Thesis

A Thesis

Presented to

The Division of Mathematics and Natural Sciences

Reed College

In Partial Fulfillment

of the Requirements for the Degree

Bachelor of Arts

Zachary Campbell

May 2018

Approved for the Division

(Mathematics)

Jim Fix

Acknowledgements

Table of Contents

Introduction

Chapter 1

Bipartite graphs and matchings

1.1 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 alternative 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.

1.2 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 $(U \times V)$.

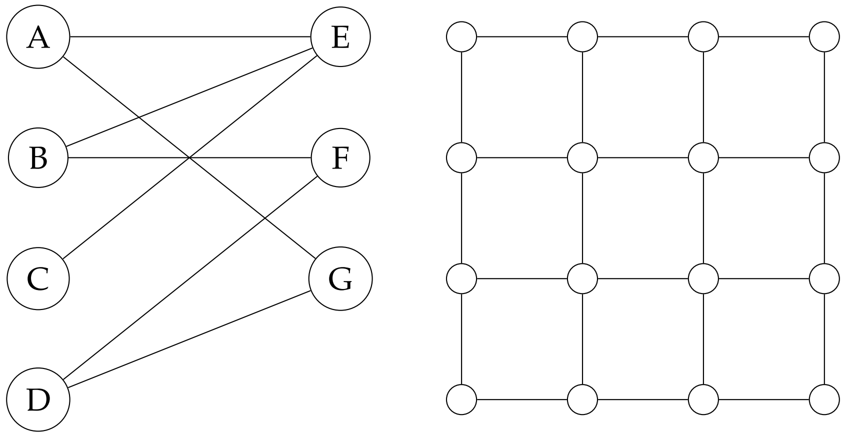


Figure 1.1: Examples of bipartite graphs

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 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, where the bold edges are the edges in the matching.

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

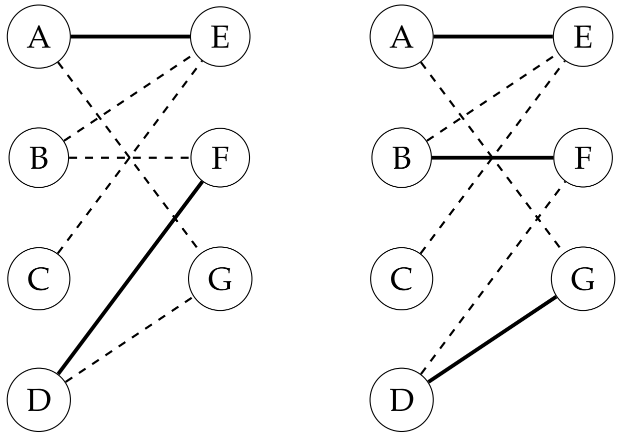


Figure 1.2: Examples of matchings on a bipartite graph

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, but it naturally leads to the following definition.

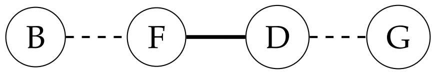
Definition (Maximal matching). A *maximal matching* on G is a matching M such that if any other edge not in M is added to M, it is no longer a valid matching. Alternatively put, M is maximal if there is no matching $M^{'}$ such that $M \subset M^{'}$.

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 these matchings have different sizes, even though both are maximal on G. This leads to the following definition.

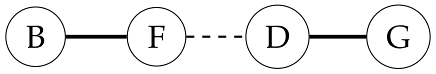
Definition (Maximum matching). A matching M on a graph G is said to be a *maximum* matching if for all other matchings $M^{'}$ on G, $|M^{'}| \leq |M|$.

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  $M := \emptyset$
2  while there exists an augmenting path $p$
3      $M := M \oplus p$
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.

1.3 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

$$\begin{aligned}
\text{maximize} \quad & \sum_{j=1}^{n} c_j x_j && (1.1)\\
\text{subject to} \quad & \sum_{j=1}^{n} a_{ij} x_j \leq b_j, \quad i = 1, \ldots, m && (1.2)\\
& x_j \geq 0, \quad j = 1, \ldots, n. && (1.3)
\end{aligned}$$

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

$$\begin{aligned}
\text{minimize} \quad & \sum_{j=1}^{n} c_j x_j && (1.4)\\
\text{subject to} \quad & \sum_{j=1}^{n} a_{ij} x_j \geq b_j, \quad i = 1, \ldots, m && (1.5)\\
& x_j \geq 0, \quad j = 1, \ldots, n. && (1.6)
\end{aligned}$$

Let's see an example of a linear program:

$$\begin{aligned}
\text{maximize} \quad & x_1 + x_2 \\
\text{subject to} \quad & 4x_1 - x_2 \leq 8 \\
& 2x_1 + x_2 \leq 10 \\
& 5x_1 - 2x_2 \geq -2 \\
& x_1, x_2 \geq 0.
\end{aligned}$$

For a simple linear program such as this, one may use basic substituion/elimination methods to get a solution; in this case it turns out that $x_1 = 2$ and $x_2 = 6$ is an optimal solution. Note that our constraint equations specificy a solution space in $\mathbb{R}^2$. In general, one can imagine n-dimensional solution spaces in which more complicated algorithms are required for finding solutions. For those interested, the two main algorithms for this are the Simplex method and the Ellipsoid method; detailed treatment on these algorithms can be found [CITE]. These algorithms are beyond the scope of this thesis, but it is important to note that they exist.

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 the next section will be to describe two problems in terms of their linear programs. One of these problems we've already looked at, which is the maximum matching problem. The other will be a problem called the minimum vertex cover problem. These two problems share a very fundamental relationship, which we will discover via their linear programs. We will also revisit a problem that algorithms students are familiar with, which is the max-flow/min-cut theorem. However, we will approach it from a different perspective, using linear programs, which will hopefully demonstrate the cool, deep math that is behind the relationship between flows and cuts.

Chapter 2

Vertex cover, duality theory, and the max-flow/min-cut relationship

This chapter aims to describe fundamental relationships between combinatorial optimization problems and linear programming.

2.1 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. Figure 1 shows some examples of vertex covers on the graph we looked at in the previous chapter.

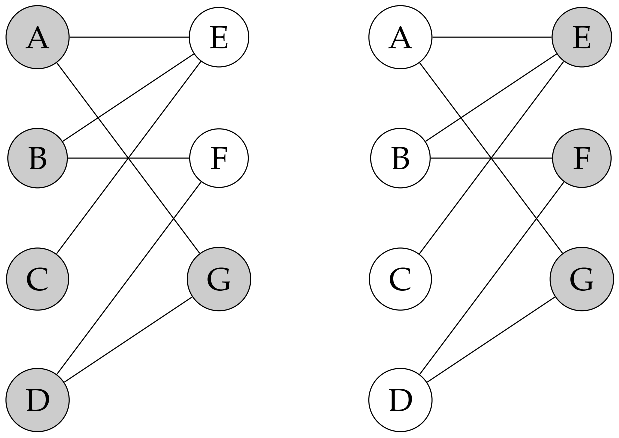


Figure 2.1: Examples of vertex covers

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 1. 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

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

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.

2.2 Duality theory

Definition (Dual). Let

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

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 duality theory to better understand the relationships between our combinatorial optimization problems.

2.2.1 Maximum-matching duality

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.

$$\begin{aligned}
\text{maximize} \quad & \sum_{u,v} x_{uv} && && (2.1)\\
\text{subject to} \quad & \sum_{v} x_{uv} \leq 1, && \forall u \in U, && (2.2)\\
& \sum_{u} x_{uv} \leq 1, && \forall v \in V, && (2.3)\\
& x_{uv} \in \{0, 1\}. && && (2.4)
\end{aligned}$$

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

$$\begin{aligned} \text{minimize} \quad & \sum_{u,v}(y_u + y_v) && (2.5) \\ \text{subject to} \quad & y_u + y_v \geq 1 \quad \forall u, v, && (2.6) \\ & y_u, y_v \geq 0. && (2.7) \end{aligned}$$

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

$$\text{maximize} \quad \sum_{u,v} w_{uv} x_{uv} \tag{2.8}$$

$$\text{subject to} \quad \sum_{v} x_{uv} \leq 1, \quad \forall u \in U, \tag{2.9}$$

$$\sum_{u} x_{uv} \leq 1, \quad \forall v \in V, \tag{2.10}$$

$$x_{uv} \geq 0. \tag{2.11}$$

Then the dual is

$$\text{minimize} \quad \sum_{u} y_u + \sum_{v} y_v \tag{2.12}$$

$$\text{subject to} \quad y_u + y_v \geq w_{uv} \quad \forall u, v, \tag{2.13}$$

$$y_u, y_v \geq 0. \tag{2.14}$$

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.

2.2.2 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 $u, v \in V$, $0 \le f(u, v) \le c_{uv}$.
- 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.

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

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.

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

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 $d_{uv}^* = 1$. This means that for any edge (u,v) in the cut, the corresponding $d_{uv}^* = 1$. Note that any other d_{uv}^* where (u,v) is not in the cut, it's value can be 0 without violating the constraints (we want them to be 0 since we are minimizing our objective function). Thus the objective function's value is equal to the capacity of this (S,T) cut, which must be a minimum cut. Strong duality tells us that this corresponds to the value of $f(t,s)$ in the primal linear program.

Thus, we have given an alternative formulation of the max-flow/min-cut relationship using linear programs. In this next chapter we present the main algorithm of this thesis, the Hungarian algorithm, which is interesting in itself, and also as a tool to motivate general primal-dual algorithms.

Chapter 3

The Hungarian algorithm

3.1 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.

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.

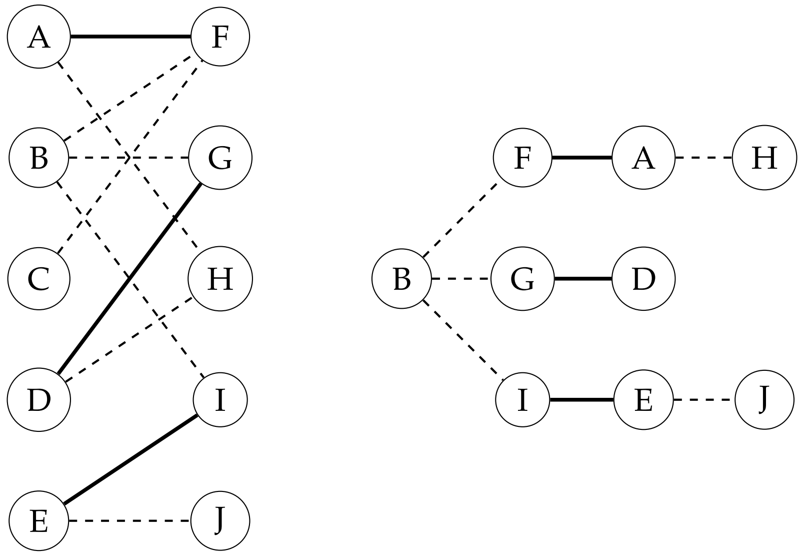


Figure 3.1: Bipartite graph with matching, corresponding alternating tree rooted at vertex B.

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 2 we show a bipartite graph along with its corresponding equality graph.

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

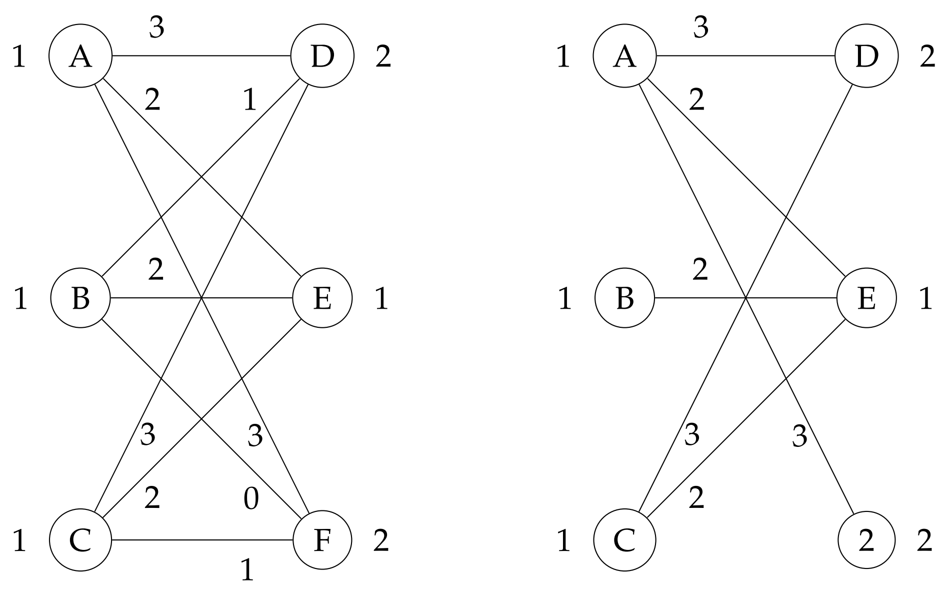


Figure 3.2: A weighted bipartite graph and its corresponding equality subgraph.

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. $\square$

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 $M = \{(B, E), (C, D)\}$

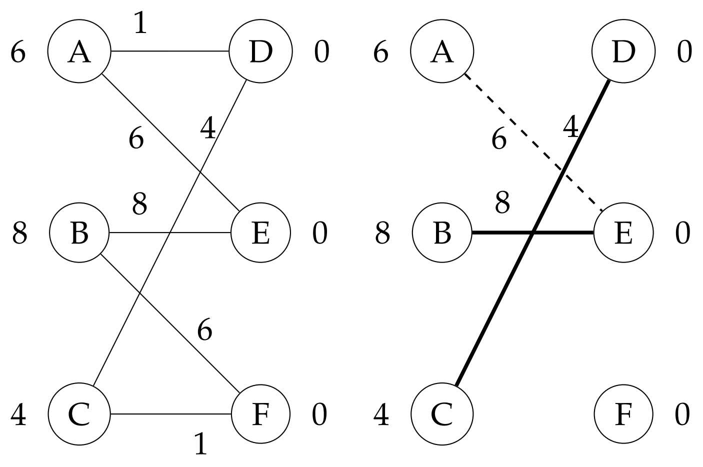


Figure 3.3: Bipartite graph (left) and corresponding equality graph (right) with initial matching

(see Figure 3). 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 $\alpha = \min_{u \in S, v \notin T}\{l(u) + l(v) - w_{uv}\} = 2$ from edge (B, F). Our new equality graph is shown in Figure 4.

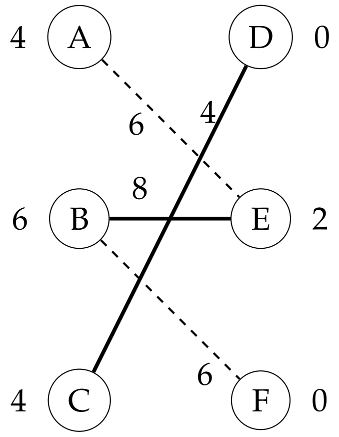


Figure 3.4: Second equality graph.

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 5.

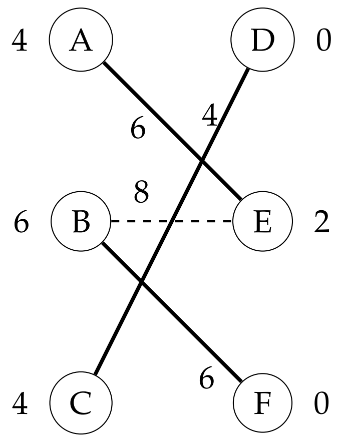


Figure 3.5: Equality graph after augmenting.

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.

Chapter 4

General primal-dual method and auction algorithms

In this chapter we will take a step back and look at primal-dual algorithms in more generality. The goal will be to describe a method of solving a set of primal-dual algorithms for *network design problems*. Again, we will be restricting our attention to bipartite graphs. In network design problems we are given a graph $G = (U, V, E)$ and a cost c_{uv} for each edge $(u, v) \in E$, and the goal is to find a minimum/maximum-cost subset $E^{'} \subset E$ that satisfies some criteria. Our maximum-cost matching problem is an example of this. There are other common examples that we will explore later on, but for now it suffices to just think of these problems as choosing subsets of our graph according to some stipulations. Throughout, we will be looking at undirected graphs.

4.1 The Classical Primal-Dual Method

We begin by looking at what's known as the "classical" primal-dual method, which is concerned with linear programs for polynomial-time solvable optimization problems. This will allow us to build up a framework for a more general primal-dual method that we

can use for approximation algorithms - i.e. for problems that are known to be NP-hard. **(Not sure if this is within the scope of the thesis – discuss with Jim)
Let's consider the linear program

$$\text{minimize } \mathbf{c}^T\mathbf{x} \tag{4.1}$$

$$\text{subject to } \; A\mathbf{x} \geq \mathbf{b} \tag{4.2}$$

$$\mathbf{x} \geq 0 \tag{4.3}$$

and its dual

$$\text{maximize } \mathbf{b}^T\mathbf{y} \tag{4.4}$$

$$\text{subject to } A^T\mathbf{y} \leq \mathbf{c} \tag{4.5}$$

$$\mathbf{y} \geq 0. \tag{4.6}$$

We first define a concept that we will use throughout the rest of this thesis.

Definition (Complementary slackness). Given two linear programs in the form above, the *primal complementary slackness conditions* are the conditions which, given primal solution $\mathbf{x}$, are necessary for a dual solution $\mathbf{y}$:

$$x_j > 0 \implies A^j\mathbf{y} = c_j,$$

where A^j is the jth column of A. Similarly, the *dual complementary slackness conditions* are the conditions which, given dual solution $\mathbf{y}$, are necessary for a primal solution $\mathbf{x}$:

$$y_i > 0 \implies A_i\mathbf{x} = b_i,$$

where A_i is the ith row of A. Together, these conditions give us necessary and sufficient conditions for solving the primal-dual system, which we will prove. The (maximization) primal slackness variables are given by $\mathbf{s} = \mathbf{b} - A\mathbf{x}$. The dual slackness variables are given by $\mathbf{t} = A^T\mathbf{y} - \mathbf{c}$.

Theorem . [CITE THIS THEOREM] Let $\mathbf{x}$ be a primal feasible solution, and $\mathbf{y}$ a dual feasible solution. Let $\mathbf{s}$ and $\mathbf{t}$ be the corresponding slackness variables. Then $\mathbf{x}$ and $\mathbf{y}$ are optimal solutions if and only if the following two conditions hold:

$$x_j t_j = 0 \quad \forall j \tag{4.7}$$

$$y_i s_i = 0 \quad \forall i. \tag{4.8}$$

Proof. Let $u_i = y_i s_i$ and $v_j = x_j t_j$, and $\mathbf{u} = \sum_i u_i$, $\mathbf{v} = \sum_j v_j$. Then $\mathbf{u} = 0$ and $\mathbf{v} = 0$ if and only if (7) and (8) hold. Also,

$$\begin{aligned}
\mathbf{u} + \mathbf{v} &= \sum y_i s_i + \sum x_j t_j \\
&= \sum y_i (b_i - A_i x_i) + \sum x_j (A_j^T y_j - c_j) \\
&= \sum b_i y_i - \sum c_j x_j,
\end{aligned}$$

so we get that $c^T\mathbf{x} = b^T\mathbf{y}$ if and only if $u + v = 0$, which proves the statement. □

The general "tug-of-war" between the primal and dual suggests an economic interpretation of slackness conditions. We can think of our primal (maximization) problem as concerned with profit given some constraints on resources, i.e. a resource allocation problem. The dual can be interpreted as a valuation of the resources – it tells us the availability of a resource, and its price. So if we have optimal $\mathbf{x}$ and $\mathbf{y}$, we can interpret slackness as follows: if there is slack in a constrained primal resource i ($s_u > 0$), then additional units of that resource must have no value ($y_u = 0$); if there is slack in the dual price constraint ($t_v > 0$) there must be a shortage of that resource ($x_v = 0$).

We now give an example of complementary slackness in action. Let's look back to our maximum weight matching problem.Recall the primal linear program for maximum-weight matching:

$$\text{maximize} \quad \sum_{u,v} c_{uv} x_{uv} \tag{4.9}$$

$$\text{subject to} \quad \sum_{v} x_{uv} \leq 1, \quad \forall u \in U, \tag{4.10}$$

$$\sum_{u} x_{uv} \leq 1, \quad \forall v \in V, \tag{4.11}$$

$$x_{uv} \geq 0. \tag{4.12}$$

and its dual

$$\text{minimize} \quad \sum_{u} y_u + \sum_{v} y_v \tag{4.13}$$

$$\text{subject to} \quad y_u + y_v \geq c_{uv} \quad \forall u \in U,\ v \in V, \tag{4.14}$$

$$y_u, y_v \geq 0. \tag{4.15}$$

The format here is a little different, since our primal is a maximization problem and the dual is a minimization, but it's easy enough to reverse the roles. It's easy to see our corresponding primal complementary slackness conditions are

$$x_{uv} > 0 \implies y_u + y_v = c_{uv}. \tag{4.16}$$

The dual complementary slackness conditions are

$$y_u > 0 \implies \sum_{v} x_{uv} = 1, \tag{4.17}$$

$$y_v > 0 \implies \sum_{u} x_{uv} = 1. \tag{4.18}$$

In general, the slackness conditions guide us in our algorithm – they tells us how, given a solution to one of the problems, we should augment the solution to the other. For example, the algorithm we presented for maximum-weight matching/minimum vertex cover intializes with a solution to both the primal and dual that satisfies conditions (8) and (10); the algorithm then at each step works to decrease the number of conditions in (9) that are unsatisfied, while maintaining satisfiability of (8) and (10). This method is not unique to the Hungarian algorithm. In fact, the Hungarian algorithm paved the way for this general method, which we describe presently.

Looking back at the original linear programs at the beginning of this chapter, suppose we have a dual feasible solution $\mathbf{y}$. We can then state the problem of finding a feasible primal solution $\mathbf{x}$ that obeys our complementary slackness conditions as another *restricted* linear program. Define the sets $A = \{j \mid A^j\mathbf{y} = c_j\}$ and $B = \{i \mid y_i = 0\}$. So A tells us which dual constraints (5) are tight, given the solution $\mathbf{y}$, and B tells us which y_i are 0. What we want to do is give a linear program to find a solution $\mathbf{x}$ that minimizes the "violation" of the complementary slackness conditions and the primal constraints, and to do so we will index our variables by these sets. We will have slack variables s_i which will describe the difference between $A_i\mathbf{x}$ and b_i for $i \notin A$. We do this because we want to look at all $y_i > 0$ where the we do not have that $A_i\mathbf{x} = b_i$. So part of our objective function will be to minimize the sum of these s_i. We also want to minimize the sum over variables x_j where $j \notin A$. This is because we want to see if there are any x_j such that $A^j\mathbf{y} \neq c_j$. So we give the following restricted primal linear program:

$$\text{minimize} \sum_{i \notin B} s_i + \sum_{j \notin A} x_j \tag{4.19}$$

$$\text{subject to} \quad A_i\mathbf{x} \geq b_i \quad i \in B, \tag{4.20}$$

$$A_i\mathbf{x} - b_i = s_i \quad i \notin B, \tag{4.21}$$

$$\mathbf{x} \geq 0, \tag{4.22}$$

$$\mathbf{s} \geq 0. \tag{4.23}$$

Observe that if this restricted primal has a feasible solution $(\mathbf{x}, \mathbf{s})$ such that the objective function is 0, then $\mathbf{x}$ is a feasible primal solution that satisfies the complementary slackness conditions for the dual solution $\mathbf{y}$. This means that $\mathbf{x}$ and $\mathbf{y}$ are optimal primal and dual solutions. If, however, the optimal solution to this restricted primal has value greater than 0, more work is required. We can consider the dual of the restricted primal:

$$\begin{aligned}
&\text{maximize } \mathbf{b}^T\mathbf{w} && && (4.24)\\
&\text{subject to } \; A^j\mathbf{w} \leq 0 && j \in A, && (4.25)\\
&\qquad A^j\mathbf{w} \leq 1 && j \notin A, && (4.26)\\
&\qquad w_i^{'} \geq -1 && i \notin B, && (4.27)\\
&\qquad w_i^{'} \geq 0 && i \in B. && (4.28)
\end{aligned}$$

What we want here is to improve our dual solution. By assumption, the optimal solution to this linear program's primal is greater than 0, so we know that this dual has a solution $\mathbf{w}$ such that $\mathbf{b}^T\mathbf{w} > 0$. What we want is the existence of some $\epsilon > 0$ such that $\mathbf{y}^{'} = \mathbf{y} + \epsilon\mathbf{w}$ is a feasible dual solution. In particular, a solution of this form will be an improvement on our original solution $\mathbf{y}$. We can calculate bounds on ϵ as follows. The two conditions we must satisfy in order to maintain dual feasibility are that $y^{'} \geq 0$ and $A^Ty^{'} \leq c$. This means that we need

$$y_i + \epsilon w_i \geq 0 \tag{4.29}$$

$$A_j^T y + A_j^T \epsilon w \leq c_j. \tag{4.30}$$

Let's consider the first one. When $w_i > 0$, we are fine; we need to be careful when $w_i < 0$ since this could potentially violate the inequality. Solving in this way, we get a first bound on ϵ:

$$\epsilon \leq \min_{i \in B: w_i < 0} (-y_i/w_i).$$

Now let's address the second inequality. When $A_j^T w \leq 0$, we are defintely okay. We need to be careful about violating the constraint when $A^T w > 0$. Thus, we can calculate a second bound on ϵ:

$$\epsilon \leq \min_{j \in A : A_j^T w > 0} \frac{c_j - A_j^T y}{A_j^T w}.$$

If we choose the lower of these two ϵ values, we obtain a new feasible dual solution that has greater objective value. We can then work by reiterating the procedure, with the hope that we find an optimal primal solution.

It's not immediately clear why reducing our original linear programs to a series of linear programs is heplful. However, note that the vector **c** has totally disappeared in the restricted primal and its dual. Recall that in the original linear program, **c** gave us the edge-costs on our graph. So this method reworks our original weighted problem into unweighted parts, which are easier to solve. Oftentimes, it is the case that we can interpret these unweighted problems as purely combinatorial problems, which means that instead of actually solving the problem with linear programming, we can solve it by combinatorial methods. Using a combinatorial algorithm to find a solution **x** that obeys the complementary slackness conditions, or to find an improved dual solution **y**, is oftentimes more efficient.

4.2 Primal-dual method for weighted matchings

Let us now look at an example of this method. We will look at a weighted matching problem, as in the previous chapter, but this time we will look at *minimizing* the cost of the matching, instead of maximizing. We do this mainly because it illustrates something important about the underlying structure of these matching problems. It will be easy to see how the same method can be used for the case in which we want a maximimum matching. So the primal linear program for a minimum weight perfect matching on a bipartite graph is given as follows.

$$\begin{aligned} \text{minimize} \quad & \sum_{u,v} c_{uv} x_{uv} && (4.31)\\ \text{subject to} \quad & \sum_{v} x_{uv} \geq 1, \quad \forall u \in U, && (4.32)\\ & \sum_{u} x_{uv} \geq 1, \quad \forall v \in V, && (4.33)\\ & x_{uv} \geq 0. && (4.34) \end{aligned}$$

Its dual is

$$\begin{aligned} \text{maximize} \quad & \sum_{u} y_u + \sum_{v} y_v && (4.35)\\ \text{subject to} \quad & y_u + y_v \leq c_{uv} \quad \forall u \in U,\ v \in V, && (4.36)\\ & y_u, y_v \geq 0. && (4.37) \end{aligned}$$

We need to start with a dual feasible solution, and try to find a primal solution that minimizes the violation of the constraints and slackness conditions. We can start with the trivial dual solution of $y_u, y_v = 0$ for all u, v. Let's now think about our primal complementary slackness. The set A is given by $\{(u,v) \in E \ : \ y_u + y_v = c_{uv}\}$. We know that these are the edges we want to include in our matching, and since we know our linear program has integer solutions at extreme points of the polyhedron, let's specify that $x_{uv} = 0$ for $(u,v) \notin a$. Now, our other slackness variables s_u, s_v look like

$$\sum_{v:(u,v)\in E} x_{uv} - s_u = 1$$

$$\sum_{u:(u,v)\in E} x_{uv} - s_v = 1.$$

So we want to minimize over the sum of s_u and s_v. Note that at this point, our set B consists of all vertices. So our restricted primal linear program is

$$\begin{aligned}
\text{minimize} \sum_{u \in U} s_u + \sum_{v \in v} s_v & & (4.38)\\
\text{subject to} \quad \sum_v x_{uv} - s_u = 1 & \quad \forall u, & (4.39)\\
\sum_u x_{uv} - s_v = 1 & \quad \forall v, & (4.40)\\
x_{uv} = 0 & \quad (u,v) \notin A, & (4.41)\\
x_{uv} \geq 0 & \quad (u,v) \in A, & (4.42)\\
\mathbf{s} \geq 0. & & (4.43)
\end{aligned}$$

Let's first observe that all components of this restricted primal take on values 0 or 1, as in the original primal. Moreover, note that we have turned a weighted problem into an unweighted combinatorial problem. We've specified that we are not including any edge $(u,v) \notin A$ in our matching, and we are trying to include as many $(u,v) \in A$ as possible in our matching by minimizing the slackness variables. Note that the graph $G^{'} = (U,V,J)$ is exactly the equality subgraph as defined in the previous chapter! In our Hungarian algorithm we repeatedly sought to find maximum cardinality matchings within this subgraph, which is exactly what this restricted primal is having us do. This tells us that the problems of maximum weight matching and minimum weight matching only differ in the labeling we are specifying. The underlying procedure for solving both of the problems is essentially the same. So if we find a perfect matching in $G^{'}$, we will have found an $\mathbf{x}$ that obeys the complementary slackness conditions, i.e. $\sum_u s_u + \sum_v s_v = 0$. Moreover, this implies that the dual solution $\sum_u y_u + \sum_v y_v$ must be optimal as well.

Now, if the solution $\sum_u s_u + \sum_v s_v > 0$, we do not have an optimal $\mathbf{x}$, so we need to adjust our dual. We look at this now, in the dual linear program of the restricted primal.

$$\text{maximize} \sum_{u\in U} w_u + \sum_{v \in V} w_v \tag{4.44}$$

$$\text{subject to} \quad w_u + w_v \leq 0 \quad \forall (u,v) \in A, \tag{4.45}$$

$$w_u + w_v \leq 1 \quad \forall (u,v) \notin A, \tag{4.46}$$

$$w_u, w_v \geq -1 \quad u,v \notin B, \tag{4.47}$$

$$w_u, w_v \geq 0 \quad u,v \in B, \tag{4.48}$$

$$\mathbf{w} \geq 0. \tag{4.49}$$

We now want to find an ϵ such that the solution $z = \sum_u y_u + \sum_v y_v + \epsilon(\sum_u w_u + \sum_v w_v)$ is (1) feasible and (2) an improvement of the dual objective. First of all, we know that since the restricted primal has solution ≥ 0, the solution to this dual will also be ≥ 0. So we just need to worry about the condition

$$y_u + y_v + \epsilon(w_u + w_v) \leq c_{uv}.$$

So we get that we at least need that $\epsilon \leq \min_{(u,v)\notin A : w_u + w_v > 0} \frac{c_{uv} - y_u - y_v}{w_u + w_v}$. We can refine this by noting that since $0 < w_u + w_v \leq 1$ for $(u,v) \notin A$, we have $\epsilon = \min_{(u,v)\notin A}(c_{uv} - y_u - y_v)$. Note that the negative of this is exactly the quantity we modify our labeling by in the Hungarian algorithm in the previous chapter. Thus we've found an ϵ that maintains dual feasibility, and increases the objectie function. We can use this solution and revisit the restricted primal in order to look for an improved feasible primal solution.

4.3 Auction algorithms

We now look at a cool application of weighted bipartite matching. Let's imagine our assignment problem is an auction. We will motivate this through an economic lens. Suppose that a good g has a price p_g, and a buyer b must pay p_g to receive this item. Suppose that the amount b is willing to pay for g, or b's valuation of b, is c_{bg}. Then the net value of this item to b is $c_{bg} - p_g$. Assuming that each bidder b is a rational agent acting in their own best interest, each b would want to be assigned a good g that maximizes their net value, i.e. for all b with good g, we have that

$$c_{bg} - p_g = \max_g \{c_{bg} - p_g\}.$$

This would give us an economic system in equilibrium; no bidder would have an incentive to seek/trade for a different good. The reason this is of interest to us is that an equilibrium assignment maximizes total profit (i.e. it solves the primal linear program for maximum weight matchings), while the corresponding set of prices solves the associated dual problem.

Our goal is to maximize the total amount earned in the auction – i.e. to maximize $\sum_{(b,g)} c_{bg}$, under the constraint that no bidder gets more than one good, and no good is purchased by more than one bidder.

At a general step in our algorithm, we will want to consider a bidder b that is currently not the owner of any good, and find a good g that maximizes $c_{bg} - p_g$. We will then need to do to things: (1), replace the current owner of g (put them back into the set of unassigned bidders) with b; (2) bump the price of g by some amount δ. Step (2) is tricky to figure out. Bertsekas [CITE] describes in detail the process of finding a good δ and, more importantly, what values of δ to avoid. Here we leave out the details of finding a decent δ. We first present the algorithm with a given δ, and then analyze why it is a good choice.

```
ALG 1
1  For each $g \in G$, set $p_g := 0$ and $owner_g := null$.
2  Queue $Q := B$.
3  Set $\delta = 1/(|G|+1)$.
4  while $Q \neq \emptyset$
5      $b = Q.deque()$
6      Find $g \in G$ that maximizes $c_{bg} - p_g$
7      if $c_{bg} - p_g \geq 0$
8          $Q.enqueue(owner_g)$
9          $owner_g = b$
10         $p_g = p_g + \delta$
11 return $(owner_g, g)$ for all $g$.
```

Definition . We say that a bidder b is δ-happy if one of the following is true:

1. b is the owner of some good g, and for all other goods $g^{'}$, we have that $\delta + c_{bg} - p_g \geq c_{bg'} - p_{g'}$, i.e. our b has the good g that maximizes the value of their contribution.

2. for no good g does it hold that $owner_g = b$ and for all goods g $w_{bg} \leq p_g$.

What we will show is that the algorithm reaches an equilibrium in which all bidders are "happy," as defined above. The loop invariant is that all bidders not in Q are δ- happy. This is clearly true at the beginning, as all bidders start in our queue. When a bidder b is dequeued, line 6 in our loop chooses good g that maximizes $w_{bg} - p_g$, which means it chooses a good that makes b δ-happy, if such a good exists. We need to confirm that this step does not hurt the invariant for any other bidder $b^{'}$. Well, an increase in p_g by δ for any g not owned by $b^{'}$ does not violate the inequality $\delta + w_{b'g'} - p_{g'} \geq w_{b'g} - p_g$. On the other hand, if $b^{'}$ did own g, this means that $b^{'}$ has been thrown back into Q, so $b^{'}$ no longer owns anything.

Lemma . If all bidders are δ-happy then for every matching $M^{'}$ we have that

$$n\delta + \sum_{b=owner_g} w_{bg} \geq \sum_{(b,g)\in M'} w_{bg},$$

where n is the number of bidders.

First, note that this lemma implies the correctness of our algorithm. Since $n\delta < 1$ and all of our weights are integral, proving this inequality will show that the matching we output is a maximum matching.

Proof. Fix a bidder b, and let good g be such that $b = owner_g$. Let $g^{'}$ be the good assigned to b in $M^{'}$. (Note: these could be *null*, in which case we adopt the convention of assigning their weights and prices to be zero.) Since b is δ- happy, we have that $\delta + w_{bg} - p_g \geq w_{bg} - p_g$. Now let's sum over all b to get

$$\sum_{b=owner_g} (\delta + w_{bg} - p_g) \geq \sum_{(b,g^{'})\in M^{'}} (w_{bg^{'}} - p_g).$$

We know that our algorithm gives us a matching, as does $M^{'}$, which means that each good g can only appear at most once on each side of this inequality. So if we subtract $\sum_g p_g$ from each side, and rearrange a bit, we get

$$\sum_{b=owner_g} \delta + \sum_{b=owner_g} w_{bg} \geq \sum_{(b,g^{'})\in M^{'}} (w_{bg^{'}}).$$

But $\sum_{b=owner_g} \delta \leq n\delta$, which gives us what we want. $\square$

Let's think about how this algorithm relates to the corresponding primal and dual linear programs. First, note that this algorithm maintains primal feasibility at all times, as each good has at most one owner, and each bidder owns at most one good. However, at no point are we maintaining dual feasibility. We can define the "price" on bidders b as $p_b = 0$ for all b. Furthermore, the prices on goods p_g never exceed $\max_b\{w_{bg}\}$. We can think of these p_b and p_g as the corresponding dual variables in the dual linear program. Throughout the course of this algorithm, we never have that $p_b + p_g \geq w_{bg}$, which means that we are violating our primal complementary slackness conditions. Note however that we do maintain the dual complementary slackness conditions. It is still useful to think about this algorithm in terms of the linear programs since, although infeasible as a dual solution, our final price vector $\mathbf{p}$ is the pointwise minimum such that all bidders are δ-happy. So the algorithm is still performing a minimization over $\sum_b p_b + \sum_g p_g = \sum_g p_g$, but relaxing the constraint that $p_b + p_g = p_g \geq w_{bg}$.