Linear Programming and its Connection to Discrete Optimization

Ansh Nagda

1 Introduction

A *Linear Program* (or LP) is a special case of constrained optimization over $\mathbb{R}^n$ where the objective function and the constraints are linear functions. *Linear Programming* is fundamentally important in two fields - first, in mathematical modelling. In particular, a large number of operations research problems can be formulated as linear programs. Secondly, and less obviously, there are problems in *Discrete Optimization* that can be reduced to solving a linear program. This is impressive because in general, discrete optimization is thought to be much harder than continuous optimization, but this reduction allows us to use the rich continuous structure of $\mathbb{R}^n$ to solve discrete optimization problems.

The goal of this term paper will be to explore one such application in discrete optimization that was introduced in 1956 by Hoffman and Kruskal in [HK10][1]. Before we can see their results in Section 2, we will have to build up our understanding of linear programming.

1.1 A Simple Example

To help the reader build familiarity with linear programs, here is a simple example of modelling an operations problem using LPs. Imagine that you are a company trying to manufacture two cereal products, say Cheerios and Froot Loops. For simplicity, assume that the only raw materials required are wheat and sugar, and that manufacturing one kilogram of the products requires the following amounts of raw materials:

	Cheerios	Froot Loops
Wheat	0.7 kg	0.6 kg
Sugar	0.3 kg	0.4 kg

Due to limitations of the factory, at most 1000 kilograms of Cheerios can be produced per day, and at most 600 kilograms of Froot Loops can be produced per day. Assume that Cheerios can be sold at a rate of \$4 per kg, and Froot Loops can be sold at a rate of \$7 per kg. Also, wheat can be bought at a rate of \$1 per kg, and sugar can be bought at a rate of \$2 per kg. Given this information, the

[1]The results in [HK10] are slightly more general than this term paper (in particular, they proved results about "minimal faces" rather than vertices)

problem of maximizing profit can be modelled as the following optimization problem over real valued variables:

$$\begin{aligned} \text{maxmimize} \quad & 4x_1 + 7x_2 - x_3 - 2x_4 \\ s.t. \quad & x \leq 1000 \\ & y \leq 600 \\ & 0.7x_1 + 0.6x_2 \leq x_3 \\ & 0.3x_1 + 0.4x_2 \leq x_4 \end{aligned}$$

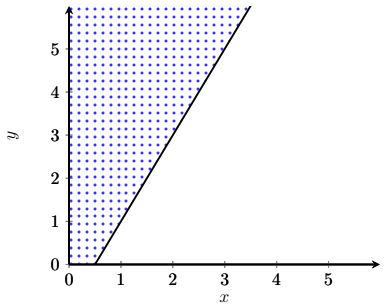
This problem is indeed an LP. A high school student could probably solve this LP using pen and paper. But this problem becomes increasingly nontrivial as more ingredients and products are considered. Fortunately, there are plenty of algorithms that computers can use to efficiently solve LPs on hundreds of variables and constraints. This term paper won't discuss these algorithms, but an interested reader might want to read about the ellipsoid algorithm[GLS81].

Note that if we were manufacturing more "discrete" products like tables and chairs, then our variables might have to be integers rather than real numbers.

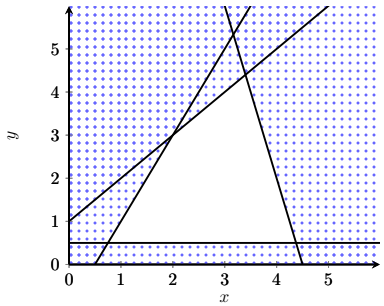
1.2 Initial Definitions

In order to study Linear Programs in detail, we need to formalize linear programs in a compact way. This section aims to provide this formalization. The definitions and proof in this section were inspired from [Sch].

We define a *halfspace* to be a set of the form $\{x \in \mathbb{R}^n : \langle a, x \rangle \leq b\}$ where $a \in \mathbb{R}^n$ and $b \in \mathbb{R}$. In other words, a halfspace is the solution set of a single linear inequality. Additionally, we define a *hyperplane* as the boundary of some halfspace. In two dimensions, halfspaces are halfplanes, and hyperplanes are lines. As an example, the halfspace $\{(x, y) \in \mathbb{R}^2 : -2x + y \leq -1\}$ is shown in white:

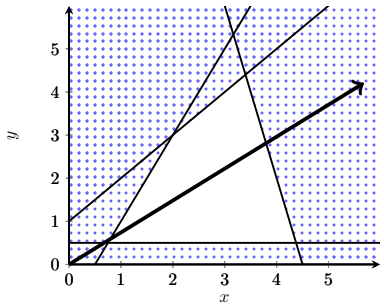


We define a *polyhedron* as the intersection of a finite number of halfspaces. In other words, a polyhedron is the solution set of a finite number of inequalities. In two dimensions, a bounded polyhedron is simply a convex polyhedron. Here is an example of a polyhedron defined by four halfspaces:



Given two vectors v and v' in $\mathbb{R}^n$, we say that $v \leq v'$ if for all $1 \leq i \leq n$, $v_i \leq v'_i$. This gives rise to a clean alternative definition of polyhedra: A *polyhedron* is any set of the form $\{x \in \mathbb{R}^n : Ax \leq b\}$ for some $m \times n$ matrix A and some $b \in \mathbb{R}^m$. As a first note, polyhedrons are convex due to linearity of matrix multiplication.

A *Linear Program* is an optimization problem over a subset of $\mathbb{R}^n$ where the objective function is linear and the constraints are linear inequalities. That is, a *Linear Program* is the problem of maximizing (or equivalently, minimizing) a linear functional over a polyhedron. We represent a linear program by $\max(\langle c, x\rangle : Ax \leq b)$. Here is a useful picture to keep in mind, where the arrow is supposed to indicate c (where c gives the coefficients of the linear function we want to optimize).



From the above picture, it should be intuitively clear why linear programs are optimized somewhere on the boundary of the polytope in question. Also observe that a point z is on the boundary of the halfspace $\{\langle a, x\rangle \leq b\}$ if $\langle a, z\rangle = b$. This is an important observation, and it motivates the following convenient definitions that will help us keep track of "boundary information" of points in a polytope.

Let $A \in \mathbb{R}^{m\times n}$ and $b \in \mathbb{R}^m$. Given an index $i \in [m]$[2], define a_i to be the i^{th} row of A. Given the polyhedron $P = \{x \in \mathbb{R}^n : Ax \leq b\}$ and a point $z \in P$, we define [3]$S_z \subseteq [m]$ to be the set of indices i such that $\langle a_i, z\rangle = b_i$. Additionally, we define A_z to be the submatrix of A obtained by keeping only the rows that appear in S_z. That is, A_z is the matrix obtained from A by deleting all rows i for which $\langle a_i, z\rangle < b_i$. Finally, we define b_z to be the correponding part of b. Note that $A_z z = b_z$ for all $z \in P$. Also note that $A_z \in \mathbb{R}^{m'\times n}$ and $b_z \in \mathbb{R}^{m'}$, where $m' \leq m$.

[2] $[m]$ equals the set $\{1, 2, \ldots, m\}$.

[3] Note that these definitions only make sense with respect to some polytope. Every time we use this notation, the polyhedron in question will be clear from context.

Lemma 1. *Let $P = \{Ax \leq b\} \subseteq \mathbb{R}^n$ be a polytope, and let $z \in P$. Let $c \in \mathbb{R}^n$ be a nonzero vector such that $A_z c = \vec{0}$. Then there exists $\delta > 0$ such that $z + \delta c \in P$.*

Proof. Let $S_1 \subseteq [m]$ be the indices of rows i such that $\langle a_i, c\rangle = 0$, and let $S_2 = [m] \setminus S_1$ be the rows i such that $\langle a_i, c\rangle \neq 0$. Pick $\delta := \min_{i \in S_2} \frac{b - \langle a_i, z\rangle}{|\langle a_i, c\rangle|}$. Note because $A_z c = \vec{0}$, $\langle a_i, c\rangle = 0$ for all $i \in S_z$. So $S_z \subseteq S_1$, which implies that S_z is disjoint from S_2. Therefore for all $i \in S_2$, $b - \langle a_i, z\rangle > 0$. Since δ is the minimum of finitely many positive numbers, it must be positive.

It remains to show that for all $i \in [m]$, $\langle a_i, z + \delta c\rangle \leq b_i$.

Case 1: $i \in S_1$. In this case, no matter what δ is,

$$\langle a_i, z + \delta c\rangle = \langle a_i, z\rangle + 0 = \langle a_i, z\rangle \leq b_i$$

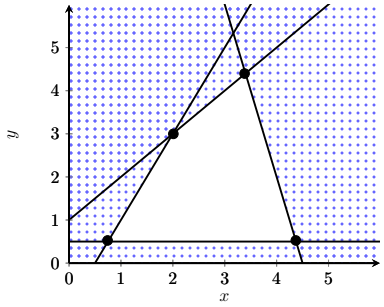
Case 2: $i \in S_2$. In this case,

$$\langle a_i, z + \delta c\rangle \leq \langle a_i, z\rangle + \delta|\langle a_i, c\rangle| \leq \langle a_i, z\rangle + \frac{b - \langle a_i, z\rangle}{|\langle a_i, c\rangle|}|\langle a_i, c\rangle| = b_i$$

Therefore $z + \delta c \in P$, and the proof is complete. □

1.3 Vertices of Polyhedra

In $\mathbb{R}^2$, one can easily obtain a visual understanding of "vertices" of a convex polygon. Here are the vertices of the polygon used as our example:



Vertices of polyhedra will be the central object that we will study, and this section is dedicated to formally defining and characterizing vertices. Given a finite set of points $X = \{x_1, \ldots, x_m\} \subseteq \mathbb{R}^n$, we define the *Convex Hull* of X (or $\text{Conv}(X)$) to be the set of points $\sum_{i=1}^m \lambda_i x_i$ such that $\lambda_i \geq 0$ and $\sum_{i=1}^m \lambda_i = 1$. If this definition is not intuitive, then one should keep the following equivalent definition in mind: $\text{Conv}(X)$ is the intersection of all convex sets containing S.

We now formally define a *vertex*: given a convex set S, we say that $x \in S$ is *not* a vertex of S if and only if there are two points $p, p' \in S$ distinct from x such that $x \in \text{Conv}(\{p, p'\})$. It turns out that in the case that S is a polyhedron, there is an nice characterization of the vertices of S:

Lemma 2. *Let $P = \{x \in \mathbb{R}^n : Ax \leq b\}$ be a polyhedron, and let $z \in P$. Then z is a vertex of P if and only if* $\text{rank}(A_z) = n$.

Proof. First assume that $z \in P$ and $\text{rank}(A_z) < n$, and will show that z is not a vertex. Since A_z is a linear transformation into $\mathbb{R}^n$, this implies that A_z has nontrivial kernel. That is, there is some vector $c \neq \vec{0}$ such that $A_z c = \vec{0}$. Applying lemma 1 on $\pm c$, there is some small $\delta > 0$ such that $z \pm \delta c \in P$. Clearly, $z = \frac{(z+\delta c)+(z-\delta c)}{2}$ lies in the convex hull of $z + \delta c$ and $z - \delta c$. Therefore z is not a vertex.

This completes the proof of the first direction. Now assume that z is not a vertx of P, and we will show that $\text{rank}(A_z) < n$. Because z is not a vertex, $z = \lambda x + (1-\lambda)y$ for some $x, y \in P$ that are different from z. Let $i \in S_z$. By definition of S_z, we have that $\langle a_i, z\rangle = b_i$. So

$$b_i = \lambda b_i + (1-\lambda)b_i \geq \lambda\langle a_i, x\rangle + (1-\lambda)\langle a_i, y\rangle = \langle a_i, z\rangle = b_i$$

Since $b_i = b_i$, it follows that the inequality is in fact an equality. Without loss of generality, assume $\lambda > 0$. Then $\langle a_i, x\rangle = b_i = \langle a_i, z\rangle$ for all $i \in S_z$. So $A_z(x - z) = \vec{0}$, and since $x \neq z$, this implies that the nullspace of A_z is nontrivial, so $\text{rank}(A_z) < n$.

□

Informally, one can think of Lemma 2 as asserting that in $\mathbb{R}^n$, vertices of P are exactly the intersections of n "linearly independent" hyperplanes that form the boundary of P.

Now let $S \subseteq [m]$ be some subset of indices, and let A' be the matrix of A formed by deleting all rows with indices in S, and let b' be the part of b. If $\text{rank}(A') = n$, then basic linear algebra tells us that the equation $A'z = b'$ has at most one solution, so there is at most one $z \in P$ such that $A_z = A'$. Since there are finitely many subsets of $[m]$, this gives us the following corollary of lemma 2:

Remark. *Polyhedra have finitely many vertices.*[4]

The following theorem is very intuitive in $\mathbb{R}^2$.

Theorem 3. *Every bounded polyhedron is the convex hull of its vertices.*

Proof. Let $P = \{Ax \leq b\}$ be a polyhedron, and let $X = \{x_1, \ldots, x_k\}$ be the vertices of P. Note that if $x \in \text{Conv}(X)$, then x is trivially in P (because $X \subseteq P$ and P is convex). Therefore $\text{Conv}(X) \subseteq P$. It remains to prove that $P \subseteq \text{Conv}(X)$.

Let $z \in P$. We will show that $z \in \text{Conv}(X)$ using induction on $n - \text{rank}(A_z)$. For the base case, if $n - \text{rank}(A_z) = 0$, then lemma 2 tells us that $z \in X$, so $z \in \text{Conv}(X)$.

Fix some $k \geq 0$. By induction, we can assume that if $n - \text{rank}(A_x) \leq k$, then $x \in \text{Conv}(X)$. Let z be a point such that $n - \text{rank}(A_z) = k + 1$. Then $\text{rank}(A_z) < n$, and there is a vector $c \neq \vec{0}$ such that $A_z c = \vec{0}$. Let $\mu = \max\{\mu : z + \mu c \in P\}$, and let $x = z + \mu c$. Note that $A_x c \neq \vec{0}$, because if it were zero, lemma 1 would imply that there is some $\delta > 0$ such that $(z + (\mu + \delta)c) \in P$, which contradicts optimality of μ. That is, $c \notin \text{nullspace}(A_x)$.

Also note that A_x contains all rows of A_z, because for all $i \in S_z$, $\langle a_i, x\rangle = \langle a_i, z\rangle + \mu\langle a_i, c\rangle = \langle a_i, z\rangle = b_i$. So $\text{nullspace}(A_x) \subseteq \text{nullspace}(A_z)$, but $c \in \text{nullspace}(A_z) \setminus \text{nullspace}(A_x)$. Therefore

[4] Additionally, a polyhedron that is described by m linear inequalities has at most 2^m vertices.

nullspace$(A_x) \subsetneq$ nullspace(A_z), which implies that A_x has greater rank than A_z, and by induction, $x \in \text{Conv}(X)$.

Similarly, define $\mu = \max\{\nu : z - \nu c \in P\}$, and $y = z - \nu c$. In the same way, we can prove that $y \in \text{Conv}(X)$. Since[5] $z = \frac{\nu}{\nu+\mu}x + \frac{\mu}{\nu+\mu}y$ lies in the convex hull of $\{x, y\}$, it follows that z is also in $\text{Conv}(X)$, which completes the proof.

□

The main use of this theorem will be the following corollary:

Corollary 3.1. *Linear Programs over bounded polyhedra are optimized at some vertex.*

Proof. Let $\max(\langle c, x\rangle : x \in P)$ be a linear program on a bounded polyhedron P, and let $\{x_1, \ldots, x_k\}$ be the vertices of P. Since P is closed and bounded, by the extreme value theorem, the linear program is optimized at some point $x \in P$, and say that $\langle c, x\rangle = M$. By theorem 3, $x = \sum \lambda_i x_i$ where $\{x_1, \ldots, x_k\}$ are the vertices of the polyhedron. Then

$$M = \langle c, x\rangle = \sum \lambda_i \langle c, x_i\rangle \leq M \sum \lambda_i = M$$

where the inequality is due to the fact that M is the maximum of the LP. Since $M = M$, the inequality is in fact an equality. Using the fact that $\langle c, x_i\rangle \leq M$ for all i, this implies that $\langle c, x_i\rangle = M$ (that is, the LP is optimized at x_i) for all i such that $\lambda_i > 0$. Since $\sum \lambda_i = 1$, there is some i such that $\lambda_i > 0$, and the linear program is optimized at x_i. □

We note that there are algorithms that can find a vertex optima of an LP[GLS81].

2 Integer Programming

We define *Integer Programs* (IPs) as optimization problems of the form $\max(\langle c, x\rangle : x \in \mathbb{Z}^n, Ax \leq b)$ where $\{Ax \leq b\}$ is a polytope. In other words, an integer program is the problem of optimizing a linear function over all integer points of a polytope. In general (unlike linear programming), integer programming is NP-hard, which means that most computer scientists don't expect that there is an efficient way to solve IPs. In this section, we will see a way to use linear programming to solve certain special cases of integer programming.

2.1 Some applications of Integer Programming to problems on graphs

Given a finite set of vertices[6] $V = \{v_1, v_2, \ldots, v_n\}$ and a set of edges $E = \{e_1, \ldots, e_m\}$, where E consists of two-element subsets of V, we say that (V, E) is an *undirected graph*. We define the

[5] Here we also use the fact that $\nu > 0$ and $\mu > 0$, which is given by lemma 1.

[6] Note that the vertices of a graph have nothing to do with the vertices of a polyhedron - this is simply an instance of using the same term in two different mathematical contexts.

incidence matrix of the graph as the $n \times m$ matrix M by

$$M_{i,j} = \begin{cases} 1 & \text{if } v_i \in e_j \\ 0 & \text{otherwise} \end{cases}$$

Given a finite set of vertices $V = \{v_1, v_2, \ldots, v_n\}$ and a set of edges $A = \{e_1, \ldots, e_m\}$, where A consists of 2-tuples over V, we say that (V, A) is a *directed graph*. We define the *incidence matrix* of this directed graph as the $n \times m$ matrix M by

$$M_{i,j} = \begin{cases} -1 & \text{if } v_i \text{ is the first entry of } e_j \\ +1 & \text{if } v_i \text{ is the second entry of } e_j \\ 0 & \text{otherwise} \end{cases}$$

Before we see how to solve some integer programs, we will motivate integer programming using some very natural and important optimization problems on graphs.

1. Assignment Problem: We say that a set $E' \subseteq E$ is a *matching* if for all sets $e_1, e_2 \in E'$, e_1 is disjoint from e_2. That is, a matching is a collection of edges that don't share any endpoints. The *maximum matching problem* is the problem of finding the largest cardinality matching in E. More generally, given weights on edges $w : E \to \mathbb{R}$, the *weighted matching problem* consists of finding a matching $E' \subseteq E$ such that $\sum_{e \in E'} w(e)$ is maximized. It is not too difficult to see that the weighted matching problem is equivalent to the following integer program[7]:

$$\begin{aligned} \max \quad & \sum w(e) x_e \\ s.t. \quad x_e \in \mathbb{Z} \quad & \forall e \in E \\ 0 \leq x_e \leq 1 \quad & \forall e \in E \\ \sum_{e \in E : v \in e} x_e \leq 1 \quad & \forall v \in V \end{aligned}$$

   If M is the incidence matrix of (V, E), then we can write this IP in a more compact equivalent form: $\max(\langle w, x \rangle : x \in \mathbb{Z}^m, \vec{0} \leq x \leq \vec{1}, Mx \leq \vec{1})$, where w is defined as the m-dimensional vector such that $w_i = w(e_i)$.

   An undirected graph (V, E) is called *bipartite* if V can be partitioned as $V = V_1 \dot{\cup} V_2$ such that all edges in E have exactly one endpoint in V_1, and exactly one endpoint in V_2. The *Assignment Problem* is the important special case of the weighted matching problem when the graph (V, E) is bipartite.

2. Integer Maximum Flow: Given a directed graph (V, A), a source $s \in V$, a sink $t \in V$, and capacities on edges $c : E \to \mathbb{R}^+$, a legal *flow* is a function $f : E \to \mathbb{R}^+$ such that for all

[7] Here, the variable x_e is supposed to be the indicator of the event "$e \in E'$".

$v \notin \{s, t\}$,

$$\sum_{(v_1,v_2)\in E: v=v_1} f((v_1, v_2)) = \sum_{(v_1,v_2)\in E: v=v_2} f((v_1, v_2))$$

In other words, a flow is an assignment of nonnegative value to each edge such that for all vertices except s and t, the amount of flow entering the vertex is the same as the amount of flow leaving the vertex. The *Maximum Flow Problem* is the problem of finding a legal flow that maximizes the net flow entering the sink t. There is a natural formulation of this problem as a linear program (where x_e is supposed to equal $f(e)$):

$$\begin{aligned}
\max \quad & \sum_{(v_1,v_2)\in E: t=v_2} x_{(v_1,v_2)} - \sum_{(v_1,v_2)\in E: t=v_1} x_{(v_1,v_2)} & \\
& s.t. \quad x_e \in \mathbb{Z} & \forall e \in E \\
& 0 \le x_e \le c(e) & \forall e \in E \\
& \sum_{(v_1,v_2)\in E: v=v_1} x_{(v_1,v_2)} = \sum_{(v_1,v_2)\in E: v=v_2} x_{(v_1,v_2)} & \forall v \in V \setminus \{s, t\}
\end{aligned}$$

Let M be the incidence matrix of the directed graph. Delete the two rows correponding to the vertices s and t, and call the resulting matrix M'. Also, let w the row of M corresponding to t. Then the condition that x induces a legal flow is equivalent to the condition that $M'x = 0$. This gives rise to the following compact form of the maximum flow LP: $\max(\langle w, x\rangle : x \in \mathbb{R}^m, \vec{0} \le x \le c, M'x = \vec{0})$, where c is defined as the m-dimensional vector such that $c_i = c(e_i)$.

The maximum flow problem is a problem included in almost every undergraduate algorithms course. One reason for this is that an astonishingly large number of discrete problems on graphs[8] can be reduced to finding an *Integer Maximum Flow* given some integer capacities, which is why the integer program $\max(\langle w, x\rangle : x \in \mathbb{Z}^m, \vec{0} \le x \le c, M'x = \vec{0})$ is of interest.

2.2 Totally Unimodular Matrices

Now we turn our focus back to the realm of linear and integer programming.
Given a matrix M, a submatrix of M is a matrix formed by deleting some number of rows and columns of M. For example, given a linear program $\max(\langle c, x\rangle : Ax \le b)$, the matrix A_z is a submatrix of A. We say that a matrix M is *Totally Unimodular* or *TU* if every square submatrix of M has determinant in $\{-1, 0, 1\}$. Note that a TU matrix automatically has entries in $\{-1, 0, 1\}$. With this definition, we are ready to present the punchline of this section.

[8] Some of these problems include finding the maximum number of paths between two vertices and finding the *weighted minimum cut* of a graph.

Theorem 4. *If A is a TU $n \times m$ matrix, $b \in \mathbb{Z}^m$, and $c \in \mathbb{R}^n$ such that $\{Ax \leq b\}$ is a bounded polyhedron, then*

$$\max(\langle c, x\rangle : x \in \mathbb{Z}^n, Ax \leq b) = \max(\langle c, x\rangle : x \in \mathbb{R}^n, Ax \leq b)$$

and there is an algorithm that can efficiently optimize the above integer program.

Essentially, this theorem tells us that given a polytope P of the form $\{Ax \leq b\}$ for TU A and integer b, linear programming over P is equivalent to integer programming over P. This theorem follows immediately by combining corollary 3.1 with the following lemma:

Lemma 5. *If A is a TU $n \times m$ matrix and $b \in \mathbb{Z}^m$, then every vertex of the polyhedron $P = \{Ax \leq b\}$ is an integer vector.*

Proof. Let z be a vertex of P. By lemma 2, $\text{rank}(A_z) = n$, so we can delete linearly dependent rows of A_z until it is a square $n \times n$ invertible matrix. Call the resulting matrix A'. By total unimodularity, it follows that $\det A' \in \{-1, 0, 1\}$, and since A' is invertible, $\det A' \in \{-1, 1\}$. Let C be the cofactor matrix of A'. C must be an integer matrix (because A' has entries in $\{-1, 0, 1\}$). Therefore $(A')^{-1} = \frac{C^T}{\det A'} = \pm C^T$ is an integer matrix.

By definition of A_z, $A_z z = b_z$. Let b' be the subvector of b that corresponds to A'. Then it is true that $A'z = b'$. So $z = (A')^{-1}b'$ is an integer vector. □

Therefore we can efficiently solve integer programs over TU matrices.

2.3 Back to applications

In order to apply the results we just proved on our graph problems, we must prove that certain matrices related to the graphs are totally unimodular. The following proofs are long but easy.

Lemma 6. *Let M be the incidence matrix of an undirected bipartite graph. Then M is totally unimodular.*

Proof. Let $(V = \{v_1, \ldots v_n\}, E)$ be the bipartite graph where $V = V_1 \dot\cup V_2$ such that all edges have exactly one endpoint in V_1 and exactly one endpoint in V_2. Without loss of generality, assume that $V_1 = \{v_1, \ldots, v_k\}$ and $V_2 = \{v_{k+1}, \ldots, v_n\}$ for some k.

Let M be the incidence matrix of (V, E). Let B be some $j \times j$ submatrix, and we will show that $\det B \in \{-1, 0, 1\}$ by induction on j.

The base case (when $j = 1$) is trivial because the entries of M are either 0 or 1. Now let $j > 1$. One can think of the submatrix B as defining a "subgraph" with vertices $\{v'_1, \ldots v'_j\}$ and edges $\{e'_1, \ldots, e'_j\}$.

Case 1: There exists some i such that $e'_i \cap \{v'_1, \ldots v'_j\}$ is empty. In this case, the i^{th} column of B is zero, so $\det B = 0$.

Case 2: There exists some i such that $e'_i \cap \{v'_1, \dots v'_j\}$ has exactly one element. In this case, the i^{th} column of B contains exectly one 1. Up to permuting the rows and columns, B has the following structure, where B'' is a $(j-1) \times (j-1)$ matrix:

$$B = \left(\begin{array}{c|c} 1 & B' \\ \hline \vec{0} & B'' \end{array}\right)$$

Since permuting rows and columns preserves determinant up to sign, we can expand the determinant along the first column to yield that $\det B = \pm \det B''$. By induction, this implies that $\det B \in \{-1, 0, 1\}$.

Case 3: For every i, $e'_i \cap \{v'_1, \dots v'_j\}$ has exactly two elements. In this case, every column of B contains exactly two 1s. We write

$$B = \left(\begin{array}{c} B_1 \\ \hline B_2 \end{array}\right)$$

where B_1 consists of the rows of A corresponding to the vertices in V_1, and B_2 consists of the rows of A corresponding to the vertices in V_2. By definition of bipartite graphs, it cannot be that some column of B_1 has two 1s, and it cannot be that some column of B_2 has two 1s. Therefore each column of B_1 has exactly one 1, so summing all rows of B_1 yields $\vec{1}$. Similarly, summing all rows of B_2 yields $\vec{1}$. This means that the rows of B are not linearly independent, so $\det B = 0$.

□

Lemma 7. *Let M be the incidence matrix of a directed graph. Then M is totally unimodular.*

Proof. This proof will be almost identical to the proof of lemma 6. Let (V, A) be the directed graph where $V = \{v_1, \dots v_n\}$. Let M be the incidence matrix of (V, A). Let B be some $j \times j$ submatrix, and we will show that $\det B \in \{-1, 0, 1\}$ by induction on j.

The base case (when $j = 1$) is trivial because the entries of M are either -1, 0 or 1. Now let $j > 1$. One can think of the submatrix B as defining a "subgraph" with vertices $\{v'_1, \dots v'_j\}$ and edges $\{e'_1, \dots, e'_j\}$.

Case 1: There exists some i such that none of the endpoints of e'_i are in $\{v'_1, \dots v'_j\}$. In this case, the i^{th} column of B is zero, so $\det B = 0$.

Case 2: There exists some i such that e'_i has exactly one endpoint in $\{v'_1, \dots v'_j\}$. In this case, the i^{th} column of B contains exactly one nonzero entry, which is either -1 or 1. Up to permuting the rows and columns, B has the following structure, where B'' is a $(j-1) \times (j-1)$ matrix:

$$B = \left(\begin{array}{c|c} \pm 1 & B' \\ \hline \vec{0} & B'' \end{array}\right)$$

Since permuting rows and columns preserves determinant up to sign, we can expand the determinant along the first column to yield that $\det B = \pm \det B''$. By induction, this implies that $\det B \in \{-1, 0, 1\}$.

Case 3: For every i, both the endpoints of e'_i are in $\{v'_1, \ldots, v'_j\}$. In this case, every column of B contains exactly one -1 and exactly one $+1$. Clearly, summing the rows of B yields $\vec{0}$, so $\det B = 0$.

□

Recall the integer program for the assignment problem: it was of the form $\max(\langle w, x\rangle : x \in \mathbb{Z}^m, \vec{0} \leq x \leq \vec{1}, Mx \leq \vec{1})$ where M is $n \times m$. Let

$$A = \begin{pmatrix} M \\ -I \\ I \end{pmatrix} \quad B = \begin{pmatrix} \vec{1} \\ \vec{0} \\ \vec{1} \end{pmatrix} \tag{1}$$

where A is $3n \times m$ and $B \in \mathbb{Z}^{3n}$. Note that the integer program for assignment is exactly the same as the integer program $\max(\langle w, x\rangle : x \in \mathbb{Z}^m, Ax \leq B)$.

Now consider the integer maximum flow problem. The integer program was of the form $\max(\langle w, x\rangle : x \in \mathbb{Z}^m, \vec{0} \leq x \leq c, M'x = \vec{0})$. Since M' was a submatrix of M, M' is TU if M is TU. Let

$$C = \begin{pmatrix} M' \\ -M' \\ -I \\ I \end{pmatrix} \quad D = \begin{pmatrix} \vec{1} \\ -\vec{1} \\ \vec{0} \\ c \end{pmatrix} \tag{2}$$

where C is $4n \times m$ and $D \in \mathbb{Z}^{4n}$. Note that the integer program for integer maximum flow is exactly the same as the integer program $\max\{\langle w, x\rangle : x \in \mathbb{Z}^m, Cx \leq D\}$.

We prove the following easy remark in Appendix A:

Remark. *If M is totally unimodular, then the matrices A and C defined in 1 and 2 are totally unimodular.*

Applying theorem 4 finally gives us the following result.

Theorem 8. *The optimization problems of* Assignment *and* Integer Maximum Flow *can be solved efficiently.*

3 Comments and Conclusion

We note that although linear programs can be solved "efficiently" in a theoretical sense, in practice, optimizing an LP is one of the worst ways to solve Assignment and Integer Maximum Flow problems. There are much faster algorithms that directly solve those problems without having to solve an LP. However, those algorithms utilize deep structural information about the problems. What's impressive about Hoffman and Kruskal's paper [HK10] is that their result barely uses any combinatorial insight into the specific graph problems.

Due to the scope of this term paper, we were not able to prove the *duality theorem* of linear programs. If we were equipped with the duality theorem, then theorem 4 would provide short proofs of some essential theorems in combinatorics (notably, Konig's theorem and the max-flow min-cut theorem). A reader interested in these proofs can read them in [Sch].

Before concluding, we would like to point out another interesting way to use linear programs to solve discrete optimization problems - *LP Rounding*. In most cases, we cannot formulate our desired discrete optimization problem as an integer program on a totally unimodular matrix. But in many cases, it turns out that we can "round" the solution of the corresponding linear program to an integer point, and if we round by a sufficiently small amount, this results in an approximately optimal solution to the integer program. This method is routinely used to create *Approximation Algorithms* for NP-hard problems. An interested reader should consider reading chapters 4 and 5 of [WS11].

References

[GLS81] Martin Grötschel, László Lovász, and Alexander Schrijver. The ellipsoid method and its consequences in combinatorial optimization. *Combinatorica*, 1(2):169–197, 1981.

[HK10] Alan J Hoffman and Joseph B Kruskal. Integral boundary points of convex polyhedra. In *50 Years of integer programming 1958-2008*, pages 49–76. Springer, 2010.

[Sch] Alexander Schrijver. *A course in combinatorial optimization.*

[WS11] David P Williamson and David B Shmoys. *The design of approximation algorithms*. Cambridge university press, 2011.

A Proof of total unimodularity of certain matrices

This section contains the proof that the matrices A and C defined in equations 1 and 2 are totally unimodular if M is totally unimodular.

Lemma 9. *If M is TU $n \times m$ matrix, then the following matrices are also TU:*

(a) $\begin{pmatrix} M \\ e \end{pmatrix}$ *where e is some standard basis vector of $\mathbb{R}^m$.*

(b) $\begin{pmatrix} M \\ m \end{pmatrix}$ *where m is some row of M.*

(c) Any matrix obtained by flipping the sign of some row of M.

Proof Sketch. For (a), the proof essentially boils down to the applying the determinant expansion formula along the row corresponding to e.

For (b), the proof boils down to two cases - firstly, if a square submatrix contains two copies of m, then the submatrix has determinant 0. Otherwise, the submatrix has determinant in $\{-1, 0, 1\}$ by total unimodularity of M.

For (c), notice that flipping the sign of any row preserves the determinant up to sign. □

Parts (a) and (b) of lemma 9 imply that we can append identity matrices and copies of M while preserving total unimodularity. Together with part (c), it is trivial to construct the matrices A and B from M using these operations that preserve total unimodularity.