Chapter 1

Zachary Campbell

March 27, 2018

1 Introduction

This thesis is aimed at those who have taken a course in algorithms, especially one in which network flows, and the max-flow/min-cut theorem were discussed. In this introductory chapter I will review these topics more briefly, and discuss their relevance to this thesis. We are not discussing flows or cuts in this thesis (although much of the work done in this thesis has a shifted frame of reference via flows), but their relevance will be made clear. The most important thing for the reader to have is a curiosity about the max-flow/min-cut theorem; there is a lot of cool math behind it, and we demonstrate that math in other settings. Specifically, we look at this relationship in weighted bipartite matchings.

2 Bipartite graphs and matchings

Throughout this thesis we will be interested in a specific subclass of graphs known as bipartite graphs. Unless otherwise noted, our algorithms will assume a bipartite structure.

Definition (Bipartite graph). A *bipartite graph* is a graph whose vertices can be partitioned into two sets U and V such that all edges connect a vertex $u \in U$ to a vertex $v \in V$. We will denote this graph $G = (U, V, E)$, where E is the edge set $(U \times V)$.

In Figure 1 we have a bipartite graph with vertex partition given by $U = \{A, B, C, D\}$ and $V = \{E, F, G\}$. All edges in this graph are between a node $u \in U$ and a node $v \in V$. The graph on the right is also bipartite. It may take a little more time to convince yourself that you can partition

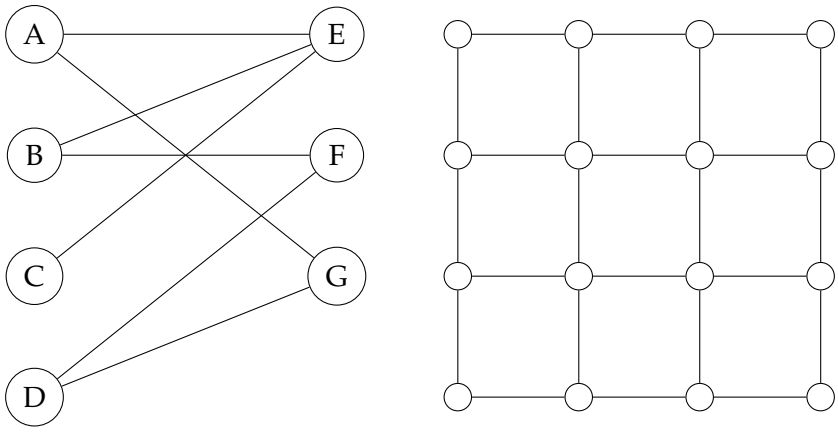


Figure 1: Examples of bipartite graphs

the vertices into disjoint U and V in a way that maintains the bipartite property. Try it! Now that we know what we are working with, let's introduce a problem that we'd like to solve on these graphs.

Definition (Matching). Let $G = (U, V, E)$ be a bipartite graph. A subset $M \subset E$ is a *matching* if no two edges in M are incident to the same vertex. We call a matching *perfect* if all vertices are an endpoint of an edge in M.

We say that a vertex $w \in U \cup V$ is *matched* with respect to M if it is an endpoint of some edge in M. The following figure shows some examples of different matchings on one of the graphs from Figure 1.

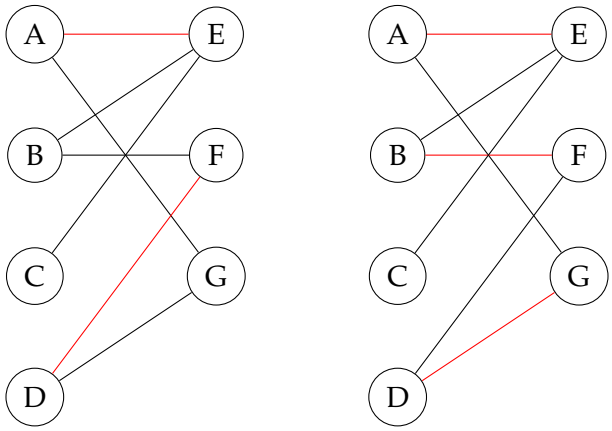


Figure 2: Examples of matchings on a bipartite graph

There many different valid matchings on the graph in Figure 2. Oftentimes, we want to find the largest matching on a graph. This is classically motivated by economic examples, where we have

a set of bidders, and a set of goods, and the edges between them denote a bidder i's willingness to pay for good j. For a money-hungry auctioneer, the goal here would be to find the "largest" matching on the graph, i.e. the one that maximizes profit of the auction. We will discuss this interpretation in more depth later on in the thesis. This leads to the following definition.

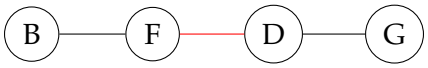
Definition (Maximal matching). A *maximal matching* on G is a matching M such that if any other edge not in M is added to M, it is no longer a valid matching. Alternatively put, M is maximal if there is no matching $M^{'}$ such that $M \subset M^{'}$.

Both matchings in Figure 2 are maximal matchings; in each case there are no edges that we can add to M and have that M is still a matching. However, notice that the size of the matchings is different, even though both are maximal on G. This leads to the following definition.

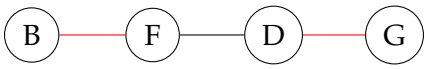
Definition (Maximum matching). A matching M on a graph G is said to be a *maximum* matching if for all other matchings $M^{'}$ on G, $|M^{'}| \leq |M|$.

In our example, the matching on the right given by $M = \{(A, E), (B, F), (D, G)\}$ is a maximum matching (convince yourself). In general there may be many unique maximum matchings on a graph.

In this section we are interested in general methods for finding maximum matchings on bipartite graphs. One of the fundamental approaches is to look at certain subgraphs called alternating paths. Before we define what these are, let's look at a motivating example. Suppose we have the matching on the left in Figure 2. So $M = \{(A, E), (D, F)\}$. Consider the following sequence of vertices in the graph:



Call this sequence p. Let's perform an operation that we will denote $M \oplus p$, which operates like XOR: add to M each edge in p that isn't in M, and remove from M each edge in p that is in M. This gives us the following segment, where (B, F) and (D, G) are now in M, but (D, F) is not:



First, we must check that the new M is still a valid matching; we do this by noticing that B and G were originally unmatched, so it's okay for one of their incident edges to be added. Also, notice

that the size of our matching has grown by 1! In fact, this new matching is exactly the matching given by the graph on the right in Figure 2. This is a general technique in finding maximum matchings. We want to look for these paths that start and end at unmatched vertices, and whose edges are alternately matched and unmatched. If we can find one of these paths, we will be able to increase the size of matching. We define this formally now.

Definition (Alternating path). Let G be a graph and M some matching on G. An *alternating path* is a sequence of vertices and edges that begins with an unmatched vertex, and whose edges alternate between being in M and not in M.

Definition (Augmenting path). An *augmenting path* is an alternating path that starts and ends on unmatched vertices. When we augment M by an augmenting path p, we use the notation $M \oplus p$.

This motivates a general method for finding a maximum matching on a bipartite graph: just keep looking for augmenting paths, and augment the current matching by that augmenting path. Of course, we need to prove that this in fact gives us a maximum matching. The following theorem says exactly that.

Theorem (Berge, 1957). A matching M on G is a maximum matching if and only if G contains no augmenting paths with respect to M.

This gives us the following framework for finding maximum matchings in bipartite graphs.

```
ALG 1(G)
1  M := ∅
2  while there exists an augmenting path p
3        M := M ⊕ p
4  return M
```

Note that we have yet to describe the details of this algorithm. Before we do so, we are going to take a step back a bit and look at the maximum matching problem from a slightly different perspective. In doing so, we will develop a language for talking about this problem that will serve us throughout the rest of this thesis. At first, the approach will appear purely pedagogic, but hopefully the reader will understand the significance of it by the end of the thesis.

3 The vertex cover problem

We begin this section by defininig a new problem. Again, this is a problem on graphs in general, but we will be restricting our attention to bipartite graphs. This is a good idea for many reasons, but the most pertinent reason is that this problem is NP-complete in the general case.

Definition (Vertex cover). Let $G = (U, V, E)$ be a bipartite graph. A subset $C \subset U \cup V$ is said to be a *vertex cover* if for each $(u, v) \in E$ we have that at least one of $u, v \in C$. C is a *minimum* vertex cover if for any other cover $C^{'}$, $|C| \leq |C^{'}|$.

Using what we've already learned, we can specify at least one relation between matchings and vertex covers: namely, the set of all vertices of all edges in any maximal matching on a graph forms a vertex cover. Here are some examples of vertex covers on the graph we've been looking at.

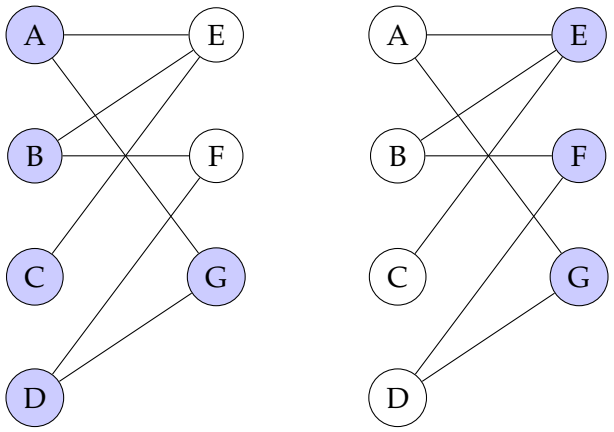


Figure 3: Examples of vertex covers

You can convince yourself that the cover on the right is a minimum cover. This brings us to an important theorem. We first prove a lemma.

Lemma . Let $G = (U, V, E)$ be a bipartite graph. Let M be a matching on G and C a cover on G such that $|M| = |C|$. Then M is a maximum matching and C is a minimum covering.

Proof. Let $M^{'}$ be a maximum matching on G and $C^{'}$ a minimum covering on G. For each $(u, v) \in M^{'}$, $C^{'}$ must include either u or v, which tells us that $|M^{'}| \leq |C^{'}|$. Then we have

$$|M| \leq |M^{'}| \leq |C^{'}| \leq |C|.$$

Thus, if $|M| = |C|$ we have equalities above, which means that the size of a maximum matching is equal to the size of a minimum covering. □

Theorem (Kőnig-Egervary). For any bipartite graph G, if M is a maximum matching on G and C is a minimum vertex cover on G, then $|M| = |C|$.

Before we prove this, we define a simple term: an M-alternating path is an alternating path with respect to a matching M.

Proof. Let $G = (U, V, E)$ be a bipartite graph, and let M be a maximum matching on G. Furthermore, define

$$A := \{s \in S \mid s \text{ unsaturated}\}$$

and

$$B := \{\text{all vertices connected to nodes in } A \text{ by } M\text{-alternating paths}\}.$$

Let $L = B \cap U$ and $R = B \cap V$. Then we have the following:

1. Every node in R is saturated.
2. $N(L) = R$,

where $N(L)$ denotes the set of all vertices connected to elements of L (the "neighbors" of L). The first claim comes from the fact that, if M is a maximum matching, then our alternating paths starting at nodes in A must have length ≥ 2, and must have even length (otherwise we would have an augmenting path, which contradicts our assumption that M is a maximum matching). The second comes from that fact that every node in $N(L)$ is connected to vertices in A by an alternating path.

Now, define $K := (U \setminus L) \cup R$. Every edge in G must have one of its endpoints in K. If not, there there would be an edge with one end in L and one end in $V \setminus R$, which contradicts $N(L) = R$. So K is a covering of G. Moreover, $|K| = |M|$, since for each edge in M we've included one of its endpoints in K (the vertices we've chosen are those in $N(L)$ and those in $U \setminus L$). Thus, by the previous lemma, K is a minimum covering. □

All of this tells us that there is a deep relationship between maximum matchings and minimum vertex covers on bipartite graphs. Given a solution to one, we can turn it into a solution to the other. This is what we seek to accomplish next.

4 Linear programming

The development of combinatorial optimization has been deeply intertwined with the discipline of linear programming. In its most basic form, linear programs are given by some linear objective function that you want to optimize, along with some linear constraint equations. For a more detailed treatment on linear programming, we will refer you to (CITE LP SOURCES). For the purposes of this thesis, we will treat linear programming more casually, only requiring a few key results. Moreover, we will not be discussing methods of actually solving linear programs, for which there are at least a couple of well known but complicated algorithms.

In the general linear-programming problem, our goal is to optimize some linear function that is constrained by a set of linear inequalities. These problems are ubiquitous in applied math and computer science, as they model a system in which something needs to be optimized according to competing resources. We can express a general *maximization* linear program as

$$\text{maximize} \quad \sum_{j=1}^{n} c_j x_j \tag{1}$$

$$\text{subject to} \quad \sum_{j=1}^{n} a_{ij} x_j \leq b_j, \quad i = 1, \ldots, m \tag{2}$$

$$x_j \geq 0, \quad j = 1, \ldots, n. \tag{3}$$

We call the function in (1) our *objective function*, and the linear inequalities (2) and (3) our constraints. Similarly, a *minimization* linear program takes the form

$$\text{minimize} \quad \sum_{j=1}^{n} c_j x_j \tag{4}$$

$$\text{subject to} \quad \sum_{j=1}^{n} a_{ij} x_j \geq b_j, \quad i = 1, \ldots, m \tag{5}$$

$$x_j \geq 0, \quad j = 1, \ldots, n. \tag{6}$$

Many problems which on the face may not appear to be optimization problems turn out to be easily rephrased as linear programs. Our goal in this section will be to describe our two problems, maximum matchings and minimum vertex covers, as linear programs. Before doing so, we describe duality theory, which allows us to draw relationships between certain linear programs. Duality theory gives us a way to prove bounds on optimal solutions to linear programs. We first define the dual of a linear program.

Definition (Dual). Let

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

be our linear program, which we will call the *primal* linear program. Then we define the *dual* of this linear program to be the linear program

$$\begin{aligned} &\text{minimize } \mathbf{b}^T \mathbf{y} \\ &\text{subject to } A^T \mathbf{y} \leq \mathbf{c} \\ &\qquad\qquad\quad \mathbf{y} \geq 0 \end{aligned}$$

The first thing to note is that the dual of the dual is the primal. Let us introduce some notation. First, let us denote the primal maximization problem by the letter Γ, and the dual minimization problem by the letter Ω. For a given linear program, we denote an optimal solution by **OPT**.

Theorem (Weak duality). If the primal linear program (in maximization form) and the dual (in minimization form) are both feasible, then

$$\mathbf{OPT}(\Gamma) \leq \mathbf{OPT}(\Omega).$$

What's surprising is the following theorem.

Theorem (Strong duality). Given two linear programs Γ and Ω that are duals of each other, if one is feasible and bounded, then so is the other. Additionally,

$$\mathbf{OPT}(\Gamma) = \mathbf{OPT}(\Omega).$$

Our goal now is to use this theory to relate the maximum matching problem to the minimum vertex cover problem.

Let's first turn maximum matching into a linear program. Our goal is to maximize the number of edges in our matching. Our constraint is that no edge is incident to more than one edge in the matching. So for each edge (u, v), we will need a corresponding x_{uv}. Our objective function is then pretty simple: maximize the number of x_{uv}. Now we need to figure out our constraint equations. For a fixed node $u \in U$, the number of edges in the matching incident to u is given by $\sum_{v \in V} x_{uv}$. So we want that this is ≤ 1. Similarly, for any node v, we want $\sum_{u \in U} x_{uv} \leq 1$. This gives us the following linear program.

$$\begin{aligned}
\text{maximize} \quad & \sum_{u,v} x_{uv} && && (7)\\
\text{subject to} \quad & \sum_{v} x_{uv} \leq 1, && \forall u \in U, && (8)\\
& \sum_{u} x_{uv} \leq 1, && \forall v \in V, && (9)\\
& x_{uv} \in \{0, 1\}. && && (10)
\end{aligned}$$

What we've given here is an *integer* linear program, since we've restricted our x variables to be integers. In general, solving integer linear programs is NP-hard. However, in this case it is well

know that this linear program attains integer solution at extreme of the polyhedron solution space, so we can drop the integrality requirements and just say $x_{uv} \geq 0$.

Now let's try and construct the dual of this linear program. We will need a variable y_u for each vertex u. Similarly, we need a variable y_v for each vertex v. Our objective will be to minimize over the sum of these y_u, y_v. Since our constraint in the primal is the constant vector 1, our only constraint will be that $y_u + y_v \geq 1$. This gives us the dual linear program

$$\text{minimize} \quad \sum_{u,v} (y_u + y_v) \tag{11}$$

$$\text{subject to} \quad y_u + y_v \geq 1 \quad \forall u, v, \tag{12}$$

$$y_u, y_v \geq 0. \tag{13}$$

This dual problem tells us that each edge must be "covered" by at least one of its incident vertices. This is exactly the vertex cover problem! So for unweighted bipartite graphs, the linear programs for maximum matchings and minimum vertex covers are duals of each other. We will use this insight to construct our algorithms for solving the maximum matching problem.

There is another version of this problem, called the *maximum weight matching*. In this version, we are given a bipartite graph with non-negative edge weights w_{uv} for all edges (u, v). Instead of trying to maximize the number of edges in the matching, the goal is to find a matching M such that $\sum_{(u,v)\in M} w_{uv}$, or the weight of the matching, is greater than the weight of any other matching. We can easily encode this problem by making a slight modification to our linear program from before. The primal is given by

$$\text{maximize} \quad \sum_{u,v} w_{uv} x_{uv} \tag{14}$$

$$\text{subject to} \quad \sum_{v} x_{uv} \leq 1, \quad \forall u \in U, \tag{15}$$

$$\sum_{u} x_{uv} \leq 1, \quad \forall v \in V, \tag{16}$$

$$x_{uv} \geq 0. \tag{17}$$

Then the dual is

$$\text{minimize} \quad \sum_u y_u + \sum_v y_v \tag{18}$$

$$\text{subject to} \quad y_u + y_v \geq w_{uv} \quad \forall u, v, \tag{19}$$

$$y_u, y_v \geq 0. \tag{20}$$

The dual is a sort of weighted vertex cover. One way to think about it is that each edge has a "cost" given by w_{uv}, and each of its endpoints has to pool "money" in order to pay at least that cost. So instead of edges being covered or not covered, there's a certain "amount" that they have to be covered.

In this section, we have only described integer linear programs. In general, solving these is NP-hard. However, we are primarily using these integer linear programs as tools to better understand the structure of the matching problem, and how it relates to the vertex cover problem. We are not developing tools to *solve* these linear programs. We will use these linear programs to motivate the algorithms given in the next section.

5 Revisiting a familiar problem - max-flow/min cut

Here we demonstrate that a common problem discussed in algorithms courses, and a quite amazing theorem, is really a special case of what we've been discussing. Recall that a flow network is defined as follows.

Definition (Flow network). A *flow network* $G = (V, E)$ is a directed graph in which each edge $(u, v) \in E$ has nonnegative *capacity* $c_{uv} \geq 0$. Furthermore, there are two vertices, a source s and a sink t. We assume that every $v \in V$ lies in some path $s \to \cdots \to v \to \cdots \to t$.

Definition (Flow). Let $G = (V, E)$ be a flow network. A *flow* in G is a function $f : V \times V \to \mathbb{R}$ that satisfies the following:

- Capacity constraint: For all $u, v \in V$, $0 \leq f(u, v) \leq c_{uv}$.

- Conservation: For all $u \in V \setminus \{s, t\}$, we have

$$\sum_{v \in V} f(v, u) = \sum_{v \in V} f(u, v).$$

  This says that for all vertices v except our source and sink, the flow out of v is equal to the flow into v.

We define the value of the flow $val(f) = \sum_{v \in V} f(s, v) - \sum_{v \in V} f(v, s)$, which is just the flow out of our source minus the flow into our source.

The problem as demonstrated in a typical algorithms course is to find a maximum flow from s to t. This is done using a method developed by Ford and Fulkerson which does what $ALG1$ does, but in a residual graph (essentially the subgraph where the flows $f(u, v) < c_{uv}$, we refer the reader to [CITE CLRS] for more detail).
Now, we recal the definition of a cut in a flow network.

Definition (Cut). A *cut* (S, T) of a flow network $G = (V, E)$ is a partition of V into sets S and $T = V \setminus S$ such that $s \in S$ and $t \in T$. Given a flow f, the *net flow* $f(S, T)$ across the cut (S, T) is defined as

$$f(S, T) = \sum_{u \in S} \sum_{v \in T} f(u, v) - \sum_{u \in S} \sum_{v \in T} f(v, u).$$

Finally, the *capacity* of the cut is $c(S, T) = \sum_{u \in S} \sum_{v \in T} c(u, v)$. A minimum cut has capacity less than or equal to all other cuts in the network/

One of the coolest and most surpising theorems in an algorithms corse is that, given a flow network G, the value of the maximum flow ($val(f)$) is equal to the capacity of the minimum S, T cut of G.
Our goal now is to build up to this same theorem using the tools developed in this chapter. What follows is due to [CITE VAZIRANI]. We will first give a linear program for the maximum flow problem. To make things simpler, let's introduce an arc of infinite capacity from the sink t to the source s; this converts this to a circulation, with the objective to maximize the flow $f(t, s)$. This allows us to enforce flow conservation at s and t as well, which makes the corresponding linear

program simpler. The linear program is as follows.

$$\begin{aligned}
\text{maximize} \quad & f(t,s) \\
\text{subject to} \quad & f(u,v) \leq c_{uv}, && (u,v) \in E \\
& \sum_{v:(v,u)\in E} f(v,u) \leq \sum_{v:(u,v)\in E} f(u,v), && u \in V \\
& f(u,v) \geq 0 && (u,v) \in E.
\end{aligned}$$

It is not immediately obvious why the second set of inequalities implies flow conservation; all it seems to say is that for each u, the total flow into u is at most the total flow out of u. However, note that if this holds for all u, we in fact have equality of incoming and outgoing flow, since a deficit of flow at some u implies a flow surplus at some v. So this does in fact give us conservation of flow. Now we want to find the dual of this program. Our sense (hopefully) is that the dual will somehow relate to minimum cuts, given the foreshadowing of the previous section. Let's see! We introduce variables d_{uv} and p_u for each type of inequality in the primal.

$$\begin{aligned}
\text{minimize} \quad & \sum_{(u,v)\in E} c_{uv} d_{uv} && (21) \\
\text{subject to} \quad & d_{uv} - p_u + p_v \geq 0, \quad (u,v) \in E, && (22) \\
& p_s - p_t \geq 1, && (23) \\
& d_{uv} \geq 0, \quad (u,v) \in E. && (24)
\end{aligned}$$

It is known that extreme point solutions to these linear programs takes on values 0 or 1 at each coordinate. Let consider an optimal dual solution $(\mathbf{d}^*, \mathbf{p}^*)$. First, in order to satisfy $p_s^* - p_t^* \geq 1$ with 0,1 values, it must be the case that $p_s^* = 1$ and $p_t^* = 0$. This motivates an $s-t$ cut (S,T) with S consisting of nodes with value 1, and T the nodes with value zero. For an edge (u,v) such that $u \in S$ and $v \in T$, we have that $p_u^* = 1$ and $p_v^* = 0$, so by the first constraint $d_{uv} = 1$.

6 The Hungarian algorithm

In this section we use the motivation of the linear programs to develop an algorithm for simultaneously solving the maximum weight matching and minimum weight vertex cover problems. Our algorithm works by taking every exposed node on the left, and from each such node building a collection of alternating paths. We define this collection formally.

Definition (Alternating tree). Let $G = (U, V, E)$ be a bipartite graph, and M a matching on G. An *alternating tree* with respect to M is a tree which satisfies two conditions:

- the tree contains exactly one node $u \in U$. We call u the *root* of the tree.
- all paths between the root and any other node in the tree are alternating paths.

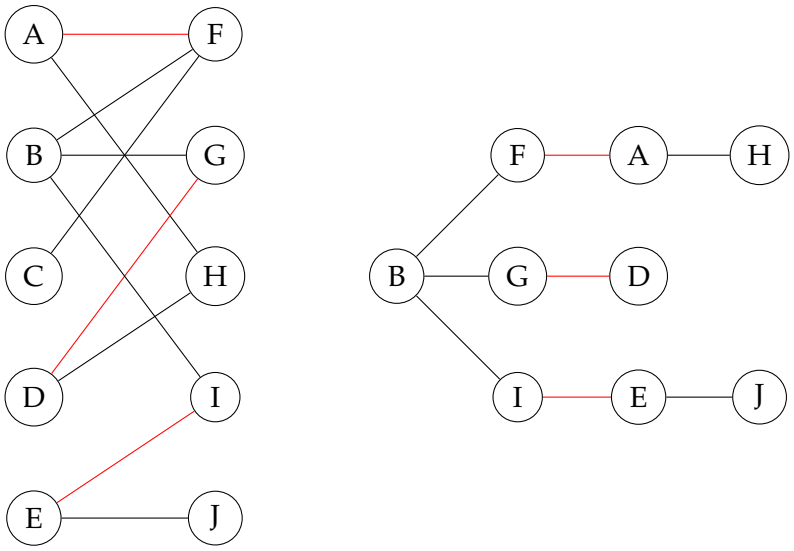


Figure 4: Bipartite graph with matching, corresponding alternating tree rooted at vertex B.

Let's remind ourselves what the primal-dual linear programs motivate. We want to minimize $\sum w_{uv}$, and maximize $\sum(y_u + y_v)$. Moreover, we want optimal solutions such that $\sum w_{uv} = \sum(y_u + y_v)$. For our primal, we are keeping track of edge weights. For the dual, we will be keeping track of a "labeling" on vertices, given by the y_u and y_v values. We define what a labeling is now.

Definition (Labeling). A *vertex labeling* on a weighted bipartite graph $G = (U, V, E)$ is a function $l : U \cup V \to \mathbb{N}$. We call the labeling *feasible* if for all $u \in U$ and $v \in V$, $l(u) + l(v) \geq w_{uv}$.

This labeling corresponds to our dual variables; i.e. a feasible labeling is a feasible dual solution. It will be helpful us to look at a certain subset of our graph where the labeling is exact ($l(u) + l(v) - w_{uv} = 0$).

Definition (Equality subgraph). The *equality subgraph* of $G = (U, V, E)$ is the graph $G_l = (U, V, E_l)$, where

$$E_l = \{(u, v) \ : \ l(u) + l(v) = w_{uv}\}.$$

In Figure 5 we show a bipartite graph along with its corresponding equality graph.

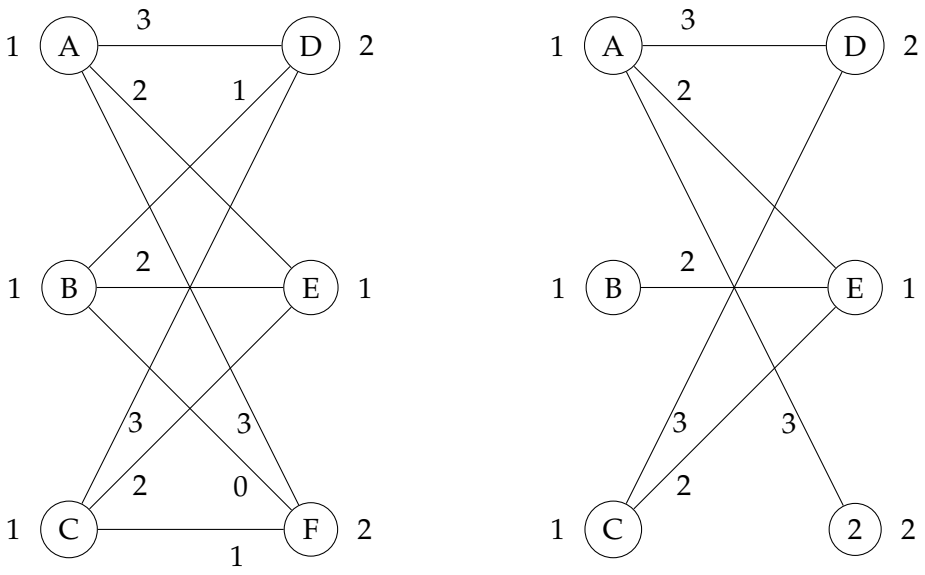


Figure 5: A weighted bipartite graph and its corresponding equality subgraph.

Theorem (Kuhn-Munkres). If l is a feasible labeling and M is a perfect matching in G_l then M is a max-weight matching.

Proof. Let $M^{'}$ be a perfect matching in G. Since every $u \in U \cup V$ is matched by exactly one edge in $M^{'}$, then

$$\sum_{(u,v)\in M^{'}} w_{uv} \leq \sum_{(u,v)\in M^{'}} (l(u) + l(v)) = \sum_{w\in U\cup V} l(w).$$

This says that the sum of our label values is an upper bound on the weight of any perfect matching.

Now suppose that M is a perfect matching in G_l. Then

$$\sum_{(u,v)\in M} w_{uv} = \sum_{w\in U\cup V} l(w).$$

So $\sum_{(u,v)\in M'} w_{uv} \leq \sum_{(u,v)\in M}$, meaning M must be maximum weight. □

Lemma 2. Let $S \subseteq U$ and $T = N_l(S) \neq V$. Set

$$\alpha_l = \min_{u\in S,\, v\notin T} \{l(u) + l(v) - w_{uv}\}$$

and

$$l^{'}(w) = \begin{cases} l(w) - \alpha_l & \text{if } w \in S \\ l(w) + \alpha_l & \text{if } w \in T \\ l(w) & \text{otherwise.} \end{cases}$$

Then $l^{'}$ is a feasible labeling, and

1. If $(u, v) \in E_l$ for $u \in S$ and $v \in T$ then $(u, v) \in E_{l'}$.
2. If $(u, v) \in E_l$ for $u \notin S$, and $v \notin T$, then $(u, v) \in E_{l'}$.
3. There is some edge $(u, v) \in E_{l'}$ for $u \in S$, $v \notin T$.

Proof. First, we show that $l^{'}$ is feasible. For $u \in U$, $v \in V$, there are four possibilities:

- if $u \in S$ and $v \in T$ then $l^{'}(u) + l^{'}(v) = l(u) + l(v)$.
- if $u \notin S$ and $v \notin T$, $l^{'}(u) = l(u)$ and $l^{'}(v) = l(v)$, so $l^{'}(u) + l^{'}(v) = l(u) + l(v)$.
- if $u \in S$ and $v \notin T$, $l^{'}(u) - \alpha$ and $l^{'}(v) = l(v)$. We know $\alpha = \min_{u\in S, v\notin T}\{l(u) + l(v) - w_{uv}\}$, which means $\alpha \leq l(u) + l(v) - w_{uv}$, and thus $l^{'}(u) + l^{'}(v) \geq w_{uv}$.
- if $u \notin S$ and $v \in T$, $l^{'}(u) = l(u)$ and $l^{'}(v) = l(v) + \alpha$, which is clearly feasible.

(1) and (2) follow from above. To see (3), note that there is some (u, v) with $u \in S$, $v \in T$ such that

$\alpha = l(u) + l(v) - w_{uv}$, so when we take $l'(u) = l(u) - \alpha$ and $l'(v) = l(v)$, we get

$$\begin{aligned} l'(u) + l'(v) - w_{uv} &= l(u) - \alpha + l(v) - w_{uv} \\ &= l(u) + l(v) - w_{uv} - \alpha \\ &= \alpha - \alpha \\ &= 0. \end{aligned}$$

This is exactly what it means for an edge (u, v) to be in E_l. □

We now look at the Hungarian method for finding maximum-weight matchings on bipartite graphs. This method was originally developed by Kuhn and Munkres, who named it in honor of the Hungarian mathematicians Kőnig and Egervary.

The Hungarian Method

1. Choose initial feasible labeling l and matching M in G_l.
2. If M is perfect in G_l, we are done. Otherwise, pick exposed vertex $u \in U$. Set $S = \{u\}$, $T = \emptyset$.
3. If $N_l(S) = T$, update labels as in lemma (this forces $N_l(S) \neq T$).
4. If $N_l(S) \neq T$, pick $v \in N_l(S) \setminus T$
   - If v is exposed, $p = u \to v$ is an augmenting path. Set $M := M \oplus p$. Go to 2.
   - If v is matched to some w, expand our alternating tree. $S := S \cup \{w\}$, $T := T \cup \{v\}$. Go to 3.

We now provide an example of this algorithm. Our initial matching is $M = \{(B, E), (C, D)\}$ (see Figure 6). Note that the current state of the graph is primal-dual feasible. Our algorithm chooses an exposed vertex in U, say A. So we have $S = \{A\}$ and $T = \emptyset$. We have that $N_l(S) \neq T$, so we find $E \in N_l(S) \setminus T$. E is matched, so we grow our alternating tree as follows: $S := S \cup \{B\} = \{A, B\}$, $T := T \cup \{E\} = \{E\}$. At this point $N_l(S) = T$, so we adjust our dual variables. Calculate

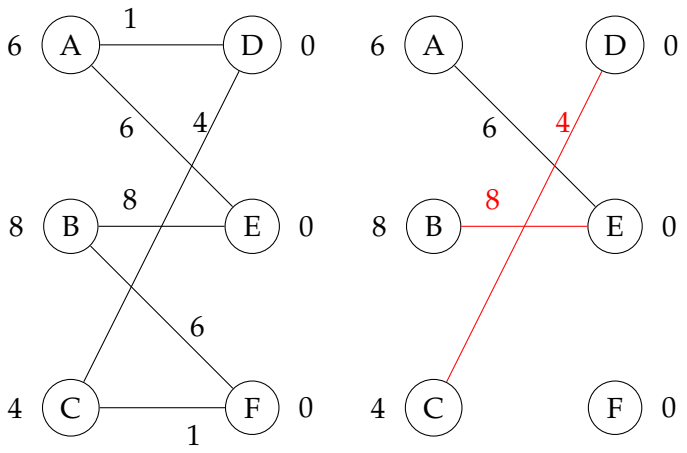


Figure 6: Bipartite graph (left) and corresponding equality graph (right) with initial matching

$\alpha = \min_{u \in S, v \notin T}\{l(u) + l(v) - w_{uv}\} = 2$ from edge (B, F). Our new equality graph is shown in Figure 7.

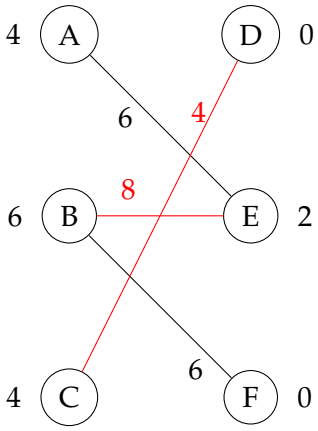


Figure 7: Second equality graph.

Now, $S = \{A, B\}$ is the same, but $N_l(S) = \{E, F\}$ has changed. $T = \{E\}$, so $N_l(S) \neq T$. So we choose $F \in N_l(S) \setminus T$. F is unmatched, meaning it is an endpoint of an augmenting path. In particular, $p = A, E, B, F$ is an augmenting path. Thus we improve our matching with $M := M \oplus p = \{(A, E), (B, F), (C, D)\}$. Our equality graph with the new matching is given in Figure 8.

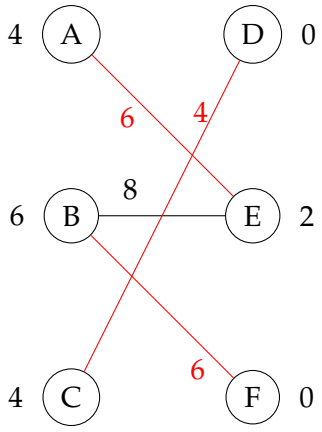


Figure 8: Equality graph after augmenting.

This is a perfect matching on the equality graph, so this matching must be a maximum weight matching on the graph. We can check that the values of the primal and dual solutions agree. The sum of weights in the matching is $6 + 6 + 4 = 16$, and the sum of the values of our dual variables is $4 + 6 + 4 + 2 = 16$.

Note that if we want to just find a maximum cardinality matching on a bipartite graph, we can just give all edges weight 1 and run this algorithm.

This algorithm was one of the first primal-dual algorithms developed, and it anticipated many later variations on the same theme. It displays the surprising connection between combinatorial optimization and linear programming, which we explore in the next section.