1	0	0	1	1	0	1	6
s_2	0	0	-1	0	1	2	2
x_1	0	1	1	0	0	-1	2
z	1	-3	-4	0	0	0	0

Now we can use normal simplex iterations to solve the problem, getting $x_1 = \frac{4}{3}$ and $x_2 = \frac{10}{3}$.

Implementing the Two-Phase Method gives us the ability to deal with any type of constraint, not just less than. For example, the problem above has a constraint where a linear combination of x_1 and x_2 is greater than some number. For any constraint type, we simply need to combine slack and additional variables in different ways.

With a less-than condition, we add a positive slack variable, since the condition can have a positive difference between the right-hand-side and left-hand-side of the constraint. Therefore,[1]

$$\sum_i a_{ij}x_i \leq b_j \quad \text{becomes} \quad \sum_i a_{ij}x_i + s_j = b_j$$

With an equality condition, we can't have any slack between the right-hand-side and left-hand-side of the constraint. We do, however, need to find a feasible region, since the right-hand-side doesn't need to be 0. Therefore,

$$\sum_i a_{ij}x_i = b_j \quad \text{becomes} \quad \sum_i a_{ij}x_i + a_j = b_j$$

With a greater-than condition, we add a negative slack variable, since the difference between the right-hand-side and left-hand-side should be negative, and also add an additional variable, since we need to find a feasible region. Therefore,

$$\sum_i a_{ij}x_i \geq b_j \quad \text{becomes} \quad \sum_i a_{ij}x_i - s_j + a_j = b_j.$$

3.2 Dealing With Cycling

The process outlined earlier is also vulnerable to cycling. Cycling means the same tableau is reached after performing simplex iterations, particularly when the right hand side of a constraint is 0. For example, the tableau

basis	z	x_1	x_2	x_3	x_4	s_1	s_2	b
s_1	0	$\frac{1}{4}$	$-\frac{1}{8}$	12	10	1	0	0
s_2	0	$\frac{1}{10}$	$\frac{1}{20}$	$\frac{1}{20}$	$\frac{1}{5}$	0	1	0
z	1	-5	-4	20	2	0	0	0

cycles and gets back to itself after only 6 simplex iterations of picking the most negative objective value as the next column.

To avoid cycling, we use Bland's Rule. Bland's Rule says that we should pick the leftmost negative value as the pivot column, not the most negative. Using this process, we are guaranteed

[1]All of these assume b_j is positive. If it's not, multiply the condition by -1 and, if necessary, flip the inequality.

to avoid cycling [2], although there is a slight decrease in performance [1].

3.3 Unbounded Solutions

Finally, optimal solutions to some LPs include unbounded solutions. For example, the optimal solution to

$$\begin{aligned} \text{maximize} \quad & x_1 + x_2 \\ \text{subject to} \quad & x_1 \leq 10 \\ & x_1 + x_2 \geq 5 \end{aligned}$$

is an infinitely large x_2.

Therefore, we need to incorporate a way to detect this into our ratio test. Luckily this is quite simple: if all values in the pivot column are negative, that column is unbounded [5]. The ratio test with a check for unboundedness incorporated is described in Algorithm 2

Algorithm 2 Ratio Test

```
if The column has negative values then                ▷ Check for unboundedness
    return The LP is unbounded
end if
ratios ← Dividing column $i$ by the $b$ column
return The index of the smallest positive value in ratios
```

3.4 The Final Algorithm

Combining all of these, a high level pseudocode for our Simplex Implementation is described in Algorithm 3

Algorithm 3 Simplex Method

```
if There are $\geq$ or $=$ constraints then                                    ▷ Phase 1
    Add additional variables where needed
    Put the constraints in a tableau with minimizing additional variables as the objective
    while The objective has negative values do
        $j \leftarrow$ the first negative objective column                      ▷ Pivot Column
        if the column has negative values then                      ▷ Check for unboundedness
            return The LP is unbounded
        end if
        ratios $\leftarrow$ Dividing column $i$ by the $b$ column
        $i \leftarrow$ the index of the smallest positive value in ratios          ▷ Pivot Row
        Divide row $i$ by the value at $i, j$
        Row reduce to eliminate column $j$ from other rows
        Replace the basic variable in row $i$ with the variable from $j$
    end while
    if The objective isn't 0 then
        return the LP is infeasible
    end if
    Remove the additional variable columns in the tableau
else
    Put the constraints in a tableau and proceed to Phase 2
end if
while The objective has negative values do                                     ▷ Phase 2
    $j \leftarrow$ the first negative objective column                          ▷ Pivot Column
    if the column has negative values then                          ▷ Check for unboundedness
        return The LP is unbounded
    end if
    ratios $\leftarrow$ Dividing column $i$ by the $b$ column
    $i \leftarrow$ the index of the smallest positive value in ratios              ▷ Pivot Row
    Divide row $i$ by the value at $i, j$
    Row reduce to eliminate column $j$ from other rows
    Replace the basic variable in row $i$ with the variable from $j$
end while
```

A Python version of this algorithm that we wrote is available on GitHub.[2]

4 Applications

4.1 The Scheduling Problem

A cousin of the scheduling problem explored by George Dantzig in his initial optimization paper, the following problem explores the effectiveness of linear programming in scheduling shift work. We want to consider a small business, say a coffee shop for example, that has a few different employees with different availability and levels of experience. A manager's goal may be to minimize the total hourly wage of the employees working while maintaining the correct number of members on the floor. What does this look like in a program format?

We choose to have our decision variables, x_i, represent the number of employees of type i that should come to work at a given hour. The amount of employees working at any given time should exceed the set minimum for each hour of the day. For example, suppose the coffee shop is open from 7am to 3pm. Depending on the rush hour and various other factors, the minimum required employees may change depending on the time of day. Also suppose that the coffee shop has three types of employees: shift managers, regular employees, and trainees. Because of the different levels of experience, an hour of work from each employee type has a different labor value, 1 for the shift managers, 0.8 for the regular employees and 0.5 for the trainees. Let's explicitly create an equation for the hour 9 am, a very busy time for the coffee shop. At 9 am, there should be at least 5 units of labor working.

$$x_m + 0.8x_r + 0.5x_t \geq 5$$

Where x_m is the number of shift managers, x_r is the number of regular workers and x_t is the number of trainees. As you can imagine, there is a new equation for each hour of the day. If we add the additional constraint that each employee must work for 4 hours in a given shift, the resulting decision variables represent the number of employees that start at a given hour. Naturally, each employee type would likely earn a different wage, an idea reflected in the objective function where the goal is to minimize the total cost of paying employees for that day. Assuming

[2]Code available at https://github.com/GavinEngelstad/Comp-Geom-Simplex-Implementation

that shift workers make $c_m = \$25$ per hour, regular workers make $c_r = \$18$ per hour and trainees make $c_t = \$15$ per hour. The objective function becomes:

$$z = c_{m,\text{ 7am}} x_{m,\text{ 7am}} + c_{r,\text{ 7am}} x_{r,\text{ 7am}} + c_{t,\text{ 7am}} x_{t,\text{ 7am}} + ... c_{m,\text{ 3pm}} x_{m,\text{ 3pm}}$$

Where $x_{m,\text{ 7am}}$ is the number of shift managers, x_m, that will be starting at 7am. The demand for labor at each hour of the day is $b = \{3, 3, 5, 5, 4, 4, 3, 2\}$. We would like to add a few constraints other than labor requirements, the first of which is that there must be a manager working at all times. Managers have the highest wage, and if experience is not accounted for, the program would likely land on an optimal solution with no managers at all. A second preference would be to have at least one trainee work each day, despite their limited productivity and proportionally high wage. The final constraint states that there can be no more trainees than regular employees working at any given time. After all, we need someone to teach the trainees what to do. Our algorithm makes quick work of finding the optimal schedule. Below we report the results, where the program decided how many of each employee should come in at each hour of the work day.

7am:	1 manager, 2 regular employees, 1 trainee
8am:	no new employees
9am:	3 regular employees
10am:	no new employees
11am:	1 manager, 2 regular employees

Keep in mind that because every shift is at least four hours, no additional employees can come in at 12pm, 1pm, or 2pm. Our simplex method implementation is not an integer program, so decimals were rounded up. The cost of paying all of the workers for this hypothetical day would then be \$764.

4.2 The Traveling Salesman Problem

The Traveling Salesman Problem (TSP) asks us to find the least expensive cycle that visits every node in a graph once with costs to traversing each edge. An example of this problem is shown in Figure 1.

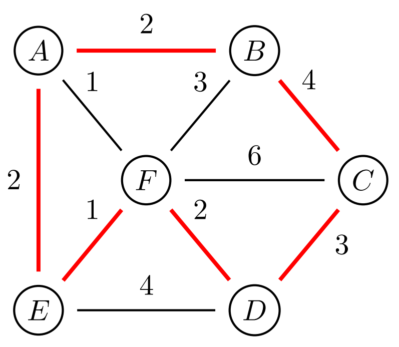

Figure 1: Traveling Salesman Problem example. Solution shown in red.

To set this up as a linear program, for each possible pair of nodes i and j, we create a variable $x_{ij} \in \{0, 1\}$. If the optimal Traveling Salesman path includes the edge from i to j, then $x_{ij} = 1$. Otherwise, $x_{ij} = 0$.

Since we want to minimize the costs of traversing the graph, the objective for a graph with n nodes will be

$$\text{minimize} \quad \sum_{i=0}^{n} \sum_{j=0, j \neq i}^{n} c_{ij} x_{ij}{}^{3}$$

where c_{ij} is the cost of traversing the edge from node i to node j.

The constraints need to be setup to ensure that a cycle is found that goes through each node. Therefore, the first two constraints are setup to ensure inflow and outflow into each node is equal to 1. Therefore, we can setup the LP as

$$\begin{aligned}
\text{minimize} \quad & \sum_{i=0}^{n} \sum_{j=0, j \neq i}^{n} c_{ij} x_{ij} \\
\text{subject to} \quad & \sum_{i=0, i \neq j}^{n} x_{ij} = 1 && \forall j \quad \text{Outflow from each node is 1} \\
& \sum_{j=0, j \neq i}^{n} x_{ij} = 1 && \forall i \quad \text{Inflow to each node is 1} \\
& x_{ij} \in \{0, 1\} \quad \forall i, j, i \neq j.
\end{aligned}$$

This formulation, however, can lead to two separate cycles. Figure 2 obeys both of these

[3] So far, this paper has only looked at maximization problems. Mininization problems can be solved through the same process, you just take the negative of the objective first.

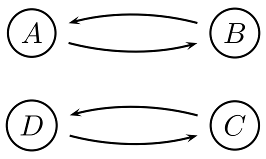

Figure 2: Degenerate cycle that still obeys the basic Traveling Salesman formulation

constraints, but isn't a valid TSP solution.

Therefore, we need to add a new constraint that forces solutions to be a single cycle that spans the whole graph, not two disconnected ones.

To do this, we'll use the Miller-Tucker-Zemlin (MTZ) formulation [3]. In the MTZ formulation, we add a term u_i for each node except the first with the condition that if $x_{ij} = 1$, then $u_j > u_i$. In practice, since the simplex method can't work with strict inequalities, this becomes $u_j \geq u_i + 1$. Under the MTZ formulation, every cycle needs to include the first node, otherwise we'll get $u_j > u_i$ and $u_j < u_i$ for some i and j nodes along the disconnected cycle. This forces us to get a complete cycle and, when minimized, a solution to the TSP.

To enforce this condition in a single inequality in an LP, we get

$$u_j \geq u_i + 1 - n(1 - x_{ij})$$

where the $-n(1 - x_{ij})$ forces the condition to hold when $x_{ij} = 1$ and there is a connection between i and j, but also allows enough slack that if $x_{ij} = 0$, then the solution values for u_i and u_j don't matter.

Therefore, the Traveling Salesman LP becomes

$$
\begin{array}{lrll}
\text{minimize} & \displaystyle\sum_{i=0}^{n} \sum_{j=0, j\neq i}^{n} c_{ij} x_{ij} & & \\
\text{subject to} & \displaystyle\sum_{i=0, i\neq j}^{n} x_{ij} = 1 & \forall j & \text{Outflow from each node is 1} \\
& \displaystyle\sum_{j=0, j\neq i}^{n} x_{ij} = 1 & \forall i & \text{Inflow to each node is 1} \\
& u_i - u_j + n x_{ij} \leq n - 1 & \forall i, j \geq 2, i \neq j & \text{MTZ Condition} \\
& x_{ij} \in \{0, 1\} & \forall i, j, i \neq j &
\end{array}
$$

When solved as an LP, the final condition that $x_{ij} \in \{0, 1\}$, isn't enforced. In our experience, however, this condition holds true unless two paths have the same cost, in which case the LP solution can have decimal values along both paths.

For example, we can use this LP to find a TSP solution for a graph with cost matrix

$$
\begin{array}{c|cccc}
 & A & B & C & D \\
\hline
A & & 8 & 6 & 2 \\
B & 9 & & 14 & 32 \\
C & 49 & 69 & & 81 \\
D & 19 & 18 & 17 &
\end{array}
$$

where each value in the matrix represents the cost of going from the node along the top row to the corresponding node in the first column (i.e. the cost to go from C to B is 14). By using an LP and solving for the optimal, minimized path,[4] you find the TSP solution is

$$\begin{tikzcd}
A \arrow[r, "49"] & C \arrow[r, "8"] & B \arrow[r, "18"] & D \arrow[lll, bend left, "2"]
\end{tikzcd}$$

[4]The LP for this problem is described in Appendix A.

5 Conclusion

Overall, Linear Programming is a powerful tool with a diverse array of applications. In this paper, we were able to successfully build an implementation of a popular, and historically rich, method to solve LPs, and use it to solve two interesting problems. The first, the scheduling problem, created a framework for an efficient way a small business can plan its labor. The second, the TSP, explored a famous difficult-to-solve problem and created a simple method to find solutions. Altogether, in this paper we explore the powerful tool of Linear Programming and learn both how to solve linear problems and some of the many real-world applications those problems can have.

Appendices

A TSP LP and Solution

The LP becomes

$$
\begin{array}{ll}
\text{minimize} & \begin{pmatrix} 9 & 49 & 19 & 8 & 69 & 18 & 6 & 14 & 17 & 2 & 32 & 81 & 0 & 0 & 0 \end{pmatrix} \vec{x} \\
\text{subject to} & \begin{pmatrix}
0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\
1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0
\end{pmatrix} \vec{x} = \begin{pmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{pmatrix} \\
& \begin{pmatrix}
0 & 0 & 0 & 0 & 4 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & -1 & 0 \\
0 & 0 & 0 & 0 & 0 & 4 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & -1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 4 & 0 & 0 & 0 & 0 & -1 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 4 & 0 & 0 & 0 & 0 & 1 & -1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 4 & 0 & -1 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 4 & 0 & -1 & 1
\end{pmatrix} \vec{x} \leq \begin{pmatrix} 3 \\ 3 \\ 3 \\ 3 \\ 3 \\ 3 \end{pmatrix}
\end{array}
$$

where

$$\vec{x} = \begin{pmatrix} x_{AB} \\ x_{AC} \\ x_{AD} \\ x_{BA} \\ x_{BC} \\ x_{BD} \\ x_{CA} \\ x_{CB} \\ x_{CD} \\ x_{DA} \\ x_{DB} \\ x_{DC} \\ u_B \\ u_C \\ u_D \end{pmatrix} = \begin{pmatrix} 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \\ 1 \\ 0 \\ 1 \\ 0 \\ 0 \\ 2 \\ 0 \\ 3 \end{pmatrix}$$

We hope it's clear why this wasn't included in the main part of paper.

References

[1] David Avis and Vasek Chvátal. "Notes on Bland's pivoting rule". In: *Polyhedral Combinatorics* (1978), pp. 24–34.

[2] Dimitris Bertsimas and John T. Tsitsiklis. *Introduction to Linear Optimization*. Athena Scientific, 1997.

[3] Martin Desrochers and Gilbert Laporte. "Improvements and extensions to the Miller-Tucker-Zemlin subtour elimination constraints". In: *Operations Research Letters* 10.1 (1991), pp. 27–36.

[4] Nebojša V. Stojković, Predrag S. Stanimirović, and Marko D. Petković. "Modification and implementation of two-phase simplex method". In: *nternational Journal of Computer Mathematics* 86.7 (2009), pp. 1231–1242.

[5] P. C. Tulsian and Vishal Pandey. “Quantitative Techniques: Theory and Problems”. In: *O’Reilly* ().