9
Graph Matchings III: Weighted Matchings

In this chapter, we study the matching problem from the perspective of linear programs, and also learn results about linear programming using the matching problem as our running example. In fact, we see how linear programs capture the structure of many problems we have been studying: MSTs, min-weight arborescences, and graph matchings.

9.1 Linear Programming

We start with some basic definitions and results in Linear Programming. We will use these results while designing our linear program solutions for min-cost perfect matchings, min-weight arborescences and MSTs. This will be a sufficient jumping-off point for the contents of this lecture; a more thorough introduction to the subject can be found in the introductory text by Matoušek and Gärtner.

Definition 9.1. Let $\vec{a} \in \mathbb{R}^n$ be a vector and let $b \in \mathbb{R}$ a scalar. Then a *half-space* in $\mathbb{R}^n$ is a region defined by the set $\{\vec{x} \in \mathbb{R}^n \mid \vec{a} \cdot \vec{x} \geq b\}$.

As an example, the half space $S = \{\vec{x} \mid \begin{bmatrix} 1 \\ 1 \end{bmatrix} \cdot \vec{x} \geq 3\}$ in $\mathbb{R}^2$ is shown on the right. (Note that we implicitly restrict ourselves to *closed* half-spaces.)

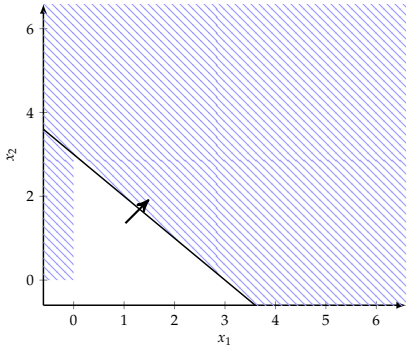


Figure 9.1: The half-space in $\mathbb{R}^2$ given by the set S

9.1.1 Polytopes and Polyhedra

Definition 9.2 (Polyhedron)**.** A *polyhedron* in $\mathbb{R}^n$ is the intersection of a finite number of half spaces.

A polyhedron is a convex region which is defined by finitely many linear constraints. A polyhedron in n dimensions with m constraints is often written compactly as

$$K = \{Ax \leq b\},$$

where A is an $m \times n$ *constraint matrix*, x is an $n \times 1$ vector of variables, and b is an $m \times 1$ vector of constants.

Definition 9.3 (Polytope)**.** A *polytope* $K \in \mathbb{R}^n$ is a bounded polyhedron.

In other words, a polytope is polyhedron such that there exists some radius $R > 0$ such that $K \subseteq B(\mathbf{0}, R) = \{x \mid \|x\|_2 \leq R\}$. A simple example of a polytope (where the bounded region of the polytope is highlighted by ▨) appears on the right. We can now define a linear program (often abbeviated as LP) in terms of a polyhedron.

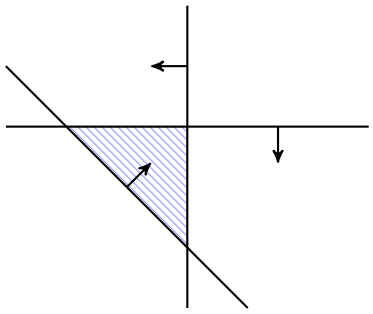


Figure 9.2: The polytope in $\mathbb{R}^2$ given by the constraints $-x_1 - x_2 \leq 1$, $x_1 \leq 0$, and $x_2 \leq 0$.

Definition 9.4 (Linear Program)**.** For some integer n, a polyhedron $K = \{x \mid Ax \leq b\}$, and an n by 1 vector c, a *linear program* in n dimensions is the linear optimization problem

$$\min\{c \cdot x \mid x \in K\} = \min\{c \cdot x \mid Ax \leq b\}.$$

The set K is called the *feasible region* of the linear program.

Although all linear programs can be put into this canonical form, in practice they may have many different forms. These presentations can be shown to be equivalent to one another by adding new variables and constraints, negating the entries of A and c, etc. For example, the following are all linear programs:

$$\max_x\{c \cdot x : Ax \leq b\} \qquad \min_x\{c \cdot x : Ax = b\}$$
$$\min_x\{c \cdot x : Ax \geq b\} \qquad \min_x\{c \cdot x : Ax \leq b, x \geq 0\}.$$

The polyhedron K need not be bounded for the linear program to have a (finite) optimal solution. For example, the following linear program has a finite optimal solution even though the polyhedron is unbounded:

$$\min_x\{x_1 + x_2 \mid x_1 + x_2 \geq 3\}. \tag{9.1}$$

9.1.2 *Vertices, Extreme Points, and BFSs*

We now introduce three different classifications of some special points associated with polyhedra. (Several of these definitions extend to convex bodies.)

Definition 9.5 (Extreme Point)**.** Given a polyhedron $K \in \mathbb{R}^n$, a point $x \in K$ is an *extreme point* of K if there do not exist distinct $x_1, x_2 \in K$, and $\lambda \in [0, 1]$ such that $x = \lambda x_1 + (1 - \lambda)x_2$.

In other words, x is an extreme point of K if it cannot be written as the convex combination of two other points in K. See Figure 9.3 for an example.

Here's another kind of point in K.

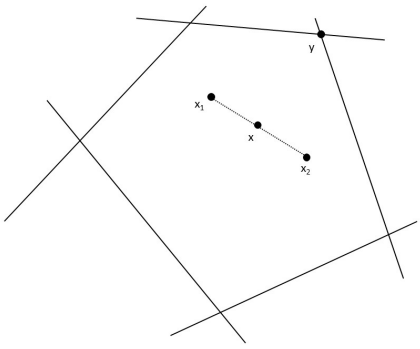


Figure 9.3: Here y is an extreme point, but x is not.

In this course, we will use the notation $c \cdot x, c^\intercal x$, and $\langle c, x \rangle$ to denote the inner-product between vectors c and x.

Definition 9.6 (Vertex)**.** Given a polyhedron $K \subseteq \mathbb{R}^n$, a point $x \in K$ is a *vertex* of K if there exists an vector $c \in \mathbb{R}^n$ such that $c \cdot x < c \cdot y$ for all $y \in K$ $y \neq x$.

In other words, a vertex is the unique optimizer of some linear objective function. Equivalently, the hyperplane $\{y \in \mathbb{R}^n \mid c \cdot y = c \cdot x\}$ intersects K at the single point x. Note that there may be a polyhedron that does not have any vertices: e.g., one given by a single constraint, or two parallel constraints.

Finally, here's a third kind of special point in K:

Definition 9.7 (Basic Feasible Solution)**.** Given a polyhedron $K \in \mathbb{R}^n$, a point $x \in K$ is a *basic feasible solution* (bfs) for K if there exist n linearly independent defining constraints for K which x satisfies at equality.

In other words, let $K := \{x \in \mathbb{R}^n \mid Ax \leq b\}$, where the m constraints corresponding to the m rows of A are denoted by $a_i \cdot x \leq b_i$. Then $x^* \in \mathbb{R}^n$ is a basic feasible solution if there exist n linearly independent constraints for which $a_i \cdot x^* = b_i$, and moreover $a_i \cdot x^* \leq b_i$ for all other constraints (because x^* must belong to K, and hence satisfy all other constraints as well). Note there are only $\binom{m}{n}$ basic feasible solutions for K, where m is the total number of constraints and n is the dimension.

As you may have guessed by now, these three definitions are all related. In fact, they are all equivalent.

Fact 9.8. Given a polyhedron K and a point $x \in K$, the following are equivalent:

- x is a basic feasible solution,
- x is an extreme point, and
- x is a vertex.

While we do not prove it here, you could try to prove it yourself, or consult a textbook. For now, we proceed directly to the main fact we need for this section.

Fact 9.9. For a polytope K and a linear program $LP := \min\{c \cdot x \mid x \in K\}$, there exists an optimal solution $x^* \in K$ such that x^* is an extreme point/vertex/bfs of K.

Observe that we claimed Fact 9.9 for LPs whose feasible region is a polytope, since that suffices for today, but it can be proven with weaker conditions. However it is not true for all LPs: e.g., the LP in (9.1) has an infinite number of optimal solutions, none of which are at vertices.

This fact suggests an algorithm for LPs when K is a polytope: simply find all of the (at most $\binom{m}{n}$ basic feasible solutions and pick the one that gives the minimum solution value. Of course, there are more efficient algorithm to solve linear programs; we will talk about them in a later chapter. However, let us state a theorem—a very restricted form of the general result—about LP solving that will suffice for now:

Theorem 9.10. *There exist algorithms that take any LP* $\min\{c \cdot x \mid Ax = b, x \geq 0, x \in \mathbb{R}^n\}$*, where both the constraint matrix* A *and the RHS* b *have entries in* $\{0,1\}$ *and* $\text{poly}(n)$ *rows, and output a basic feasible solution to this LP in* $\text{poly}(n)$ *time.*

We will see a sketch of the proof in a later chapter. Discuss the dependence on the number of bits to represent c? Or make this an informal theorem?

9.1.3 *Convex Hulls and an Alternate Representation*

The next definition allows us to give another representation of polytopes:

Definition 9.11 (Convex Hull)**.** Given $x_1, x_2, \ldots, x_N \in \mathbb{R}^n$, the *convex hull* of $x_1, \ldots, x_N$ is the set of all convex combinations of these points. In other words, $\text{CH}(x_1, \ldots, x_N)$ is defined as

$$\left\{ x \in \mathbb{R}^n \;\middle|\; \exists \lambda_1, \ldots, \lambda_N \geq 0 \text{ s.t. } \sum_{i=1}^{N} \lambda_i = 1 \text{ and } x = \sum_{i=1}^{N} \lambda_i x_i \right\}. \quad (9.2)$$

Put yet another way, the convex hull of $x_1, \ldots, x_N$ is the intersection of all convex sets that contain $x_1, \ldots, x_N$. It follows from the definition that the convex hull of finitely many points is a polytope. (Check!) We also know the following fact:

Fact 9.12. Given a polytope K with extreme points $\text{ext}(K)$,

$$K = \text{CH}(\text{ext}(K)).$$

The important insight that polytopes may be represented in terms of their extreme points, or their bounding half-planes. One representation may be easier to work with than the other, depending on the situation. The rest of this chapter will involve moving between these two methods of representing polytopes.

9.2 *Weighted Matchings in Bipartite Graphs*

While the previous chapters focused on finding maximum matchings in graphs, let us now consider the problem of finding a minimum-weight perfect matching in a graph with edge-weights. As before, we start with bipartite graphs, and extend our techniques to general graphs.

We are given a bipartite graph $G = (L, R, E)$ with edge-weights w_e. We want to use linear programs to solve the problem, so it is natural to have a variable x_e for each edge e of the graph. We want our solution to set $x_e = 1$ if the edge is in the minimum-weight

perfect matching, and $x_e = 0$ otherwise. Compactly, this collection of variables gives us a $|E|$-dimensional vector $x \in \mathbb{R}^{|E|}$, that happens to contain only zeros and ones.

A bit of notation: for any subset $S \subseteq E$, let $\chi_S \in \{0,1\}^{|E|}$ denote the *characteristic vector* of this subset S, where χ_S has ones in coordinates that correspond to elements in S, and zeros in the other coordinates.

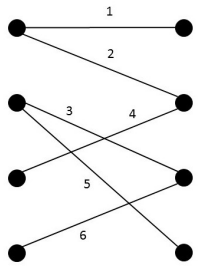


Figure 9.4: This graph has one perfect matching M: it contains edges 1, 4, 5, and 6, represented by the vector $\chi_M = (1,0,0,1,1,1)$.

9.2.1 Goal: the Bipartite Perfect Matching Polytope

It is conceptually easy to define an $|E|$-dimensional polytope whose vertices are precisely the perfect matchings of G: we simply define

$$C_{PM(G)} = CH(\{\chi_M \mid M \text{ is a perfect matching in } G\}). \tag{9.3}$$

And now we get a linear program that finds the minimum-weight perfect matching in a bipartite graph.

$$\min\{w \cdot x \mid x \in C_{PM(G)}\}.$$

By Fact 9.9, there is an optimal solution at a vertex of $C_{PM(G)}$, which by construction represents a perfect matching in G.

The good part of this linear program is that its feasible region has (a) only integer extreme points, (b) which are in bijection with the objects we want to optimize over. So optimizing over this LP will immediately solve our problem. (We can assume that there are linear program solvers which always return an optimal vertex solution, if one exists.) Moreover, the LP solver runs in time polynomial in the size of the LP.

The catch, of course, is that we have no control over the size of the LP, as we have written it. Our graph G may have an exponential number of matchings, and hence the definition of $C_{PM(G)}$ given in (9.3) is too unwieldly to work with. Of course, the fact that there are an exponential number of vertices does not mean that there cannot be a smaller representation using half-spaces. Can we find a compact way to describe $C_{PM(G)}$?

The unit cube

$$K = \{x \in \mathbb{R}^n \mid 0 \leq x_i \leq 1 \ \forall i\}$$

is a polytope with $2n$ constraints but 2^n vertices.

9.2.2 A Compact Linear Program

The beauty of the bipartite matching problem is that the "right" linear program is perhaps the very first one you may write. Here is the definition of the polytope using linear constraints:

$$K_{PM(G)} = \left\{ x \in \mathbb{R}^{|E|} \text{ s.t. } \begin{cases} \sum_{r \in N(l)} x_{lr} = 1 & \forall l \in L \\ \sum_{l \in N(r)} x_{lr} = 1 & \forall r \in R \\ x_e \geq 0 & \forall e \in E \end{cases} \right\}$$

The constraints of this polytope merely enforce that each coordinate is non-negative (which gives us $|E|$ constraints), and that the x_e values of the edges leaving each vertex sum to 1 (which gives us $|L| + |R|$ more constraints). All these constraints are satisfied by each χ_M corresponding to a matching M, which is promising. But it still always comes as a surprise to realize that his first attempt is actually successful:

Theorem 9.13. *For any bipartite graph G, $K_{PM(G)} = C_{PM(G)}$.*

Proof. For brevity, let us refer to the polytopes as K and C. The easy direction is to show that $C \subseteq K$. Indeed, the characteristic vector χ_M for each perfect matching M satisfies the constraints for K. Moreover K is convex, so if it contains all the vertices of C, it contains all their convex combinations, and hence all of C.

For the other direction, we show that an arbitrary vertex x^* of K is contained within C. Using Fact 9.8, we use the fact that x^* is also an extreme point for K. (We can also use the fact that x^* is a basic feasible solution, or that it is a vertex of the polytope, to prove this theorem; we will add the former proof soon, the latter proof appears in §9.3.)

Let $\text{supp}(x^*) = \{e \mid x^*_e > 0\}$ be the support of this solution. We claim that $\text{supp}(x^*)$ is acyclic. Indeed, suppose not, and cycle $C = e_1, e_2, \ldots, e_k$ is contained within the support $\text{supp}(x^*)$. Since the graph is bipartite, this is an even-length cycle. Define

$$\varepsilon := \min_{e \in \text{supp}(x^*)} x^*_e.$$

Observe that for all $e_i \in C$, $x^*_{e_i} + x^*_{e_{i+1}} \leq 1$, so $x^*_{e_i} \leq 1 - \varepsilon$. And of course $x^*_{e_i} \geq \varepsilon$, merely by the definition of ε. Now consider two solutions x^+ and x^-, where

$$x^+_{e_i} = x^*_{e_i} + (-1)^i \, \varepsilon$$

and

$$x^+_{e_i} = x^*_{e_i} - (-1)^i \, \varepsilon.$$

I.e., the two solutions add and subtract ε on alternate edges; this ensures that both the solutions stay within K. But then $x^* = \frac{1}{2}x^+ + \frac{1}{2}x^-$, contradicting our that x^* is an extreme point.

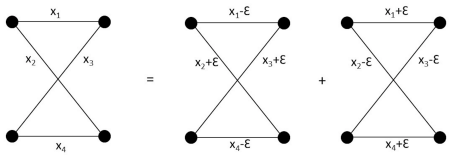


Figure 9.5: There cannot be a cycle in $\text{supp}(x^*)$, because this violates the assumption that x^* is an extreme point.

Therefore there are no cycles in $\text{supp}(x^*)$; this means the support is a forest. Consider a leaf vertex v in the support. Then, by the equality constraint at v, the single edge $e \in \text{supp}(x^*)$ leaving v must have $x^*_e = 1$. But this edge $e = uv$ goes to another vertex u; because x^* is in K, this vertex u cannot have other edges in $\text{supp}(x^*)$ without violating its equality constraint. So u and v are a matched pair in x^*. Now remove u and v from consideration. We have introduced no cycles into the remainder of $\text{supp}(x^*)$, so we may perform the same step inductively to show that x^* is the indicator of a perfect matching, and hence $x^* \in C$. This means all vertices of K are in C, which proves $C \subseteq K$, and completes the proof. □

This completes the proof that the polytope $K_{PM(G)}$ exactly captures precisely the perfect matchings in G, despite having such a simple description. Now, using the fact that the linear program

$$\min\{w \cdot x \mid x \in K_{PM(G)}\}$$

can be solved in polynomial time, we get an efficient algorithm for finding minimum-weight perfect matching in graphs.

9.2.3 *A Proof via Basic Feasible Solutions*

Here is how to prove Theorem 9.13 using the notion of basic feasible solutions (bfs). Suppose $x^* \in \mathbb{R}^{|E|}$ is a bfs: we now show that $x^*_e \in \{0,1\}$ for all edges. By the definition of a bfs, there is a collection of $|E|$ tight linearly independent constraints that define x^*. These constraints are of two kinds: the degree constraints $\sum_{e \in \partial(v)} x_e = 1$ for some subset S of vertices, and the non-negativity constraints $x_e \geq 0$ for some subset $E' \subseteq E$ be edges. (Hence we have $|E'| + |S| = |E|$.)

By reordering columns and rows, we can put the degree constraints at the top, and put all the edges in E' at the end, to get that x^* is defined by:

$$\begin{bmatrix} C & C' \\ 0 & I \end{bmatrix} x^* = \begin{bmatrix} \mathbf{1}_s \\ \mathbf{0}_{m-s} \end{bmatrix}$$

where $C \in \{0,1\}^{s \times s}$, $C' \in \{0,1\}^{(m-s) \times s}$, and $m = |E|$ and $s = |S|$. The edges in E' have $x^*_e = 0$, so consider edges in $E \setminus E'$. By the linear independence of the constraints, we have C being non-singular, so

$$x^*|_{E \setminus E'} = C^{-1}(\mathbf{1} - C' x^*|_{E'}) = C^{-1}\mathbf{1}.$$

By Cramer's rule,

$$x^*_e = \frac{\det(C[\mathbf{1}]_i)}{\det(C)}.$$

The numerator is an integer (since the entries of C are integers), so showing $\det(C) \in \{\pm 1\}$ means that x^*_e is an integer.

Claim 9.14. Any $k \times k$-submatrix of C has determinant in $\{-1, 0, 1\}$.

Proof. The proof is by induction on k; the base case is trivial. If the submatrix D has a column with a single 1, we can expand using that entry, and use the inductive hypothesis. Else each column of D has two non-zeros. Recall that the columns of D correspond to some edges E_D in $E \setminus E'$, and the rows correspond to vertices S_D in S—two non-zeros in each column means each edge in E_D has both endpoints in S_D. Now if we sum rows for vertices in $S_D \cap L$ would give the all ones vector, as will summing up rows for vertices in $S_D \cap R$. (Here is the only place we're using bipartiteness.) In this case $\det(D) = 0$. □

Using the claim and using the fact C is non-singular and hence $\det(C)$ cannot be zero, we get that the entries of x^*_e are integers. By the structure of the LP, the only integers possible in a feasible solution are $\{0, 1\}$ and the vector x^* corresponds to a matching.

9.2.4 *Minimum-Weight Matchings*

How can we we find a minimum-weight (possibly non-perfect) matching in a bipartite graph G? If the edge weights are all non-negative, the empty matching would be the solution—but what if some edge weights are negative? (In fact, that's how we would find a maximum-weight matching–by negating all the weights.) As before, we can define the matching polytope for G as

$$C_{Match(G)} = CH(\{\chi_M \mid M \text{ is a matching in } G\}).$$

To write a compact LP that describes this polytope, we slightly modify our linear constraints as follows:

$$K_{Match} = \left\{ x \in \mathbb{R}^{|E|} \text{ s.t. } \begin{cases} \sum_{j \in R} x_{ij} \leq 1 & \forall i \in L \\ \sum_{j \in L} x_{ji} \leq 1 & \forall i \in R \\ x_{i,j} \geq 0 & \forall i, j \end{cases} \right\}$$

We leave it as an exercise to apply the techniques used in Theorem 9.13 to show that the vertices of K_{Match} are matchings of G, and hence the following theorem:

Theorem 9.15. *For any bipartite graph G, $K_{Match} = CH_{Match}$.*

9.3 *Another Perspective: Buyers and sellers*

The results of the previous section show that the bipartite perfect matching polytope is integral, and hence the max-weight perfect

matching problem on bipartite graphs can be be solved by "simply" solving the LP and getting a vertex solution. But do we need a generic linear program solver? Can we solve this problem faster? In this section, we develop (a variant of) the Hungarian algorithm that finds an optimal solutions using a "combinatorial" algorithm. This proof also shows that any vertex of the polytope $K_{PM(G)}$ is integral, and hence gives another proof of Theorem 9.13.

9.3.1 *The Setting: Buyers and Items*

Consider the setting with a set B with n *buyers* and another set I with n *items*, where buyer b has *value* v_{bi} for item i. The goal is to find a max-value perfect matching, that matches each buyer to a distinct item and maximizes the sum of the values obtained by this matching.

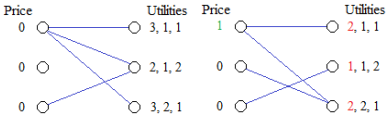
Our algorithm will maintain a set of *prices* for items: each item i will have price p_i. Given a price vector $p := (p_1, \ldots, p_n)$, define the *utility* of item i to buyer b to be

$$u_{bi}(p) := v_{bi} - p_i.$$

Intuitively, the utility measures how favorable it is for buyer b to buy item i, since it factors in both the value and the price of the item. We say that buyer b *prefers* item i if item i gives the highest utility to buyer b, among all items. Formally, buyer $b \in B$ *prefers* item i at prices p if $i \in \arg\max_{i' \in I} u_{bi'}(p)$. The *utility* of buyer b at prices p is the utility of this preferred item:

$$u_b(p) := \max_{i \in I} u_{bi}(p) = \max_{i \in I}(v_{bi} - p_i). \tag{9.4}$$

A buyer has at least one preferred item, and can have multiple preferred items, since there can be ties. Given prices p, we build a *preference graph* $H = H(p)$, where the vertices are buyers B on the left, items I on the right, and where bi is an edge if buyer b prefers item i at prices p. The two examples show preference graphs, where the second graph results from an increase in price of item 1. Flip the figure.



Theorem 9.16. *For any price vector p^*, if the preference graph $H(p^*)$ contains a perfect matching M, then M is a max-value perfect matching.*

Proof. This proof uses weak linear programming duality. Indeed, recall the linear program we wrote for the bipartite perfect matching problem: we allow fractional matchings by assigning each edge bi a

fractional value $x_{bi} \in [0,1]$.

$$\begin{aligned}
\text{maximize} \quad & \sum_{bi} v_{bi} x_{bi} \\
\text{subject to} \quad & \sum_{b=1}^{n} x_{bi} = 1 && \forall i \\
& \sum_{i=1}^{n} x_{bi} = 1 && \forall b \\
& x_{bi} \geq 0 && \forall (b,i)
\end{aligned}$$

The perfect matching M is clearly feasible for this LP, so it remains to show that it achieves the optimum. Indeed, we show this by exhibiting a feasible dual solution with value $\sum_{bi \in M} v_{bi}$, the value of the primal solution. Then by weak duality, both these solutions must be optimal.

The dual linear program is the following:

$$\begin{aligned}
\text{minimize} \quad & \sum_{i=1}^{n} p_i + \sum_{b=1}^{n} u_b \\
\text{subject to} \quad & p_i + u_b \geq v_{bi} && \forall bi
\end{aligned}$$

(Observe that u and p are unconstrained variables.) In fact, given any settings of the p_i variables, the u_b variables that minimize the objective function, while still satisfying the linear constraints, are given by $u_b := \max_{i \in I}(v_{bi} - p_i)$, exactly matching (9.4). Hence, the dual program can then be rewritten as the following (non-linear, convex) program with no constraints:

$$\min_{p=(p_1,\ldots,p_n)} \quad \sum_{i \in I} p_i + \sum_{b \in B} u_b(p).$$

Consider the dual solution given by the price vector p^*. Recall that M is a perfect matching in the preference graph $H(p^*)$, and let $M(i)$ be the buyer matched to item i by it. Since $u_{M(i)}(p) = v_{M(i)i} - p_i$, the dual objective is

$$\sum_{i \in I} p_i^* + \sum_{b \in B} u_b(p^*) = \sum_{i \in I} p_i^* + \sum_{i \in I} (v_{M(i)i} - p_i) = \sum_{bi \in M} v_{bi}.$$

Since the primal and dual values are equal, the primal matching M must be optimal. □

Prices $p = (p_1, \ldots, p_n)$ are said to be *market-clearing* if each item can be assigned to some person who has maximum utility for it at these prices, subject to the constraints of the problem. In our setting, having such prices are equivalent to having a perfect matching in the preference graph. Hence, Theorem 9.16 shows that market-clearing prices give us an optimal matching, so our goal will be to find such prices.

9.3.2 *The Hungarian algorithm*

The "Hungarian" algorithm uses the buyers-and-sellers viewpoint from the previous section. The idea of the algorithm is to iteratively change item prices as long as they are not market-clearing, and the key is to show that this procedure terminates. To make our proofs easier, we assume for now that all the values v_{bi} are integers.

The price-changing algorithm proceeds as follows:

1. Initially, all items have price $p_i = 0$.
2. In each iteration, build the current preference graph $H(p)$. If it contains a perfect matching M, return it. Theorem 9.16 ensures that M is an optimal matching.
3. Otherwise, by Hall's theorem, there exists a set S of buyers such that if
$$N(S) := \{i \in I \mid \exists b \in S, bi \in E(H(p))\}$$
is the set of items preferred by at least one buyer in S, then $|N(S)| < |S|$. ($N(S)$ is the *neighborhood* of S in the preference graph.) Intuitively, we have many buyers trying to buy few items, so logically, the sellers of those items should raise their prices! The algorithm increases the price of every item in $N(S)$ by 1, and starts a new iteration by going back to step 2.

H.W. Kuhn (1955)

The algorithm was named the Hungarian algorithm by Harold Kuhn who based his ideas on the works of Jenö Egervary and Dénes König. Munkres subsequently showed that the algorithm was in fact implementable in $O(n^3)$. Later, the algorithm was found to have been proposed even earlier by Carl Gustav Jacobi, before 1851.

That's it. Running the algorithm on our running example gives the prices on the right.

The only way the algorithm can stop is to produce an optimal matching. So we must show it does stop, for which we use a "semi-invariant" argument. We keep track of the "potential"

$$\Phi(p) := \sum_i p_i + \sum_b u_b(p),$$

where p_i are the current prices and $u_b(p) = \max_i(v_{bi} - p_i)$ as above. This is just the dual value, and hence is is lower-bounded by the *optimal value* of the dual program. (We assume the optimal value of the LP is finite, e.g., if all the input values are finite.) Then, it suffices to prove the following:

Lemma 9.17. *Every time we increase the prices in $N(S)$ by 1, the value of $\sum_i p_i + \sum_b u_b$ decreases by at least 1.*

Proof. The value of $\sum_i p_i$ increases by $|N(S)|$, because we increase the price of each item $i \in N(S)$ by 1. For each buyer $b \in S$, the value u_b must decrease by 1, since all their preferred items had their prices increased by 1, and all other items previously had utilities at least one lower than the original $u_b(p)$. (Here, we used the fact

that all values were integral.) Therefore, the value of the potential $\sum_i p_i + \sum_b u_b$ changes by $|N(B)| - |B| \leq -1$. □

Hence the algorithm stops in finite time, and produces a maximum-value perfect matching. By the arguments above **??** we get yet another proof of integrality of the LP **??** for the bipartite pefect matching problem. A few other remarks about the algorithm:

- In fact, one can get rid of the integrality assumption by raising the prices by the maximum amount possible for the above proof to still go through, namely

$$\min_{b \in S} \left(u_b(p) - \max_{i \notin N(S)} (v_{ib} - p_i) \right).$$

  It can be shown that this update rule makes the algorithm stop in only $O(n^3)$ iterations.

- If all the values are non-negative, and we don't like the utilities to be negative, then we can do one of the following things: (a) when all the prices become non-zero, subtract the same amount from all of them to make the lowest price hit zero, or (b) choose S to be a minimal "consticted" set and raise the prices for $N(S)$. This way, we can ensure that each buyer still has at least one item which gives it nonngegative utility. (Exercise!)

- Suppose there are n buyers and a single item, with all non-negative values. (Imagine there are $n-1$ dummy items, with buyers having zero values for them.) The above algorithm behaves like the usual ascending-price English or Vickery auction, where prices are raised until only one bidder remains. Indeed, the final price for the "real" item will be such that the second-highest bidder is indifferent between it and a dummy item.

  This is a more general phenomenon: indeed, even in the setting with multiple items, the final prices are those produced by the Vickery-Clarke-Groves truthful mechanism, at least if we use the version of the algorithm that raises prices on minimal constricted sets. The truthfulness of the mechanism means there is no incentive for buyers to unilaterally lie about their values for items. See, e.g., [1] for the rich connection of matching algorithms to auction theory and (algorithmic) mechanism design.

Check about negative values, they don't seem to matter at all, as long as everything is finite. What about max-weight maximum matching: we can always convert the graph, but does the algorithm work out of the box?

This proof shows that for any setting of values, there is an optimal integer solution to the linear program

$$\max\{v \cdot x \mid x \in K_{LP(G)}\}.$$

This implies that every vertex x^* of the polytope $K_{LP(G)}$ is integral—indeed, the definition of vertex means x^* is the unique solution to the linear program for some values v, and our proof just produced an integral matching that is the optimal solution. Hence, we get another proof of Theorem 9.13, this time using the notion of vertices instead of extreme points.

9.4 *A Third Algorithm: Shortest Augmenting Paths*

Let us now see yet another algorithm for solving weighted matching problems in bipartite graphs. For now, we switch from maximum-weight matchings to minimum-weight matchings, because they are conceptually cleaner to explain here. Of course, the two problems are equivalent, since we can always negate edges.

In fact, we solve a min-cost max-flow problem here: given an flow network with terminals s and t, edge capacities u_e, and also edge costs/weights w_e, find an s-t flow with maximum flow value, and whose total cost/weight is the least among all such flows. (Moreover, if the capacities are integers, the flow we find will also have integer flow values on all edges.) Casting the maximum-cardinality bipartite matching problem as a integer max-flow problem, as in §blah gives us a minimum-weight bipartite matching.

This algorithm uses an augmenting path subroutine, much like the algorithm of Ford and Fulkerson. The subroutine, which takes in a matching M and returns one of size $|M| + 1$, is presented below. Then, we can start with the empty matching and call this subroutine until we get a maximum matching.

Let the original bipartite graph be G. Construct the **directed** graph G_M as follows: For each edge $e \in M$, insert that edge directed from right to left, with weight $-w_e$. For each edge $e \in G \backslash M$, insert that edge directed from left to right, with weight w_e. Then, compute the shortest path P that starts from the left and ends on the right, and return $M \triangle P$. It is easy to see that $M \triangle P$ is a matching of size $|M| + 1$, and has total weight equal to the sum of the weights of M and P.

Call a matching M an extreme matching if M has minimum weight among all matchings of size $|M|$. The main idea is to show that the above subroutine preserves extremity, so that the final matching must be extreme and therefore optimal.

Theorem 9.18. *If M is an extreme matching, then so is $M \triangle P$.*

Proof. Suppose that M is extreme. We will show that there exists an augmenting path P such that $M \triangle P$ is extreme. Then, since the algorithm finds the shortest augmenting path, it will find a path that is no longer than P, so the returned matching must also be extreme.

Consider an extreme matching M' of size $|M| + 1$. Then, the edges in $M \triangle M'$ are composed of disjoint paths and cycles. Since $M \triangle M'$ has more edges in M' than edges in M, there is some path $P \subset M \triangle M'$ with one more edge in M' than in M. This path necessarily starts and ends on opposite sides, so we can direct it to start from the left and end on the right. We know that $|M' \cap P| = |M \cap P| + 1$, which means that $M \backslash P$ and $M' \backslash P$ must have equal size. The total weight of $M \backslash P$ and $M' \backslash P$ must be the same, since otherwise, we can swap the two matchings and improve one of M and M'. Therefore, $M \triangle P = (M' \cap P) \cup (M \backslash P)$ has the same weight as M' and is extreme. □

Note that the formulation of G_M is exactly the graph constructed if we represent the minimum matching problem as a min-cost flow. Indeed, the previous theorem can be generalized to a very similar statement for the augmenting path algorithm for min-cost flows.

9.5 *Perfect Matchings in General Graphs*

Interestingly, Theorem 9.13 is false for non-bipartite graphs. Indeed, consider graph K_3 which consists of a single 3-cycle: this graph has no perfect matching, but setting $x_e = 1/2$ for each edge satisfies all the constraints. This suggests that the linear constraints defining $K_{PM(G)}$ are not enough, and we need to add more constraints to capture the convex hull of perfect matchings in general graphs.

In situations like this, it is instructive to look at the counter-example, to see what constraints must be satisfied by any integer solution, but are violated by this fractional solution. For a set of vertices $S \subseteq V$, let $\partial(S)$ denote the edges leaving S. Here is one such set of constraints:

$$\left\{ \sum_{e \in \partial(S)} x_e \geq 1 \qquad \forall S \subseteq V \text{ such that } |S| \text{ is odd}, \qquad \right\}.$$

These constraints say: the vertices belonging to a set $S \subseteq V$ of odd size cannot be perfectly matched within themselves, and at least one edge from any perfect matching must leave S. Indeed, this constraint would be violated by $S = V(K_3)$ in the example above.

Adding these *odd-set constraints* to the previous degree constraints gives us the following polytope, which was originally proposed by Edmonds:

J. Edmonds. *Maximum matching and a polyhedron with 0,1-vertices* (1965)

$$K_{\text{genPM(G)}} = \left\{ x \in \mathbb{R}^{|E|} \text{ s.t. } \begin{cases} \sum_{u \in \partial(v)} x_{vu} = 1 & \forall v \in V \\ \sum_{e \in \partial(S)} x_e \geq 1 & \forall S \text{ s.t. } |S| \text{ odd} \\ x_e \geq 0 & \forall e \in E \end{cases} \right\}$$

Theorem 9.19. *For an undirected graph G, we have*

$$K_{genPM(G)} = C_{genPM(G)},$$

where $C_{genPM(G)}$ the convex hull of all perfect matchings of G.

One proof is a more sophisticated version of the one in §9.2.3, where we may now have tight odd-set constraints; we leave it as a slightly challenging exercise.

This LP potentially contains a exponential number of constraints, in contrast with the linear number of constraints needed for the bipartite case. In fact, a powerful theorem by Thomas Rothvoß (building on work by Mihalis Yannakakis) shows that any polytope whose vertices are the perfect matchings of the complete graph on n vertices must contain an exponential number of constraints.

T. Rothvoß (2014,2017)
M. Yannakakis (1991)

Given this negative result, this LP seems useless: we cannot even write it down explicitly in polynomial time! It turns out that despite this large size, it is possible to solve this LP in polynomial time. In a later lecture, we will see the Ellipsoid method to solve linear programs. This method can solve such a large LP, provided we give it a helper procedure called a "separation oracle", which, given a point $x \in \mathbb{R}^{|E|}$, outputs YES if x lies is within the desired polytope, and otherwise it outputs NO and returns a violated constraint of this LP. It is easy to check if x satisfies the degree constraints, so the challenge here is to find an odd set S with $\sum_{e \in \partial(S)} x_e < 1$, if there exists one. Such an algorithm can be indeed obtained using a sequence of min-cut computations in the graph, as was shown by. We will see this in a HW problem later in the course.

M.W. Padberg and M.R. Rao (1982)

9.6 *Integrality of Polyhedra*

We just saw several proofs that the bipartite perfect matching polytope has a compact linear program. Moreover, we claimed that the pefect matching polytope on general graphs has an explicit linear program that, while exponential-sized, can be solved in polynomial time. Such results allow us to solve the weighted bipartite matching problems using generic linear programming solvers (as long as they return vertex solutions).

Having many different ways to view a problem gives us a deeper insight, and thereby come up with faster and better ways to solve it. Moreover, these different perspectives give us a handle into solving extensions of these problems. E.g., if we have a matching problem with two different kinds weights w_1 and w_2 on the edges: we want to find a matching $x \in K_{PM(G)}$ minimizing $w_1 \cdot x$, now subject to the additional constraint $w_2 \cdot x \leq B$. While the problem is now NP-hard, this linear constraint can easily be added to the linear program to get a fractional optimal solution. Then we can reason about how to "round" this solution to get a near-optimal matching.

We now show how two problems we considered earlier, namely minimum-cost arborescence and spanning trees, can be exactly modeled using linear programs. We then conclude with a pointer to a general theory of integral polyhedra.

9.6.1 *Arborescences*

We already saw a linear program for the min-weight r-arborescence polytope in §2.3.2: since each node that is not the root r must have a path in the arborescence to the root, it is natural to say that for any subset of vertices $S \subseteq V$ that does not contain the root, there must be an edge leaving it. Specifically, given the digraph $G = (V, A)$, the polytope can be written as

$$K_{Arb(G)} = \left\{ x \in \mathbb{R}^{|A|} \text{ s.t. } \begin{cases} \sum_{a \in \partial^+(S)} x_a \geq 1 & \forall S \subset V s.t. r \notin S \\ x_a \geq 0 & \forall a \in A \end{cases} \right\}.$$

Here $\partial^+(S)$ is the set of arcs that leave set S. The proof in §2.3.2 already showed that for each weight vector $w \in \mathbb{R}^{|A|}$, we can find an optimal solution to the linear program $\min\{w \cdot x \mid x \in K_{Arb(G)}\}$.

9.6.2 *Minimum Spanning Trees*

One way to write an LP for minimum spanning trees is to reduce it to minimum-weight r-arborescences: indeed, replace each edge by two arcs in opposite directions, each having the same cost. Pick any node as the root r. Observe the natural bijection between r-arborescence in this digraph and spanning trees in the original graph, having the same weight.

But why go via arborescences? Why not directly model the fact that any tree has at least one undirected edge crossing each cut $(S, V \setminus S)$, perhaps as follows:

$$K_{STtry} = \left\{ x \in \mathbb{R}^{|E|} \text{ s.t. } \begin{cases} \sum_{e \in \partial(S)} x_e \geq 1 & \forall S \subset V, S \neq \emptyset, V \\ x_e \geq 0 & \forall e \in E \end{cases} \right\}.$$

(The first constraint excludes the case where S is either empty or the entire vertex set.) Sadly, this does not precisely capture the spanning tree polytope: e.g., for the familiar cycle graph having three vertices, setting $x_e = 1/2$ for all three edges satisfies all the constraints. If all edge weights are 1, this solution get a value of $\sum_e x_e = 3/2$, whereas any spanning tree on 3 vertices must have 2 edges.

One can indeed write a different linear program that captures the spanning tree polytope, but it is a bit non-trivial:

$$K_{ST(G)} = \left\{ x \in \mathbb{R}^{|E|} \text{ s.t. } \begin{cases} \sum_{ij \in E : i,j \in S} x_{ij} \leq |S| - 1 & \forall S \subseteq V, S \neq \emptyset \\ \sum_{ij \in E} x_{ij} = |V| - 1 & \\ x_{ij} \geq 0 & \forall ij \in E \end{cases} \right\}$$

Define the convex hull of all minimum spanning trees of G to be CH_{MST}. Then, somewhat predictably, we will again find that $CH_{MST} = K_{MST}$.

Both the polytopes for arborescences and spanning trees had exponentially many constraints. Again, we can solve these LPs if we are given separation oracles for them, i.e., procedures that take x and check if it is indeed feasible for the polytope. If it is not feasible, the oracle should output a violated linear inequality. We leave it as an exercise to construct separation oracles for the polytopes above.

A different approach is to represent such a polytope K compactly via an *extended formulation*: i.e., to define a polytope $K' \in \mathbb{R}^{n+n'}$ using a polynomial number of linear contraints (on the original variables $x \in \mathbb{R}^n$ and perhaps some new variables $y \in \mathbb{R}^{n'}$) such that projecting K' down onto the original n-dimensions gives us K. I.e., we want that

$$K = \{x \in \mathbb{R}^n \mid \exists y \in \mathbb{R}^{n'} s.t.\ (x, y) \in K'\}.$$

The homework exercises will ask you to write such a compact extended formulation for the arborescence problem.

9.6.3 *Integrality of Polyhedra and Total Unimodularity*

This section still needs work. We have seen that LPs are a powerful way of formulating problems like min-cost matchings, min-weight r-aborescences, and MSTs. We reasoned about the structure of the polytopes that underly the LPs, and we were able to show that these LPs do indeed solve their combinatorial problems. But notice that simply forming the LP is not sufficient–significant effort was expended to show that these polytopes do indeed have integer solutions at the vertices. Without this guarantee, we could get fractional

When we say "integer solutions", we mean the solution vector is integer valued

solutions to these LPs that do not actually give us solutions to our problem.

There is a substantial field of study concerned with proving the integrality of various LPs. We will briefly introduce a matrix property that implies the integrality of corresponding LPs. Recall that an LP can be written as

$$[A]_{m\times n} \cdot \vec{x} \leq \vec{b},$$

where A is a $m \times n$ matrix with each row corresponding to a constraint, $\vec{x}$ is a vector of n variables, and $\vec{b} \in \mathbb{R}^m$ is a vector corresponding to the m scalars $b_i \in \mathbb{R}$ in the constraint $A^{(i)} \cdot \vec{x} \leq b_i$.

Definition 9.20. A matrix $[A]_{m\times n}$ is called *totally unimodular* if every square submatrix B of A has the property that $\det(B) \in \{0, \pm 1\}$

We then have the following neat theorem, due to Alan Hoffman and Joe Kruskal:

Theorem 9.21 (Hoffman and Kruskal Theorem)**.** *If the constraint matrix $[A]_{m\times n}$ is totally unimodular and the vector $\vec{b}$ is integral, i.e., $\vec{b} \in \mathbb{Z}^m$, then the vertices of the polytope induced by the LP are integer valued.*

Moreover, if for some matrix A the polytope has integer vertices for all integer vectors b, then the matrix A is totally unimodular.

A.J. Hoffman and J.B. Kruskal (1956)

Proof. (Sketch) This proof uses that solutions to linear systems can be obtained using Cramer's rule. □

Thus, to show that the vertices are indeed integer valued, one need not go through producing combinatorial proofs, as we have. Instead, one could just check that the constraint matrix A is totally unimodular. Here's a nice presentation by Marc Uetz about the relation between total unimodularity and graph matchings.