Notes on Approximation Algorithms

Bhargav Samineni

Abstract

A collection of some notes on the design and analysis of approximation algorithms and approximation techniques. Based mainly off of [1, 2].

1 Introduction to LP-Duality

For a minimization linear program in canonical form, the goal is to find a non-negative, rational vector x that minimizes a given linear objective function in x subject to some linear constraints on x. If there are n decision variables and m linear constraints, then $x \in \mathbb{Q}^n$, the coefficients of the linear objective function can be represented by a vector $c \in \mathbb{Q}^n$, the coefficients of the linear constraints can be represented by a matrix $A = (a_{ij}) \in \mathbb{Q}^{m \times n}$, and the values of the linear constraints can be represented by a vector $b \in \mathbb{Q}^m$.

Definition 1.1 (Primal and Dual)

Given a linear programming problem in canonical form denoted as (P), we can induce a problem denoted by (D) with the following form:

$$
\begin{array}{llll}
(P) & \text{minimize} & \sum_{j=1}^{n} c_j x_j & \\
 & \text{subject to} & \sum_{j=1}^{n} a_{ij} x_j \geq b_i & i = 1, \ldots, m \\
 & & x_j \geq 0 & j = 1, \ldots, n
\end{array}
\qquad
\begin{array}{llll}
(D) & \text{maximize} & \sum_{i=1}^{m} b_i y_i & \\
 & \text{subject to} & \sum_{i=1}^{m} a_{ij} y_i \leq c_j & j = 1, \ldots, n \\
 & & y_i \geq 0 & i = 1, \ldots, m
\end{array}
$$

where a_{ij}, b_i, and c_i are given rational numbers and y_i corresponds to the ith inequality of (P).

Definition 1.2 (Primal)

The problem (P) is referred to as the *primal*.

Definition 1.3 (Dual)

The *dual* of the primal is problem (D).

Every feasible solution for the dual serves as a lower bound on the optimal objective function value of the primal. The reverse also holds in that every feasible solution to the primal serves as an upper bound on the optimal objective function value of the dual.

Theorem 1.1 (Weak Duality). *If $x = (x_1, \ldots, x_n)$ is a feasible solution to (P) and $y = (y_1, \ldots, y_m)$ a feasible solution to (D), then $\sum_{j=1}^{n} c_j x_j \geq \sum_{i=1}^{m} b_i y_i$.*

Proof.

$$\sum_{j=1}^{n} c_j x_j \geq \sum_{j=1}^{n} \left(\sum_{i=1}^{m} a_{ij} y_i \right) x_j = \sum_{i=1}^{m} \left(\sum_{j=1}^{n} a_{ij} x_j \right) y_i \geq \sum_{i=1}^{m} b_i y_i$$

■

As a consequence of Theorem 1.1, if we find that there exist some feasible solutions to (P) and (D) that have matching objection function values, then these solutions must be optimal.

Theorem 1.2 (Strong Duality/LP-Duality). *If (P) and (D) are both feasible, and $x^* = (x_1^*, \ldots, x_n^*)$ and $y^* = (y_1^*, \ldots, y_m^*)$ are optimal solutions to (P) and (D), respectively, then $\sum_{j=1}^{n} c_j x_j^* = \sum_{i=1}^{m} b_i y_i^*$.*

As a corollary of Theorem 1.2, we find that optimal solutions to (P) and (D) must satisfy a set if conditions.

Corollary 1.1 (Complementary Slack Conditions). *Let x and y be feasible solutions to (P) and (D), respectively. Then x and y are both optimal iff the following are satisfied:*

1. *For each $1 \leq j \leq n$, either $x_j = 0$ or $\sum_{i=1}^{m} a_{ij} y_i = c_j$*
2. *For each $1 \leq i \leq m$, either $y_i = 0$ or $\sum_{j=1}^{n} a_{ij} x_j = b_i$.*

1.1 Algorithm Design Techniques

Definition 1.4 (Extreme Point Solution)

Consider the polyhedron defining the set of feasible solutions to an LP. A feasible solution is called an *extreme point solution* if it is a vertex of the polyhedron.

Theorem 1.3. *For any objective function, there is an extreme point solution that is optimal.*

Combinatorial optimization problems can often be stated as integer programs (IP), which can be relaxed to an LP. For the case of NP-Hard problems, however, the polyhedron that defines the set of feasible solutions to the LP-relaxation may not have integer vertices. With the LP-relaxation, the goal is to find near optimal integral solutions.

One method to get an approximation algorithm is through *LP-rounding*. Rounding involves solving the LP-relaxation and the converting the fractional solution to an integral solution while ensuring the in the process the cost does not increase dramatically. The approximation ratio is given by comparing the cost of the integral and fractional solutions.

Another method is the *primal-dual schema*. Consider an LP-relaxation as the primal program. Under the scheme, an integral solution to the primal and a feasible solution to the dual are constructed iteratively. Any feasible solution to the dual also provides a lower bound on OPT. The approximation ratio is given by comparing the two solutions.

1.1.1 Integrality Gap

Definition 1.5 (Integrality Gap)

Given an LP-relaxation for a minimization problem, let $\mathrm{OPT}_f(I)$ be the cost of an optimal fractional solution to an instance I of the problem. Similarly, let $\mathrm{OPT}(I)$ denote the cost of an optimal solution to the original IP for an instance I of the problem. The *integrality gap* of the relaxation is

$$\sup_I \frac{\mathrm{OPT}(I)}{\mathrm{OPT}_f(I)}.$$

In the case of a maximization problem, the integrality gap is given by the infimum of the ratio.

When the integrality gap of an LP-relaxation is exactly 1, such a relaxation is called an *exact relaxation*. If the cost of a solution found by an algorithm is compared to the cost of an optimal fractional solution (or feasible dual solution), the best approximation factor one can hope to achieve is the integrality gap of the relaxation.

2 Set Cover

Problem 2.1 (Set Cover)

Given a ground set $U = \{e_1, \dots, e_n\}$, a collection $\mathcal{S} = \{S_1, \dots, S_m\}$ of subsets of U, and a cost function $c \colon \mathcal{S} \to \mathbb{Q}^+$, find a minimum cost subcollection of $\mathcal{S}$ that covers all elements in U.

To formulate Set Cover as an IP, denote the binary decision variables by x_j for each $S_j \in \mathcal{S}$. If S_j is in the cover, then $x_j = 1$.

$$\begin{aligned} \text{minimize} \quad & \sum_{j=1}^{m} c(S_j) x_j \\ \text{subject to} \quad & \sum_{j\colon e_i \in S_j} x_j \geq 1 \quad & i = 1, \dots, n \\ & x_j \in \{0,1\} & j = 1, \dots, m \end{aligned} \tag{1}$$

The LP-relaxation to this IP is given by letting $x_j \geq 0$. A solution to the LP-relaxation can be viewed as a fractional set cover.

$$\begin{aligned} \text{minimize} \quad & \sum_{j=1}^{m} c(S_j) x_j \\ \text{subject to} \quad & \sum_{j\colon e_i \in S_j} x_j \geq 1 \quad & i = 1, \dots, n \\ & x_j \geq 0 & j = 1, \dots, m \end{aligned} \tag{2}$$

Introducing a variable y_i for each $e_i \in U$, we get the dual program.

$$\begin{aligned} \text{maximize} \quad & \sum_{i=1}^{n} y_i \\ \text{subject to} \quad & \sum_{i\colon e_i \in S_j} y_i \leq c(S_j) \quad & j = 1, \dots, m \\ & y_i \geq 0 & i = 1, \dots, n \end{aligned} \tag{3}$$

2.1 Dual-Fitting Analysis

Algorithm 1 defines the dual program's value for each variable as $y_i = p(e_i)$. However, the solution y constructed by this approach is infeasible as it may not satisfy the first constraint. To see this, note that if a set S_j is chosen by the algorithm, it distributes its cost across all its uncovered elements. If $S_j \cap C \neq \emptyset$, then this means that it contains some covered elements that already have some other set's cost distributed to them, which would cause $\sum_{i\colon e_i \in S_j} y_i \geq c(S_j)$. Instead, we may be able to show that some scaled solution $y' = \frac{y}{\kappa}$ where $\kappa > 1$ is feasible for the dual.

Algorithm 1 Greedy Set Cover

$C, I \leftarrow \emptyset$
while $C \neq U$ **do**
    $i = \underset{j:\, S_j \in \mathcal{S}}{\operatorname{argmin}} \frac{c(S_j)}{|S_j \setminus C|}$
    $p(e_i) = \frac{c(S_i)}{|S_i \setminus C|} \qquad \forall e_i \in S_i \setminus C$
    $C \leftarrow C \cup S_i$, $I \leftarrow I \cup i$, $\mathcal{S} \leftarrow \mathcal{S} \setminus S_i$
return C

Lemma 2.1. *The vector $y' = \frac{y}{H_g}$ is a feasible solution for the dual program Eq. (3), where $g = \max_i |S_i|$ and H_g is the gth harmonic number.*

Proof. Assume the algorithm runs for ℓ iterations. Let $a_{j,k}$ be the number of elements in S_j that are still uncovered at the start of the kth iteration (hence $a_{1,j} = |S_j|$), $a_{\ell+1,j} = 0$). Let $A_{j,k}$ denote the set of uncovered elements in S_j that are covered at the end of the kth iteration. If S_i is chosen in the kth iteration, for each covered element $e_i \in A_{i,k}$,

$$y'_i = \frac{c(S_i)}{H_g a_{i,k}} \leq \frac{c(S_j)}{H_g a_{j,k}} \qquad \forall S_j \in \mathcal{S}$$

since S_i minimizes the ratio $\frac{S_i}{a_{i,k}}$. Then for any set S_j,

$$\begin{aligned}
\sum_{i:\, e_i \in S_j} y'_i = \sum_{k=1}^{\ell} \sum_{i:\, e_i \in A_{j,k}} y'_i &\leq \sum_{k=1}^{\ell} (a_{j,k} - a_{j,k+1}) \frac{c(S_j)}{H_g a_{j,k}} \\
&= \frac{c(S_j)}{H_g} \sum_{k=1}^{\ell} \frac{a_{j,k} - a_{j,k+1}}{a_{j,k}} \\
&= \frac{c(S_j)}{H_g} \sum_{k=1}^{\ell} \sum_{l=a_{j,k+1}+1}^{a_{j,k}} \frac{1}{a_{j,k}} \\
&\leq \frac{c(S_j)}{H_g} \sum_{k=1}^{\ell} \left(\frac{1}{a_{j,k}} + \frac{1}{a_{j,k} - 1} + \ldots + \frac{1}{a_{j,k+1} + 1} \right) \\
&\leq \frac{c(S_j)}{H_g} \sum_{l=1}^{|S_j|} \frac{1}{l} = \frac{c(S_j)}{H_g} H_{|S_j|} \leq c(S_j)
\end{aligned}$$

which satisfies the first constraint of Eq. (3). ■

Theorem 2.1. *Algorithm 1 gives a H_g approximation to* Set Cover*.*

Proof. Since the greedy algorithm distributes the cost of the sets it chooses across all the elements, $\sum_{j \in I} c(S_j) = \sum_{i=1}^{n} y_i$. By Lemma 2.1, we know that y' is a feasible solution to the dual, so using Theorem 1.1 we get that

$$\sum_{j \in I} c(S_j) = \sum_{i=1}^{n} y_i = H_g \sum_{i=1}^{n} y'_i \leq H_g \mathrm{OPT}_f \leq H_g \mathrm{OPT}.$$

■

3 Matchings

Definition 3.1 (Matching)

Given a graph $G = (V, E)$, a matching $M \subseteq E$ is a set of edges such that no two edges in M share a common vertex.

Definition 3.2 (Covered/Exposed)

A node is M-covered if some edge in M is incident to it. Else it is M-exposed.

Note that a matching M covers exactly $2|M|$ nodes, leaving $|V| - 2|M|$ nodes exposed. A matching is *maximal* if adding any edge $e \in E \setminus M$ to it causes it to no longer be a matching. A matching is *perfect* if it covers all the nodes. A basic decision problem is to decide if a graph has a perfect matching. The following theorem gives a way to determine if a graph has a perfect matching.

Theorem 3.1 (Tutte's Matching Theorem). *A graph $G = (V, E)$ has a perfect matching iff for every subset A of nodes, $\mathrm{odd}(G \setminus A) \leq |A|$, where $\mathrm{odd}(\cdot)$ denotes the number of connected components having an odd number of nodes.*

A more general problem is to find a maximum cardinality matching.

Problem 3.1 (Maximum Cardinality Matching)

Given a graph $G = (V, E)$, find a matching M that has maximum cardinality. Equivalently, find a matching with the fewest exposed nodes.

3.1 Augmenting Paths

A path P is a collection of edges $(v_0, v_1), \ldots, (v_{k-1}, v_k) = e_1, \ldots, e_k$ where each v_i is distinct.

Definition 3.3 (Alternating Path)

Given a matching M in a graph G, a path P in G is M-alternating if it alternates between edges in M and edges in $E \setminus M$.

Definition 3.4 (Augmenting Path)

Given a matching M in a graph G, a path P in G is M-augmenting if it is M-alternating and its end nodes are distinct and M-exposed.

We use the notation $A \triangle B = (A \cup B) \setminus (A \cap B)$ to denote the symmetric difference of A and B (i.e. the set of elements unique to A and B).

Lemma 3.1. *If M is a matching and P an M-augmenting path, then $M' = M \triangle P$ is a matching that contains one more edge than M.*

Proof. By definition of an M-augmenting path, P is of odd length with edges in M occurring at every even index. Therefore, $M' = M \triangle P$ contains the edges of $M \setminus P$ and edges of P at odd indices, which covers all nodes covered by M plus the end nodes of P and contains exactly one more edge than M. ■

Augmenting paths can be used to identify a Maximum Cardinality Matching in the following way.

Theorem 3.2 (Augmenting Path Theorem)**.** *A matching M in a graph G is maximum iff there is no M-augmenting path.*

Proof. $\Longrightarrow$ (by contrapositive): Direct consequence of Lemma 3.1. ■

Proof. $\Longleftarrow$ (by contrapositive): Let M^* be a maximum matching. Let $Q = M^* \triangle M$. Then each node is incident to at most one edge in $M^* \cap Q$ and one edge in $M \cap Q$. Hence, Q is an edge set of node disjoint paths and circuits where edges alternate between belonging in M^* and M. Because the edges are taken from matchings, all circuits must be of even length and contain the same number of edges from M^* and M. Therefore, since $|M^*| > |M|$, there must be at least one path in Q that contains more edges from M^* than M. Such a path is M-augmenting. ■

3.2 Alternating Trees

Let M be a current matching and X the set of M-exposed nodes.

Definition 3.5 (Alternating Tree)

An M-alternating tree is a tree T with root node $r \in X$ such that along every path to a node v, the path is M-alternating with $e_i \in M$ iff $i = 2k$ for some $k \in \mathbb{Z}^+$.

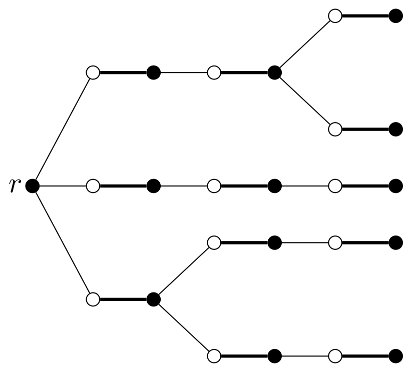

Figure 1: An example of an alternating tree. Nodes in A_T are white while nodes in B_T are black.

We can divide an alternating tree $T = (V_T, E_T)$ into two node sets A_T and B_T such that A_T is the set of nodes at the other end of an odd-length M-alternating path starting at r and B_T is the set of nodes at the other end of an even-length M-alternating path starting at r. Such sets can be built by the following relation: starting with $A_T = \emptyset$ and $B_T = \{r\}$, if $(u, v) \in E$ such that $u \in B_T$ and $v \notin A_T \cup B_T$ and there exists an edge $(v, w) \in M$, then $A_T \leftarrow A_T \cup \{v\}$ and $B_T \leftarrow B_T \cup \{w\}$. Note that $|B_T| = |A_T| + 1$. Additionally, if there exists some edge (i, j) where $i \in B_T$ and $j \notin T$ such that j is M-exposed, then the path $P = (r, v_1), \ldots, (i, j)$ is M-augmenting.

Definition 3.6

An M-alternating tree is *maximal* if for all $b \in B_T$, $N(b) \subseteq V_T$ (i.e. no additional nodes can be added to the tree). It is *frustrated* if for all $b \in B_T$, $N(b) \subseteq A_T$.

Alternating trees are useful in deciding if a graph has a perfect matching in the following way.

Lemma 3.2. *Suppose that G has a matching M and an M-alternating tree T that is frustrated. Then G has no perfect matching.*

Proof. Consider $G \setminus A_T$. The nodes B_T are then single node odd components of $G \setminus A_T$ by definition of a frustrated tree. Therefore,

$$\text{odd}(G \setminus A_T) \geq |B_T| > |A_T|$$

which fails the condition of Theorem 3.1. ∎

3.3 Maximum Cardinality Matchings in the Bipartite Case

Algorithm 2 Maximum Cardinality Matching in Bipartite Graphs

Input: A bipartite graph $G = (V, E)$
Output: A matching M

$X \leftarrow V, M \leftarrow \emptyset, \mathcal{T} \leftarrow \emptyset$ ▷ *X is the set of M-exposed nodes*
while $X \neq \{\text{root}(T_i) : T_i \in \mathcal{T}\}$ **do**
  $T \leftarrow$ MaximalTree(G)
  $\mathcal{T} \leftarrow \mathcal{T} \cup T$
  $G \leftarrow G \setminus T$ ▷ *Remove subgraph T from G*
return M

function MaximalTree$(G = (V, E))$
  Choose $r \in X \cap V$ ▷ *Choose M-exposed vertex in current graph*
  $T \leftarrow ((A, B), E_T)$ where $A = \emptyset$, $B = \{r\}$, $E_T = \emptyset$
  while $\exists e = (u, v) \in E$ where $u \in B$, $v \notin T$ **do**
    if $v \in X$ **then** ▷ *Forms an M-augmenting path*
      Let P be the path from r to v
      $M \leftarrow$ Augment(M, P)
      $X \leftarrow X \setminus \{r, v\}$
      if $X = \emptyset$ **then**
        return $\emptyset$ ▷ *M is a perfect matching*
      else
        Choose $r \in X \cap V$ ▷ *Start a new M-alternating tree*
        $T \leftarrow ((\emptyset, \{r\}), \emptyset)$
    else
      $T \leftarrow$ ExtendTree(T, e)
  return T ▷ *A maximal M-alternating tree is returned*

procedure Augment(M, P)
  return $M \triangle P$

procedure ExtendTree$(T = ((A, B), E_T), e = (u, v))$
  if $\exists e' = (v, w) \in M$ **then**
    $A \leftarrow A \cup \{v\}$
    $B \leftarrow B \cup \{w\}$
    $E_T \leftarrow E_T \cup \{e, e'\}$

We first note the following.

Lemma 3.3. *Let G be a bipartite graph. Consider a tree T returned by* MaximalTree(G)*. T is maximal.*

Proof. It suffices to show that there exists no edge between any two nodes $b_1, b_2 \in B$. By construction of T, if such an edge were to exist, it would form an odd length cycle as any two nodes in B always have an even length path in T. However, since G is bipartite, it is impossible for it to have an odd length cycle. Therefore, for all $b \in B$, $N(b) \subseteq A$, and T is maximal and frustrated. ■

As an immediate consequence of Lemma 3.2 and Lemma 3.3, we get the following.

Corollary 3.1. *If in the first iteration of Algorithm 2 a tree T is returned by* MaximalTree(G), *then G has no perfect matching.*

The correctness of Algorithm 2 is based on the following.

Theorem 3.3. *Let G be a graph, M a matching on G, and X the set of M-exposed nodes. Let $\mathcal{T}$ be a maximal forest of (disjoint) maximal M-alternating trees. Define $A = \bigcup_{T_i \in \mathcal{T}} A_{T_i}$ and $B = \bigcup_{T_i \in \mathcal{T}} B_{T_i}$. If*

- $X = \{\text{root}(T_i) : T_i \in \mathcal{T}\}$
- *There are no edges between nodes in B*

then M is a maximum cardinality matching.

Proof. By assumption, there are no edges between nodes in B. Therefore, all edges with an endpoint in B must have the other endpoint in A.

By definition of an M-alternating tree T_i, we have that $|B_{T_i}| - |A_{T_i}| = 1$ with the root node being M-exposed. Thus, across all trees we have that $|B| - |A| = |\mathcal{T}|$ (i.e. there are $|T|$ M-exposed nodes). For a matching to be of greater cardinality than M, it must have fewer than $|\mathcal{T}|$ exposed nodes. However, such a matching still needs to match each node of B onto a node of A, so it is not possible for it to have fewer exposed nodes. Therefore, M is a maximum cardinality matching. ■

As the above theorem applies to general graphs, we get the following for bipartite graphs.

Theorem 3.4. *If G is bipartite, the matching M that Algorithm 2 returns is a maximum cardinality matching.*

Proof. As shown in Lemma 3.3, the algorithm must return a forest $\mathcal{T}$ of maximal M-alternating trees such that for each tree T_i, there exists no edge between nodes in B_{T_i}. Additionally, there do not exist any edges between nodes of B_{T_i} and B_{T_j} for $i \neq j$ as when the first of the two trees were being constructed, such an edge would have been used to extend the tree. Moreover, by design of the algorithm, it must be that $X = \{\text{root}(T_i) : T_i \in \mathcal{T}\}$. Therefore, by Theorem 3.3 M is a maximum cardinality matching. ■

3.4 Minimum Weight Perfect Matchings in the Bipartite Case

If edges are given weights, we can look at the question of finding a perfect matching of minimum weight.

Problem 3.2 (Minimum Weight Perfect Matching)

Given a graph $G = (V, E)$ that can have a perfect matching and an edge weight function $w\colon E \to \mathbb{R}$, find a perfect matching M of minimum weight.

This can be formulated as an IP as shown below where the x_e are binary variables denoting whether an edge is in the matching or not.

$$\begin{aligned} \text{minimize} \quad & \sum_{e \in E} w_e x_e \\ \text{subject to} \quad & \sum_{e \in E\colon v \in e} x_e = 1 \quad && \forall v \in V \\ & x_e \in \{0,1\} && \forall e \in E \end{aligned} \tag{4}$$

To facilitate this in a bipartite graph $G = (V = (A, B), E)$, we can reformulate Eq. (4) to emphasize the properties of the bipartite graph. We now have binary variables x_{ij} where $i \in A$ and $j \in B$ to denote whether the edge (i, j) is in the matching or not.

$$\begin{aligned}
\text{minimize} \quad & \sum_{i \in A, j \in B} w_{ij} x_{ij} \\
\text{subject to} \quad & \sum_{j \in B} x_{ij} = 1 \quad \forall i \in A \\
& \sum_{i \in A} x_{ij} = 1 \quad \forall j \in B \\
& x_{ij} \in \{0, 1\} \quad \forall i \in A, j \in B
\end{aligned} \tag{5}$$

The LP relaxation of this IP is given by letting $x_{ij} \geq 0$.

$$\begin{aligned}
\text{minimize} \quad & \sum_{i \in A, j \in B} w_{ij} x_{ij} \\
\text{subject to} \quad & \sum_{j \in B} x_{ij} = 1 \quad \forall i \in A \\
& \sum_{i \in A} x_{ij} = 1 \quad \forall j \in B \\
& x_{ij} \geq 0 \qquad \forall i \in A, j \in B
\end{aligned} \tag{6}$$

The dual is given by

$$\begin{aligned}
\text{maximize} \quad & \sum_{i \in A} u_i + \sum_{j \in B} v_j \\
\text{subject to} \quad & u_i + v_j \leq w_{ij} \quad \forall i \in A, j \in B
\end{aligned} \tag{7}$$

where u_i denotes nodes in the A partition and v_j nodes in the B partition.

A theorem by Birkhoff shows that the optimal value of Eq. (6) is equivalent to the optimal value of Eq. (5). Therefore, by Theorem 1.2 we have that an optimal feasible solution to the dual Eq. (7) gives an optimal feasible solution to Eq. (5).

> **Theorem 3.5** (Birkhoff)**.** *Let G be a bipartite graph with edge weight function $w \colon E \to \mathbb{R}$. Then G has a perfect matching iff Eq. (6) has a feasible solution. Moreover, if G has a perfect matching, then the minimum weight of a perfect matching is equal to the optimal value of Eq. (6).*

Define $\overline{w}_{ij} = w_{ij} - (u_i + v_j)$. A solution to Eq. (7) is feasible iff $w_{ij} \geq 0$ for all $i \in A, j \in B$. By Corollary 1.1 we have that a complementary slackness condition for optimal solutions to Eq. (6) and Eq. (7) is

$$x_{ij} > 0 \Longleftrightarrow \overline{w}_{ij} = 0. \tag{8}$$

If (u, v) are solutions to Eq. (7), denote by $E_=(u, v)$ the set $\{(i, j) \in E \colon \overline{w}_{ij} = 0\}$. By the complementary slackness condition, this implies that if M is an optimal perfect matching described by vector x, then $M \subseteq E_=(u, v)$ for an optimal dual solution (u, v). However, for a given dual solution (u, v), we may not be able to find a perfect matching among the edges of $E_=(u, v)$.

Given some feasible dual solution (u, v), we can use the MaximalTree function in Algorithm 2 to search for a perfect matching in $G_= = (V, E_=)$. Otherwise, the algorithm returns a matching M of $G_=$ and an M-alternating tree T such that nodes of B_T are joined to nodes in A_T by edges of $E_=$.

In this case, we want to update the dual solution in such a way that its value increases. By definition of an M-alternating tree T in a bipartite graph, we know that A_T and B_T belong to separate partitions. WLOG assume that $A_T \subseteq A$ and $B_T \subseteq B$. We can update the dual by keeping the edges of of M and T in $E_=(u, v)$ and decreasing the $\overline{w}_{cd}$ for edges (c, d) such that $c \in B_T$ and $d \notin A_T$. The decrease can be achieved by choosing some $\varepsilon = \min\{\overline{w}_{cd} \colon c \in B_T, d \notin T\}$ and decreasing u_i by ε for all $i \in A_T$ and increasing v_j by ε for all $j \in B_T$. The dual is clearly still feasible as $\overline{w}_{ij} \geq 0$ for $i \in A, j \in B$ and some edge joining a node $c \in B_T$ and $d \notin T$ will enter $E_=$. Since this new node $d \notin T$, it can lead to either an augmentation or tree extension step. We can iteratively continue doing this until a perfect matching is returned or stop when we get a frustrated tree.

Algorithm 3 Minimum Weight Perfect Matching in Bipartite Graphs

Input: A bipartite graph $G = (V, E)$, a feasible dual solution (u, v), a matching M of $G_=$
Output: Either a perfect matching M or a NO certificate

```
Let X be the set of M-exposed nodes. Choose r ∈ X
T ← ((A, B), E_T) where A = ∅, B = {r}, E_T = ∅
while T is not frustrated do
    while ∃e = (c, d) ∈ E_= where c ∈ B, d ∉ T do
        if d ∈ X then
            Let P be the path from r to d
            M ← Augment(M, P)
            X ← X \ {r, d}
            if X = ∅ then
                return M                                  ▷ M is a perfect matching
            else
                Choose r ∈ X
                T ← ((∅, {r}), ∅)
        else
            T ← ExtendTree(T, e)
    ε ← min {w̄_cd : c ∈ B_T, d ∉ T}
    u_i ← u_i − ε for all i ∈ A, v_j ← v_j + ε for all j ∈ B
return NO                                                 ▷ No perfect matching exists

procedure Augment(M, P)
    return M △ P

procedure ExtendTree(T = ((A, B), E_T), e = (u, v))
    if ∃e' = (v, w) ∈ M then
        A ← A ∪ {v}
        B ← B ∪ {w}
        E_T ← E_T ∪ {e, e'}
```

The correctness of this algorithm is based on the fact that if a perfect matching is returned, we get a corresponding characteristic vector x and a feasible dual solution (u, v) that satisfy the complementary slackness condition Eq. (8). By Corollary 1.1, they must therefore be optimal solutions to Eq. (6) and Eq. (7), respectively. By Theorem 3.5, this gives that the matching M returned is of minimum weight.

Theorem 3.6. *Algorithm 3 can be implemented in $O(n^2 m)$ time where $n = |V|, m = |E|$.*

Proof. In the worst case, the algorithm may perform only one tree extension in an iteration of the outer while loop, and thus requires a dual change during this iteration. This gives $O(n^2)$ dual changes. A naive way to compute ε for each dual change is to just look at every edge, so in total we get a running time of $O(n^2 m)$. ∎

References

[1] V. V. Vazirani. *Approximation Algorithms*. Springer, 2001.

[2] D. P. Williamson and D. B. Shmoys. *The Design of Approximation Algorithms*. Cambridge University Press, 2011.