The Hungarian Algorithm and the Primal-Dual Method

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 Sean Campbell

May 2018

Approved for the Division

(Mathematics)

James D. Fix

Acknowledgements

Accomplishments are not made in isolation. I have received a significant amount of help along the way from many people, too many to list and give due credit. I use this page to thank all of you now. I most definitely would not have gotten this far without those of you who have supported me. Thank you.

Table of Contents

Abstract

This thesis explores the relationship between problems in combinatorial optimization and those in linear programming. The former field of study is often introduced to students in an introductory course on discrete mathematics or algorithms, while the latter is often reserved for upper-division algorithms courses. Regardless, we focus on combinatorial problems typically introduced in an algorithms course and how they relate to the field of linear programming. In particular, we look at various matching problems on bipartite graphs and their associated linear programs. In doing so, we explore the fruitful intersection of these two fields by looking at the development of algorithms that solve combinatorial problems using theoretical results from linear programming.

"Deep in the human unconscious is a pervasive need for a logical universe that makes sense. But the real universe is always one step beyond logic."

- Frank Herbert, *Dune*

Introduction

This thesis explores the curious mathematics behind a theorem covered in a typical course in algorithms and data structures, namely the max-flow min-cut theorem. The aim of the thesis is to demonstrate the intimate relationship between such problems in combinatorial optimization and linear programming.

In order to do this, we first review concepts in elementary graph theory (bipartite graphs and matchings) and introduce the topic of linear programming. We then look at a problem called the vertex cover problem, which we will discover is fundamentally related to the matching problem on bipartite graphs. It turns out that given a solution to one of these problems, we can construct a solution to the other. We discover this relationship using duality theory, an important concept in linear programming.

As a consequence of the duality theorems, we show that the max-flow min-cut relationship in flow networks is simply a special case of what we call linear programming duality. After this, we turn our attention to the main algorithm in this thesis: the Hungarian algorithm. Its invention was a significant development in combinatorial optimization in that its generalization led to several other techniques based on linear programming and linear programming duality.

Finally, we take a step back in our final chapter and look at the general "primal-dual method" inspired by the Hungarian algorithm. This method uses general linear programming principles to design algorithms for network design problems (essentially problems on weighted graphs in which one desires to choose some optimal subset of the graph).

The main result here is that, quite magically, we are able to turn these *weighted* combinatorial problems into equivalent unweighted problems, which are often much easier to solve. We finish by describing a class of auction algorithms which use this primal-dual methodology.

This thesis began as an endeavour to understand the Hungarian algorithm. From this, we learned quite a lot about its influence on the world of combinatorial optimization and linear programming. Although we restrict our attention to polynomial-time solvable problems in this thesis, the tools developed generally extend to, and are geared towards, approximation algorithms for NP-hard problems.

Chapter 1

Matchings and Linear Programming

In this chapter, we review bipartite graphs and matchings and then discuss their relevance to this thesis. We will also begin our discussion of linear programming, in a casual way, to give the reader a taste for what is to come in subsequent chapters.

1.1 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 1.1. 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 a subset of $(U \times V)$.

In Figure 1.1 we have a bipartite graph with vertex partition given by $U = \{\mathrm{A}, \mathrm{B}, \mathrm{C}, \mathrm{D}\}$ and $V = \{\mathrm{E}, \mathrm{F}, \mathrm{G}\}$. All edges in this graph are between a vertex $u \in U$ and a vertex $v \in V$. The graph on the right is also bipartite, though less obviously so. It may take a little more time to convince yourself that you can partition the vertices into disjoint sets 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

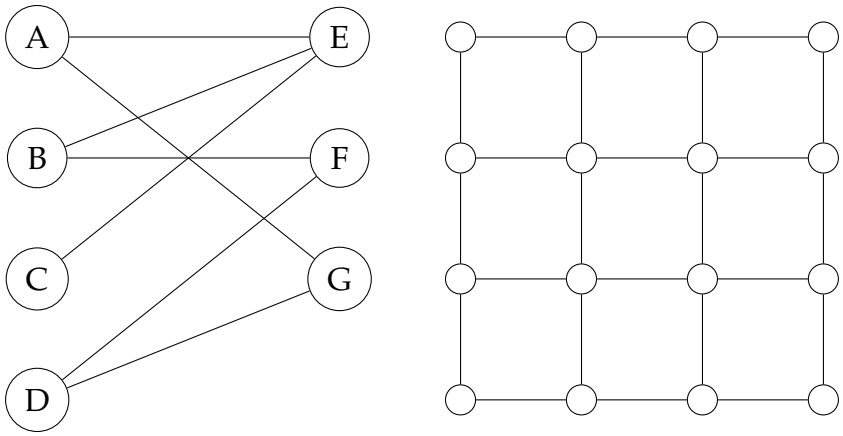

Figure 1.1: Examples of bipartite graphs.

like to solve on these graphs.

Definition 1.2. 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 graphs in Figure 1.2 show some of the many possible examples of matchings on one of the graphs from Figure 1.1, where the bold edges denote edges that are in the matching. Oftentimes, we want to find the largest matching on a graph. This is

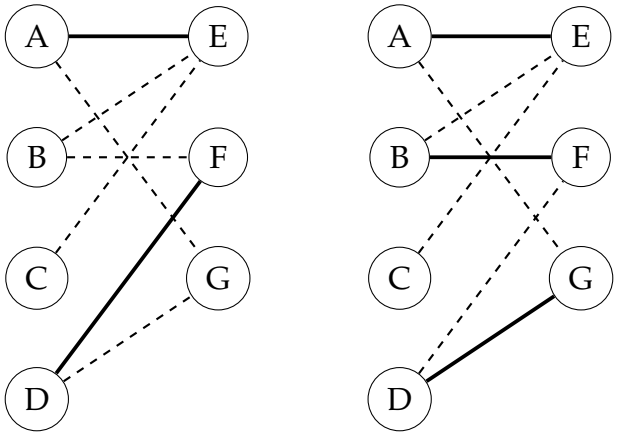

Figure 1.2: Examples of matchings on a bipartite graph, where bold edges denote those in the matching.

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, but it naturally leads to the following definition.

Definition 1.3. 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 1.2 are maximal matchings; in each case, there are no edges that we can add to M and have that M is still a valid matching. However, notice that these matchings have different sizes, even though both are maximal on G. This leads to the following definition.

Definition 1.4. 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 $M = \{(\mathrm{A},\mathrm{E}),(\mathrm{B},\mathrm{F}),(\mathrm{D},\mathrm{G})\}$ is a maximum matching, since there are no valid matchings such that $|M| > 3$ on this 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, we will first look at a motivating example. Suppose that we have the matching M on the left in Figure 1.2. So $M = \{(\mathrm{A},\mathrm{E}),(\mathrm{D},\mathrm{F})\}$. Consider the path p of edges and vertices in our graph, shown in Figure 1.3. Let's perform an operation that we will denote $M \oplus p$, which operates like

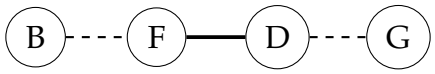

Figure 1.3: The path p

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 a new path $p^{'}$, where (B,F) and (D,G) are now in M, but (D,F) is not.

This new path is shown in Figure 1.4. First, we must check that the new matching $M \oplus p$

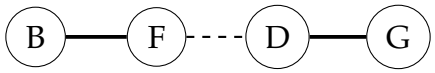

Figure 1.4: Augmented path p'

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 1.2.

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 by one. We define this formally now.

Definition 1.5. Let G be a graph and M be some matching on G. An *alternating path* is a sequence of edges that alternate between being matched and unmatched.

Definition 1.6. 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 provides a general method for finding a maximum matching on a bipartite graph: look for augmenting paths, and augment the current matching by that augmenting path. The following theorem tells us that this procedure will terminate with a maximum matching.

Theorem 1.7. *(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. Students with a background in algorithms will be familiar with this framework.

```
MAXIMUM MATCHING ALGORITHM(U, V, E)
1  M := ∅
2  while there exists an augmenting path p with respect to M:
3        M := M ⊕ p
4  return M
```

Figure 1.5: Algorithm for maximum cardinality matching.

For a detailed treatment of the subleties of this algorithm (known as the Ford-Fulkerson-Method) see Chapter 26 of Cormen, Leiserson, Rivest, and Stein [5]. We will rely on this method of finding maximum cardinality matchings later on in this thesis.

1.2 Linear Programming

The development of combinatorial optimization has been deeply intertwined with the discipline of linear programming. A detailed treatment of linear programming can be found in Bertsimas and Tsitsiklis [2] and Vazirani [12]. For the purposes of this thesis, we will treat linear programming more casually, only focusing 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. These are the Ellipsoid algorithm and the Simplex algorithm. An interested reader may find a detailed treatment of these in Papadimitriou and Steiglitz [11].

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

$$\text{maximize} \quad \sum_{j=1}^{n} c_j x_j \tag{1.1}$$

$$\text{subject to} \quad \sum_{j=1}^{n} a_{ij} x_j \leq b_i, \quad i = 1, \ldots, m \tag{1.2}$$

$$x_j \geq 0, \quad j = 1, \ldots, n. \tag{1.3}$$

We call the function in (1.1) our *objective function*, and the linear inequalities (1.2) and (1.3) our constraints. Similarly, a *minimization* linear program takes the form

$$\text{minimize} \quad \sum_{j=1}^{n} c_j x_j \tag{1.4}$$

$$\text{subject to} \quad \sum_{j=1}^{n} a_{ij} x_j \geq b_i, \quad i = 1, \ldots, m \tag{1.5}$$

$$x_j \geq 0, \quad j = 1, \ldots, n. \tag{1.6}$$

A possible solution to a linear program is *feasible* if it satisfies all of the linear constraint equations. The following is a simple example from Cormen, Leiserson, Rivest, and Stein [5] 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 and 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 methods are required for finding solutions.

Many problems which on the face may not appear to be linear 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. The first of these problems will be the is the maximum matching problem. The second will be a problem called the minimum vertex cover problem. Both 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 and Duality Theory

This chapter aims to describe fundamental relationships between combinatorial optimization problems and linear programming. More specifically, we will define a new problem – the vertex cover problem – and show how this problem is intimately related to the matching problem through duality theory. We also use duality theory to shed light on the max-flow min-cut theorem.

2.1 The Vertex Cover Problem

We begin this section by defining the vertex cover problem. Again, this is a problem that can be posed for graphs in general (not necessarily bipartite), but we will be restricting our attention to bipartite graphs. This is a good idea for many reasons, the most pertinent being that this problem is NP- complete in the general case.

Definition 2.1. 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 either $u \in C$ or $v \in C$. A vertex cover C on G is a *minimum vertex cover* if for any other vertex cover $C^{'}$ on G, we have that $|C| \leq |C^{'}|$.

Using what we have already learned, we now can specify at least one relation between matchings and vertex covers: namely, the set of all vertices incident to at least one edge in

any maximum matching forms a vertex cover. Figure 2.1 shows some examples of vertex covers on the graph we looked at in the previous chapter.

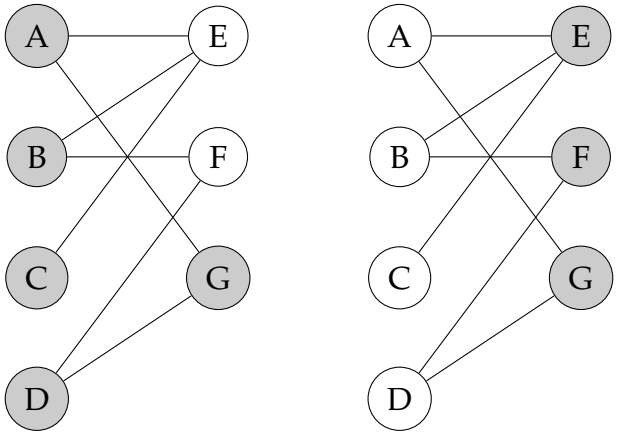

Figure 2.1: Examples of vertex covers.

You can convince yourself that the cover on the right is a minimum cover, i.e. there are no vertex covers on this graph of size less than three. This brings will bring us to an important theorem, but we must first prove a lemma.

Lemma 2.2. Let $G = (U, V, E)$ be a bipartite graph. Let M be a matching on G and C a vertex cover on G such that $|M| = |C|$. Then M is a maximum matching and C is a minimum vertex cover.

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

Before we prove the following theorem, we want to be able to talk about collections of alternating paths, as it will be useful to us in discussing the methods of finding maximum matchings on graphs.

Definition 2.3. 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.

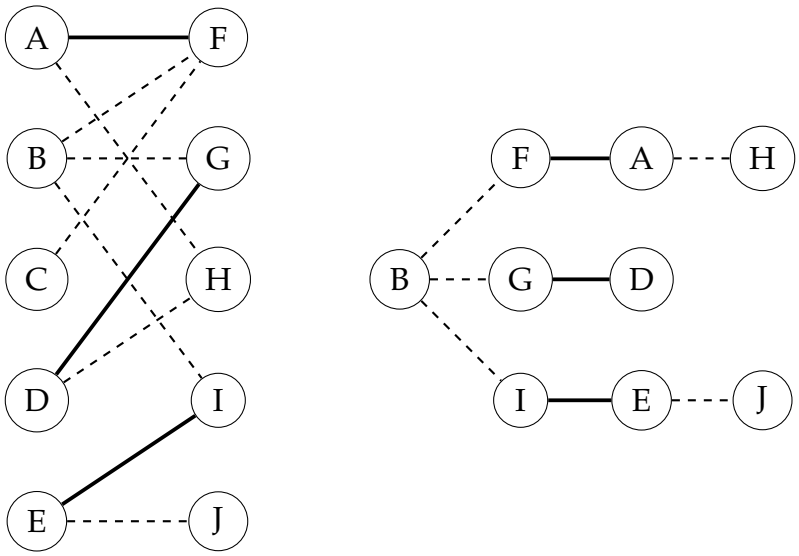

Figure 2.2: Bipartite graph with matching (left) and corresponding alternating tree rooted at vertex B (right).

Theorem 2.4. *(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 $|M| = |C|$.*

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

$$F := \{u \in U \mid u \text{ unmatched}\} \tag{2.1}$$

$$R := \{\text{all vertices connected to vertices in } F \text{ by alternating paths}\}. \tag{2.2}$$

Let $S = R \cap U$ and $T = R \cap V$. Denote $N(S)$ to be the set of all vertices connected to

elements of S (i.e. the "neighbors" of S). Then we have the following:

$$(1) \text{ Every vertex in } T \text{ is matched;}$$
$$(2)\ N(S) = T.$$

The first claim comes from the fact that, if M is a maximum matching, then our alternating paths that start at a vertex in F must have an even length ≥ 2. Otherwise, we would have an augmenting path, which contradicts our assumption that M is a maximum matching. The second claim comes from the fact that every node in $N(S)$ is connected to vertices in F by an alternating path.

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

This theorem and lemma tell us that there is a deep relationship between maximum matchings and minimum vertex covers on bipartite graphs. Subsequently, given a solution to a maximum matching problem, we can turn it into a solution to a minimum vertex cover problem, and vice versa. Our goal now is to generalize this concept.

2.2 Duality Theory

We now turn to duality theory, which allows us to speak more generally about the relationships between certain combinatorial problems that we are interested in solving.

Definition 2.5. Let the *primal linear program* be the following linear program.

$$\text{maximize} \quad \sum_{i=1}^{n} c_i x_i \tag{2.3}$$

$$\text{subject to} \quad \sum_{i=1}^{n} a_{ij} x_i \leq b_j, \quad j = 1, \ldots, m \tag{2.4}$$

$$x_i \geq 0, \quad i = 1, \ldots, n. \tag{2.5}$$

Then we define the *dual linear program* of this primal linear program to be the following linear program.

$$\text{minimize} \quad \sum_{j=1}^{m} b_j y_j \tag{2.6}$$

$$\text{subject to} \quad \sum_{j=1}^{n} a_{ij} y_j \geq c_i, \quad i = 1, \ldots, m \tag{2.7}$$

$$y_j \geq 0, \quad j = 1, \ldots, n. \tag{2.8}$$

The dual of the dual is always the primal.

In some settings we instead have a minimization problem (the dual here) acting as the primal problem, while the maximization problem is actually the dual. Ultimately, it does not matter which one we call the dual and which one we call the primal; what matters is how they relate to each other. Let us introduce some notation. First, let us denote the primal maximization problem by Γ, and the dual minimization problem by Ω. For a given linear program, we denote an optimal solution by **OPT**, by which we mean the value that optimizes the objective function and satisfies the linear constraints. Naturally, we want to know how solutions to linear programs that are primal-dual to each other relate.

Theorem 2.6. *(Weak duality) If the primal linear program (in maximization form) and the dual*

(in minimization form) are both feasible, then

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

Proof. Suppose that $\mathbf{x} = (x_1, x_2, \ldots, x_n)$ is a feasible solution to Γ, and $\mathbf{y} = (y_1, y_2, \ldots y_m)$ is a feasible solution to Ω. So $\mathbf{OPT}(\Gamma) = \sum_{i=1}^{n} c_i x_i$ and $\mathbf{OPT}(\Omega) = \sum_{j=1}^{n} b_j y_j$. Because $\mathbf{y}$ is dual feasible and all x_i and y_j are nonnegative, we have

$$\sum_{j=1}^{m} b_j y_j \geq \sum_{j=1}^{m} \left(\sum_{i=1}^{n} a_{ij} x_i \right) y_j.$$

Similarly, we have that

$$\sum_{i=1}^{n} c_i x_i \leq \sum_{i=1}^{n} \left(\sum_{j=1}^{m} a_{ij} y_j \right) x_i.$$

Note that by rearranging terms, we have the following equality:

$$\sum_{i=1}^{n} \left(\sum_{j=1}^{m} a_{ij} y_j \right) x_i = \sum_{j=1}^{m} \left(\sum_{i=1}^{n} a_{ij} x_i \right) y_j,$$

which shows that

$$\sum_{i=1}^{n} c_i x_i \leq \sum_{i=1}^{n} \left(\sum_{j=1}^{m} a_{ij} y_j \right) x_i = \sum_{j=1}^{m} \left(\sum_{i=1}^{n} a_{ij} x_i \right) y_j \leq \sum_{j=1}^{m} b_j y_j,$$

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

What's surprising is the following theorem.

Theorem 2.7. *(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,*

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

In Chapter 4 we will use duality theory to get necessary and sufficient conditions for $\mathbf{OPT}(\Gamma) = \mathbf{OPT}(\Omega)$, which will give us **Theorem 2.6**. Our goal now is to use facts about duality to better understand the relationships between our combinatorial optimization problems.

2.2.1 Maximum-Matching Duality

Let us formulate the maximum matching as a linear program. Our goal is to maximize the number of edges in our matching. Let us define a set of variables x_{uv}, one for each $(u, v) \in E$. These will be the unknowns in the linear program. We will use these to define a subset of edges as follows: when $x_{uv} = 1$, we will say that (u, v) is included in that subset; when $x_{uv} = 0$, we will say that (u, v) is excluded from that subset. Since we are trying to construct a matching on our graph, we need the constraints that ensure no edge is incident to more than one vertex in the matching; otherwise, the matching would be invalid. For a fixed vertex $u \in U$, the number of edges in the matching incident to u is given by $\sum_{v \in V} x_{uv}$. So we want this to be less than or equal to one. Similarly, for any vertex $v \in V$, we want $\sum_{u \in U} x_{uv}$ to be less than or equal to one. This gives us the following linear program:

$$\text{maximize} \quad \sum_{u,v} x_{uv} \tag{2.9}$$

$$\text{subject to} \quad \sum_{v} x_{uv} \leq 1, \qquad \text{for all } u \in U \tag{2.10}$$

$$\sum_{u} x_{uv} \leq 1, \qquad \text{for all } v \in V \tag{2.11}$$

$$x_{uv} \in \{0, 1\}. \tag{2.12}$$

What we have presented 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 known that this linear program attains an integer solution at extreme

points (see Birkhoff [3] and von Neumann [13]) of the polyhedron solution space, so we can drop the integrality requirements and just say $x_{uv} \geq 0$.

Now let us try and construct the dual of this linear program. To do this 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 and y_v. Since our coefficient on x_{uv} in the primal is the constant vector one, our only constraint will be that $y_u + y_v \geq 1$. This gives us the dual linear program:

$$\text{minimize} \quad \sum_{u \in U} y_u + \sum_{v \in V} y_v \tag{2.13}$$

$$\text{subject to} \quad y_u + y_v \geq 1, \quad \text{for all } (u, v) \in E \tag{2.14}$$

$$y_u, y_v \geq 0. \tag{2.15}$$

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 variant of the maximum matching problem called the *maximum weight matching problem*. In this version, we are given a bipartite graph with non-negative edge weights w_{uv} for all edges $(u, v) \in E$. 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 total "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 the following linear program:

$$\text{maximize} \quad \sum_{u,v} w_{uv} x_{uv} \tag{2.16}$$

$$\text{subject to} \quad \sum_{v} x_{uv} \leq 1, \quad \text{for all } u \in U \tag{2.17}$$

$$\sum_{u} x_{uv} \leq 1, \quad \text{for all } v \in V \tag{2.18}$$

$$x_{uv} \geq 0. \tag{2.19}$$

Then the dual is given by:

$$\text{minimize} \quad \sum_{u} y_u + \sum_{v} y_v \tag{2.20}$$

$$\text{subject to} \quad y_u + y_v \geq w_{uv}, \quad \text{for all } (u,v) \in E \tag{2.21}$$

$$y_u, y_v \geq 0. \tag{2.22}$$

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 (vertices) must pool "money" in order to pay at least that cost. So instead of edges being covered or not covered, there is a certain "amount" that they have to be covered.

These two problems, the maximum weight matching and the minimum weight vertex cover, are the problems that the Hungarian algorithm solves, which we discuss in Chapter 3. For now, we digress and discuss the max-flow min-cut theorem, as an ode to students who have taken a course in algorithms.

2.2.2 The Max-Flow Min-Cut Theorem

Here we demonstrate that a common problem discussed in algorithms courses, and a quite amazing theorem, is really a special case of what we have been discussing. We first recall the definition of a flow network.

Definition 2.8. A *flow network* $G = (V, E)$ is a directed graph in which each edge $(u, v) \in E$ has an assigned *capacity* $c_{uv} \geq 0$. Furthermore, there are two distinguished vertices in V, a source s and a sink t. We assume that every $v \in V$ lies in some path $s \to \cdots \to v \to \cdots \to t$.

Given a flow network, we can define a function on the network.

Definition 2.9. Let $G = (V, E)$ be a flow network. A *flow* on G is a function $f : V \times V \to \mathbb{R}$ that satisfies the following:

- Capacity constraint: For all $u, v \in V$, $0 \leq f(u, v) \leq c_{uv}$.
- Flow 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 to be $val(f) = \sum_{v \in V} f(s, v) - \sum_{v \in V} f(v, s)$, which is just the flow out of our source minus the flow into our source.

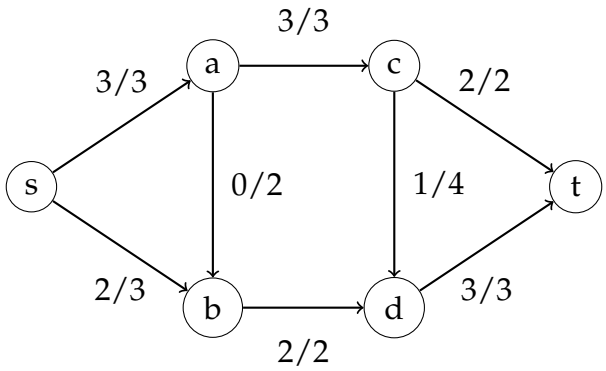

Figure 2.3: Flow network with maximum flow of 5. Notation: X/Y means an edge has capacity Y and flow X, given by the flow function.

In Figure 2.3, (from Cormen, Leiserson, Rivest, and Stein [5]) we have a flow network with a corresponding flow. For example, the capacity of the edge (s, b) is 3, and our flow function in sending a flow of value 2 through this edge.

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 looks for augmenting paths in the residual graph (essentially the subgraph where the flows $f(u, v) < c_{uv}$. The reader should refer to Cormen, Leiserson, Rivest, and Stein [5] for a more detailed treatment. Now, we recall the definition of a cut in a flow network.

Definition 2.10. An $s - t$ *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 defined as $c(S, T) = \sum_{u \in S} \sum_{v \in T} c(u, v)$. 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 course is that, given a flow network G, the value of the maximum flow on G 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, following the treatment given by Vazirani [12]. We will first give a linear program for the maximum flow problem. To make things simpler, let us introduce an arc of infinite capacity from the sink t to the source s; this converts this to a *circulation* problem, 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:

$$\begin{aligned}
\text{maximize} \quad & f(t,s) && && (2.23)\\
\text{subject to} \quad & f(u,v) \leq c_{uv}, && (u,v) \in E && (2.24)\\
& \sum_{v:(v,u)\in E} f(v,u) \leq \sum_{v:(u,v)\in E} f(u,v), && u \in V && (2.25)\\
& f(u,v) \geq 0. && (u,v) \in E && (2.26)
\end{aligned}$$

It is not immediately obvious why the second set of inequalities implies flow conservation; at first glance 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 inequalities (2.24) and (2.25), respectively. Then the dual linear program is as follows:

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

It is known that extreme point solutions to these linear programs take on values zero or one at each coordinate. Let us now consider an optimal dual solution $(\mathbf{d}^*, \mathbf{p}^*)$. First, in order to satisfy $p_s^* - p_t^* \geq 1$ with zero/one 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 vertices with value one, and T the vertices 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 can take value zero without violating the constraints (in fact we want these $d_{uv}^* = 0$ 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.

To make this clear, let us look back at our flow network in Figure 2.3. We expect that, given our discussion of the dual, we should be able to find a cut (S, T) of the flow network that has value five. Moreover, this cut (S, T) should be such that S consists of vertices with value one, and T consists of vertices with value zero. We know that we must set $p_s = 1$ and $p_t = 0$. This gives us the following dual solution, where grey vertices are in S and white vertices are in T:

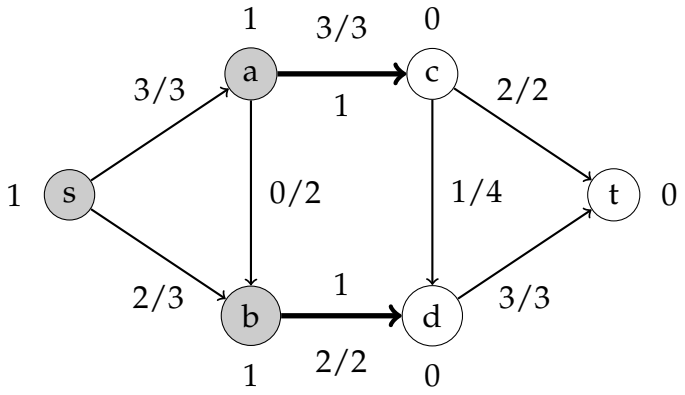

Figure 2.4: Flow network with corresponding dual solution (bold edges are the edges in our cut).

In Figure 2.4, the $s - t$ cut corresponds to the sets $S = \{\mathrm{s}, \mathrm{a}, \mathrm{b}\}$ and $T = \{\mathrm{c}, \mathrm{d}, \mathrm{t}\}$; the edges in the cut are the bold edges (a, c) and (b, d). So the capacity of the cut is five, which is what we'd expect. In order to make the figure readable, we leave out our dual

labeling on edges that are not in the cut (their label is "0" anyways). Edges in the cut have label "1", as shown in the figure. Note that our dual labeling of edges and vertices in this flow network satisfies our dual constraints.

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

Chapter 3

The Hungarian Algorithm

In this chapter, we present a comprehensive look at the Hungarian algorithm. The algorithm relies on some interesting techniques, which we spell out carefully. Ultimately, we show how this algorithm solves its associated primal and dual linear programs – the maximum weight matching and the minimum weight vertex cover problems, respectively.

3.1 Preliminaries

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 vertex on the left, and from each such vertex building a collection of alternating paths. For our graphs $G = (U, V, E)$ we will assume $|U| = |V|$.

Let us remind ourselves what the primal-dual linear programs motivate. We want to minimize $\sum w_{uv}x_{uv}$ and maximize $(\sum_u y_u + \sum_v y_v)$. Moreover, we want optimal solutions (i.e. $\sum w_{uv}x_{uv} = \sum_u y_u + \sum_v y_v$). 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 3.1. A *vertex labeling* on a weighted bipartite graph $G = (U, V, E)$ is a function $l : U \cup V \to \mathbb{N}$. We call a 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; a feasible labeling is a feasible dual solution. It will be helpful for us to look at a certain subset of our graph where the labeling is exact (i.e. where $l(u) + l(v) - w_{uv} = 0$).

Definition 3.2. 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 3.1 we show a bipartite graph along with its corresponding equality graph.

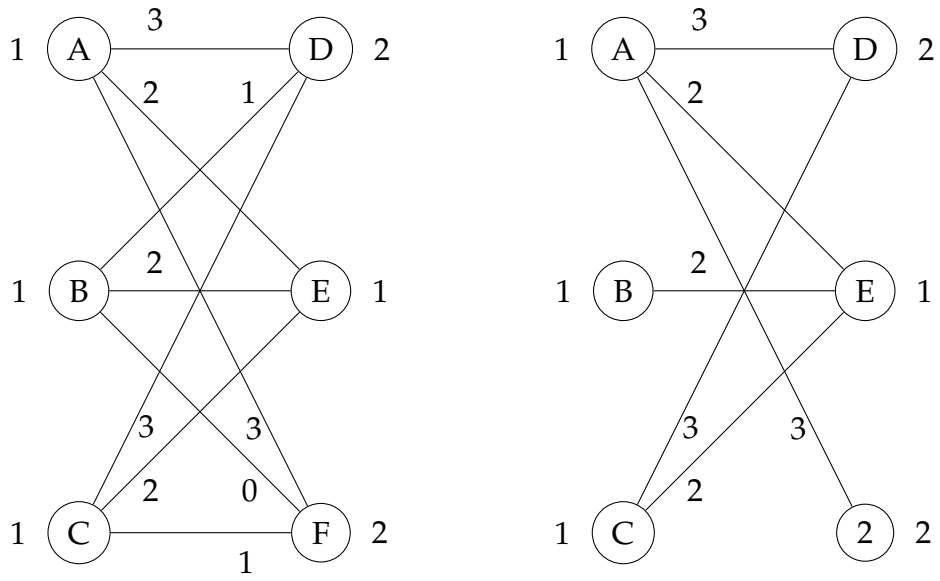

Figure 3.1: A weighted bipartite graph (left) and its corresponding equality subgraph (right).

Theorem 3.3. *(Kuhn-Munkres) If l is a feasible labeling and M is a perfect matching in G_l then M is a maximum weight matching.*

Proof. Let M' be a perfect matching in G. Since every $u \in U \cup V$ is included in exactly

one edge in $M^{'}$, then

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

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

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

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

□

This theorem tells us that if we can give an algorithm for finding a perfect matching in the equality subgraph (if such a matching exists), then we can find a maximum weight matching in the graph. To do this, we will first define a few terms.

Let M be our matching on G. As in Chapter 2, let R be the set of vertices reachable from unmatched vertices in U via M-alternating paths, and let $S := R \cap U$, $T := R \cap V$. We will use these sets to construct our collection of alternating trees.

Now, consider a feasible labeling l on G and set

$$\delta := \min_{\substack{u\in S,\ v\notin T \\ (u,v)\in E}} \{l(u) + l(v) - w_{uv}\}.$$

We define a new labeling $l^{'}$ as follows:

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

Lemma 3.4. The labeling $l^{'}$ as defined above is a feasible labeling.

Proof. To show this labeling is feasible, we must show that for any edge (u, v), $l^{'}(u) + l^{'}(v) \geq w_{uv}$. For $u \in U, v \in V$, there are four possibilities:

1. If $u \in S$ and $v \in T$ then $l^{'}(u) + l^{'}(v) = l(u) + l(v)$. This is because $l^{'}(u) = l(u) - \delta$ and $l^{'}(v) = l(v) + \delta$.

2. If $u \notin S$ and $v \notin T$, $l^{'}(u) = l(u)$ and $l^{'}(v) = l(v)$, so $l^{'}(u) + l^{'}(v) = l(u) + l(v)$.

3. If $u \in S$ and $v \notin T$, $l^{'}(u) = l(u) - \delta$ and $l^{'}(v) = l(v)$. We know $\delta = \min_{u \in S, v \notin T}\{l(u) + l(v) - w_{uv}\}$, which means $\delta \leq l(u) + l(v) - w_{uv}$, and thus $l^{'}(u) + l^{'}(v) \geq w_{uv}$.

4. If $u \notin S$ and $v \in T$, $l^{'}(u) = l(u)$ and $l^{'}(v) = l(v) + \delta$. So we get $l^{'}(u) + l^{'}(v) = l(u) + l(v) + \delta \geq w_{uv}$.

□

We also need to show that this labeling increases the size of our equality graph.

Lemma 3.5. Given an initial feasible labeling l and the labeling $l^{'}$, the following hold:

1. If $(u, v) \in E_l$ and $u \in S, v \in T$, then $(u, v) \in E_{l'}$.

2. If $(u, v) \in E_l$ and $u \notin S, v \notin T$, then $(u, v) \in E_{l'}$.

3. If $u \notin S$ and $v \in T$, then $(u, v) \notin E_l$.

4. If $u \in S$ and $v \notin T$ then $(u, v) \notin E_l$.

5. There exists some (u, v) such that $u \in S$ and $v \notin T$, and $(u, v) \in E_{l'}$.

6. $E_l \subsetneq E_{l'}$.

Proof. Claim (6) will follow from proving the other claims. Claims (1) and (2) are made clear by the previous lemma.

To prove claim (3), we suppose $(u, v) \in E_l$ (towards a contradiction), and consider the two possible cases: (a) is that $(u, v) \in M$, and (b) is that $(u, v) \notin M$. If it is the case that (a) holds, then note that since $v \in T$ we know that u is reachable by some alternating path in R. This means $u \in S$, which is a contradiction. If it is the case that (b) holds, simply note that $u \in S$ is free, meaning it is the root of some alternating tree. This means it must be in S, which is a contradiction.

Part (4) follows from a similar argument to that of (3).

To see (5), note that there is some edge (u, v) with $u \in S, v \notin T$ such that $\delta = l(u) + l(v) - w_{uv}$, so when we take $l'(u) = l(u) - \delta$ and $l'(v) = l(v)$, we get

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

This is exactly what it means for an edge (u, v) to be in E_l. Hence $E_l \subset E_{l'}$. □

Theorem 3.6. $|E_l| < |E_{l'}|$.

The proof of this theorem follows from the previous lemma, as we showed that $E_l \subset E_{l'}$.

3.2 The Hungarian Algorithm

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 algorithm is shown in Figure 3.2.

Line 4 of the algorithm searches for a maximum-cardinality matching in our equality

HUNGARIAN-MAX-ASSIGN(U, V, E, w)
1 $l_u := \max_v w_{uv}$
2 $l_v := 0$
3 Repeat the following:
4 $\quad M :=$ MAX-CARD-MATCHING(U, V, E_l)
5 $\quad$ **if** M is a perfect matching in G_l
6 $\quad\quad$ **return** M
7 $\quad R :=$ ALT-PATH-REACHABLE(U, V, E_l, M)
8 $\quad l :=$ LABEL-UPDATE(R, U, V, E, w, l)

Figure 3.2: The Hungarian Algorithm

subgraph. Line 7 builds our collection of vertices reachable from unmatched vertices in U, and line 8 updates the labeling.

The correctness of the algorithm follows from the Lemmas 3.4, 3.5, and Theorem 3.6. Note that our labeling l remains feasible by Lemma 3.4 and the equality graph increases in size by Lemma 3.5. This tells us that, assuming our graph G has a perfect matching, one will eventually be found by this algorithm. Since this perfect matching will be in G_l, Theorem 3.3 tells us that our final matching M will be of maximum weight.

We now provide an example of this algorithm. This example is due to Golin [7]. Our

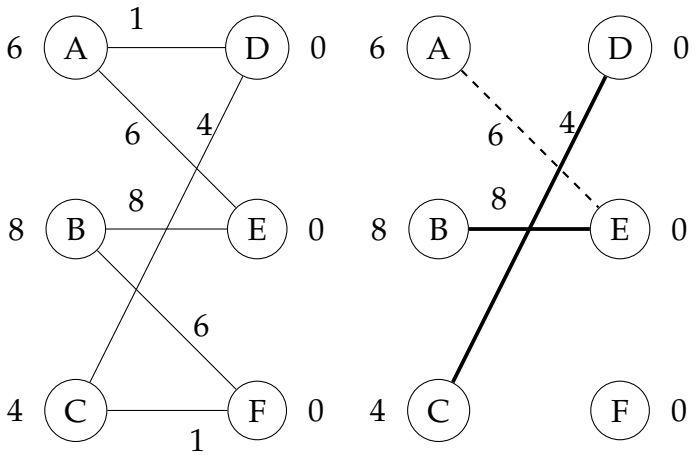

Figure 3.3: Bipartite graph (left) and corresponding equality graph (right) with initial matching

initial matching is $M = \{(\mathrm{B}, \mathrm{E}), (\mathrm{C}, \mathrm{D})\}$ (see Figure 3.3). This matching M is not perfect in G_l. Initially, we have $R = \emptyset$. We will again look at the sets $S = R \cap U$ and $T = R \cap V$. The set of unmatched vertices in U consists of a single member, A, meaning our algorithm chooses A to add to R. So we have $S = \{\mathrm{A}\}$ and $T = \{\mathrm{E}\}$, since E is connected to A in G_l via the single alternating path. The vertex E is matched, so we grow our alternating tree as follows: $S := S \cup \{\mathrm{B}\} = \{\mathrm{A}, \mathrm{B}\}$, $T := T \cup \{\mathrm{E}\} = \{\mathrm{E}\}$. At this point we adjust our labeling. Calculate $\delta = \min_{u \in S, v \notin T}\{l(u) + l(v) - w_{uv}\} = 2$ from edge (B, F). Our new equality graph is shown in Figure 3.4.

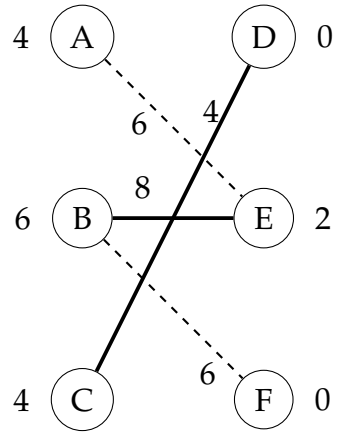

Figure 3.4: Second equality graph.

Now, $S = \{\mathrm{A}, \mathrm{B}\}$ is the same, but $T = \{\mathrm{E}, \mathrm{F}\}$ has changed (we've implicitly updated R). The vertex F is unmatched, meaning it is an endpoint of an augmenting path. In particular, $p = \mathrm{A} \to \mathrm{E} \to \mathrm{B} \to \mathrm{F}$ is an augmenting path. Thus we can improve our matching with $M := M \oplus p = \{(\mathrm{A}, \mathrm{E}), (\mathrm{B}, \mathrm{F}), (\mathrm{C}, \mathrm{D})\}$. Our equality graph with the new matching is given in Figure 3.5.

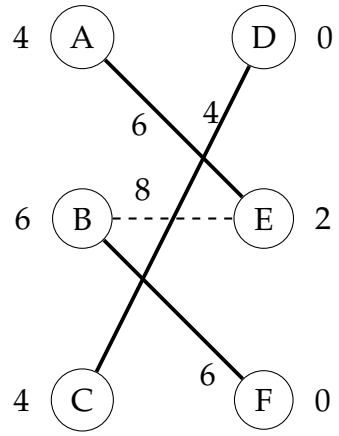

Figure 3.5: Equality graph after augmenting.

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 one 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

The 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 restrict 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- or maximum- cost subset $E' \subset E$ that satisfies some criteria. Our maximum-weight matching problem is an example of this.

We will also look at a two algorithms for solving auctions. Oftentimes, auctions can be modelled by weighted bipartite graphs, and we approach them as a special case of the weighted bipartite matching problem. However, we will see that in general we do not attain exact solutions, only approximate ones. This will hint at some of the further applications of the primal-dual method.

4.1 The Classical Primal-Dual Method

We begin by looking at what is known as the "classical" primal-dual method, which is concerned with linear programs for polynomial-time solvable optimization problems.

Let us 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 has secretly been at work throughout this thesis.

Definition 4.1. (Complementary slackness) Given two linear programs in the form above, the *primal complementary slackness conditions* are the conditions:

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

where A^j is the jth column of A. Similarly, the *dual complementary slackness conditions* are the conditions:

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

where A_i is the ith row of A.

These conditions say that for a positive solution in the primal (dual), the correspond-

ing constraint in the dual (primal) is strict. What is surprising is that together, these conditions give us necessary and sufficient conditions for solving a primal-dual system of linear programs, 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 4.2. *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:*

$$x_j t_j = 0 \quad \textit{for all } j \tag{4.7}$$

$$y_i s_i = 0 \quad \textit{for all } i \tag{4.8}$$

Before we prove this, note that this says we cannot have optimality *and* have slackness in both a variable of the primal and a dual constraint, and vice versa; we can only have slackness in one of them.

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 (4.7) and (4.8) 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 ($s_i > 0$), then additional units of that resource must have no value ($y_i = 0$); if there is slack in the dual price constraint ($t_j > 0$) there must be a shortage of that resource ($x_j = 0$).

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:

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

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

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

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

and its dual

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

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

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

The format here is a little different than what we have been working with in the thesis thus far, since our primal is a maximization problem and the dual is a minimization, but it is not difficult to switch between the two. 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 (4.16) and (4.18); the algorithm then at each step works to decrease the number of conditions in (4.17) that are unsatisfied, while maintaining satisfiability of (4.16) and (4.18). 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 (4.5) 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. 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:

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

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

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

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

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

Observe that if this restricted primal has a feasible solution $(\mathbf{x}, \mathbf{s})$ such that the objective function evaluates to zero, 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 zero, more work is required. We can then consider the dual of the restricted primal:

$$\text{maximize } \mathbf{b}^T \mathbf{w} \tag{4.24}$$

$$\text{subject to } A^j \mathbf{w} \leq 0, \quad \text{for all } j \in A \tag{4.25}$$

$$A^j \mathbf{w} \leq 1, \quad \text{for all } j \notin A \tag{4.26}$$

$$w_i' \geq -1, \quad \text{for all } i \notin B \tag{4.27}$$

$$w_i' \geq 0. \quad i \in B \tag{4.28}$$

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 $A^T y' \leq c$. 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 us consider the first one. When $w_i > 0$, we are fine since clearly $y_i + \epsilon w_i > 0$. However, we need to be careful when $w_i < 0$ since this could potentially violate the inequality (4.29). So we need that

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

Now let's address the second inequality (4.30). When $A_j^T w \leq 0$, we again don't need to worry. We need to be careful about violating the constraint when $A^T w > 0$. Thus, we need that

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

If we choose the minimum of these two ϵ values, we obtain a new feasible dual solution that has greater objective value than our original dual solution. 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 The 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.

$$\text{minimize} \quad \sum_{u,v} c_{uv} x_{uv} \tag{4.31}$$

$$\text{subject to} \quad \sum_{v} x_{uv} \geq 1, \quad \text{for all } u \in U \tag{4.32}$$

$$\sum_{u} x_{uv} \geq 1, \quad \text{for all } v \in V \tag{4.33}$$

$$x_{uv} \geq 0. \tag{4.34}$$

Its dual is

$$\text{maximize} \quad \sum_{u} y_u + \sum_{v} y_v \tag{4.35}$$

$$\text{subject to} \quad y_u + y_v \leq c_{uv}, \quad \text{for all } u \in U,\ v \in V \tag{4.36}$$

$$y_u, y_v \geq 0. \tag{4.37}$$

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 comple-

mentary slackness conditions. 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

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

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

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

$$\text{minimize} \sum_{u\in U} s_u + \sum_{v\in v} s_v \tag{4.38}$$

$$\text{subject to} \quad \sum_v x_{uv}, -s_u = 1, \qquad \text{for all } u \in U \tag{4.39}$$

$$\sum_u x_{uv}, -s_v = 1, \qquad \text{for all } v \in V \tag{4.40}$$

$$x_{uv} = 0, \quad \text{for all } (u,v) \notin A \tag{4.41}$$

$$x_{uv} \geq 0, \quad \text{for all } (u,v) \in A \tag{4.42}$$

$$\mathbf{s} \geq 0. \tag{4.43}$$

It is a fact that all components of this restricted primal take on values zero or one, 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 subgraph $G' = (U, V, A)$ 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:

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

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

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

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

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

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

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 by which we modify our labeling by in the Hungarian algorithm. Thus we've found an ϵ that maintains dual feasibility, and increases the objective 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 matchings, given by Bertsekas [1]. 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 bidder b must pay p_g to receive this item. Suppose that the amount bidder b is willing to pay for good g, or b's valuation of g, is c_{bg}. Then the net value of this item g to bidder b is $c_{bg} - p_g$. Let's also adopt the notation that $A(b)$ is the set of all items bidder b is interested in (willing to pay for). Assuming that each bidder b is a rational agent acting in their own best interest, each bidder b would want to be assigned a good g^* that maximizes their net value, i.e. for all bidders b with goods g^*, we have that

$$c_{bg^*} - p_{g^*} = \max_{g \in A(b)} \{c_{bg} - p_g\}. \tag{4.50}$$

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 an 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. It will be useful for us to consider an alternate but equivalent dual problem, which is suggested by our complementary slackness conditions. For a more detailed treatment of this equivalence, see Bertsimas and Tsitsiklis [2]. In the auction setting, it does not make sense to have a labeling on the vertices in B (the set of buyers) whereas in the usual dual we have variables on both sets of vertices that partition

our graph. The usual dual is as follows:

$$\text{minimize} \sum_{b=1}^{n} r_b + \sum_{g=1}^{n} p_g \tag{4.51}$$

$$\text{subject to} \qquad r_b + p_g \geq c_{bg}, \quad \text{for all } (b,g) \in E \tag{4.52}$$

$$r_b, p_g \geq 0. \qquad b \in B,\ g \in G \tag{4.53}$$

In the auction setting, only values of p_g make sense; "prices" (r_b) on the bidders is meaningless (unless our auction is two-way, which we are not considering). Note that $\sum_b r_b$ is minimized (while still maintaining feasibility) when we set each r_b to the largest value allowed by the constraints (4.52), i.e. given p_g for $g = 1, \ldots, n$, set

$$r_g := \max_{g=1,\ldots,n} \{c_{bg} - p_g\}.$$

This gives us the following equivalent dual objective function

$$\text{minimize } \{\sum_b \max_g \{c_{bg} - p_g\} + \sum_g p_g\}, \tag{4.54}$$

which is unconstrained (except for the usual nonnegativity requirement). Note that this is an exact translation of our complementary slackness conditions (4.50). Again, this illustrates the importance of complementary slackness in the creation of algorithms for combinatorial optimization.

4.3.1 The Naive Algorithm

We will proceed by first describing a naive auction algorithm, which, although clever, does not always work. Nonetheless, we will use it as motivation for our final auction algorithm.

Throughout the algorithm, we wil keep track of two sets of bidders – those that are

currently assigned a good g, and those that are not. 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^* such that $c_{bg^*} - p_{g^*} = \max_{g \in A(b)}\{c_{bg} - p_g\}$. We will then need to do two things: (1), replace the current owner $b^{'}$ of g^* (put them back into the set of unassigned bidders) with b; (2) bump the price of g^* by some amount δ_b. Intuitively, we want to increase p_{g^*} by the largest amount that still maintains the fact that g^* offers maximal value to b. We can easily accomplish this by setting $\delta_b = v_b - w_b$, where $v_b = \max_{g \in A(b)}\{c_{bg} - p_g\} = p_{g^*}$ and $w_b = \max_{g \in A(b),\, g \neq g^*}\{c_{bg} - p_g\}$. This simply says that we are bumping the price p_{g^*} by the difference in the profit of the best object and the profit of the second best object for the current bidder. That is the essence of the naive algorithm: choose an unassigned bidder b, find an item that maximizes their individual value, bump the price of that good by δ_b, and unassign the previous owner. We do this until all bidders are assigned.

This is a simple algorithm that has one major problem: there are cases in which it does not terminate. The issue is that the bidding increment δ_b is zero when more than one object offers maximum value to a bidder b. Consequently, a situation may arise in which a number of bidders contend for a smaller number of equally desirable objects, creating an infinite loop. An example of this is shown in Figure 4.1, taken from Bertsekas [1]. In

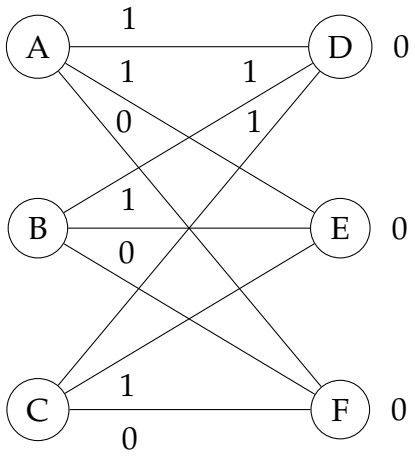

Figure 4.1: Example where the naive algorithm does not terminate.

this figure the initial prices on all goods is zero. All bidders desire objects D and E since

they offer a benefit of $1 - 0 = 1$, while object F offers a benefit of zero to all bidders. Say we first assign bidder A to object D. Then the algorithm will cycle as bidders B and C compete for item E without its price being raised.

The issue with our naive algorithm can be remedied by adding a small perturbation term to our complementary slackness, which will break the cycles that cause problems like in Figure 4.1.

4.3.2 The Auction Algorithm

In real auctions it is often the case that each bid for an object must raise the object's price by some minimum positive increment ϵ. This will ensure that no matter what, whenever someone is assigned an item, the price raises by some non-zero amount. We will say that an assignment and a price vector satisfy *ϵ-complementary slackness* if

$$c_{bg^*} - p_{g^*} \geq \max_{g \in A(b)} \{c_{bg} - p_g\} - \epsilon. \tag{4.55}$$

The new auction algorithm will be the same as the naive one, except in the way we adjust prices; now $\delta_b := v_b - w_b + \epsilon$. We give the pseudocode in Figure 4.2. We will use a data structure called a queue, denoted $\mathcal{Q}$, to keep track of unassigned bidders. A queue works exactly how one might think: new elements are placed at the "end," or enqueued, and members are removed from the "top," or dequeued.

We now discuss why this algorithm terminates in the case where our graph is fully dense. The first thing we should note is that once all goods receive a bid the algorithm is done, since each person bids on the good they currently desire most. Now, suppose there is some good that receives k bids. This will increase the price of this good by at least $k\epsilon$. Clearly there is some k for which the good becomes less desirable than some other good that has not received any bids thus far. So it must be the case that a good can only receive a finite number of bids before it is considered inferior to some other good that has

```
AUCTION ALGORITHM(B, G, E, c)
 1  For each g ∈ G, set p_g := 0
 2  Queue Q := B.
 3  while Q ≠ ∅
 4        b = Q.deque()
 5        Find g* ∈ A(b) that maximizes c_bg − p_g
 6        Let b' be the bidder currently assigned to g* (null if none)
 7        if c_bg* − p_g* ≥ 0
 8              Q.enqueue(b')
 9              Assign g* to b
10              p*_g = p*_g + δ_b
11  return Assignment and price vector
```

Figure 4.2: Pseudocode for the auction algorithm.

not been bid on. We know this because prices *always* increase when a bid/assignment is made. For the case in which our graph is not fully dense, see Bertsekas [1], Appendix 2.

Now we need to discuss how satisfying ϵ-complementary slackness affects our solutions. Because we are relaxing our original complementary slackness constraint, we should not in general expect to get optimal solutions. In fact, optimality of our solution relies entirely on how we choose our ϵ.

It turns out that whatever ϵ is, an assignment of bidders and a price vector that together satisfy ϵ- complementary slackness will be within $n\epsilon$ of optimal, where $n = |B|$. We state this formally.

Theorem 4.3. *A feasible assignment of bidders along with a price vector* $\mathbf{p}$*, which together satisfy* ϵ*-complementary slackness, is within* $n\epsilon$ *of being optimal. Furthermore, the price vector is within* $n\epsilon$ *of being an optimal dual solution.*

Proof. Let $\mathbf{OPT}(\Gamma)$ be the value of a total optimal assignment, and $\mathbf{OPT}(\Omega)$ the corre-

sponding dual cost. So

$$\mathbf{OPT}(\Gamma) = \max_{g^* \in A(b)} \sum_b c_{bg^*} \tag{4.56}$$

$$\mathbf{OPT}(\Omega) = \min_{p_g} \{ \sum_b \max_{g \in A(b)} \{c_{bg} - p_g\} + \sum_g p_g \}, \tag{4.57}$$

where in (4.56) we specify that no good is assigned to two bidders, and no bidder is assigned to two goods.

Firstly, weak duality tells us that $\mathbf{OPT}(\Gamma) \leq \mathbf{OPT}(\Omega)$. Since our assignment and prices satisfy ϵ-complementary slackness, we know that for all b, g

$$\max_{g \in A(b)} \{c_{bg} - p_g\} - \epsilon \leq c_{bg^*} - p_{g^*}.$$

This means that we have

$$\mathbf{OPT}(\Omega) \leq \sum_b \left(\max_{g \in A(b)} \{c_{bg} - p_g\} + p_{g^*} \right) \leq \sum_b c_{bg^*} + n\epsilon \leq \mathbf{OPT}(\Gamma) + n\epsilon.$$

This shows that both our primal and dual objective functions are within $n\epsilon$ of optimality.

□

Note that in the case where all c_{bg} are integers (which is usually the case in auctions), choosing an $\epsilon < 1/n$, gives an optimal output for both the primal and dual solutions.

4.4 Concluding Remarks

The primal-dual method described in this chapter is just a glimpse into what is a fruitful area of active research. Ultimately, the goal is to use this method to develop approximation algorithms for problems known to be NP-hard. Unfortunately, we did not have the time to thoroughly explore this facet in this thesis. However, this thesis provides an in-

troduction to the method so that perhaps in the future some dilligent student may look into this method as it applies to problems in NP.

References

[1] D. P. Bertsekas. Auction algorithms for network flow problems: A tutorial introduction. *Computational Optimization and Applications*, 1(1):7–66, Oct 1992.

[2] D. Bertsimas and J.N. Tsitsiklis. *Introduction to Linear Optimization*. Athena Scientific series in optimization and neural computation. Athena Scientific, 1997.

[3] G. Birkhoff. Tres observaciones sobre el algebra lineal. *Univ. Nac. Tucuman Rev. Ser. A*, 5:147–151, 1946.

[4] J.A. Bondy and U.S.R. Murty. *Graph Theory with Applications*. Elsevier Publishing, 1976.

[5] T.H. Cormen, C.E. Leiserson, R.L. Rivest, and C. Stein. *Introduction to Algorithms*. MIT Press, 2009.

[6] G. Demange, D. Gale, and M. Sotomayor. Multi-item auctions. *Journal of Political Economy*, 94(4):863–872, 1986.

[7] M. Golin. Bipartite matching and the hungarian method. `http://www.cse.ust.hk/~golin/COMP572/Notes/Matching.pdf`.

[8] D. S. Hochbaum, editor. *Approximation Algorithms for NP-hard Problems*. PWS Publishing Co., Boston, MA, USA, 1997.

[9] E.L. Lawler. *Combinatorial Optimization: Networks and Matroids*. Dover Books on Mathematics Series. Dover Publications, 1976.

[10] N. Nisan. Auction algorithm for bipartite matching. `https://agtb.wordpress.com/author/algorithmicgametheory/`.

[11] C.H. Papadimitriou and K. Steiglitz. *Combinatorial Optimization: Algorithms and Complexity*. Dover Books on Computer Science. Dover Publications, 2013.

[12] V.V. Vazirani. *Approximation Algorithms*. Springer Berlin Heidelberg, 2002.

[13] J. Von Neuman and Shapley. *Contributions to the Theory of Games (AM-28), Volume II*. Princeton University Press, 1953.