# Minimum-Weight Perfect Matching and Maximum-Weight Matching in Bipartite Graphs, and The Primal-Dual Schema

CSCI 4113/6101
Instructor: Norbert Zeh
October 27, 2025

In this topic, we discuss the Hungarian algorithm for finding a minimum-weight perfect matching in a bipartite graph. We will use this algorithm as an example to introduce the primal-dual schema. This is a general technique based on complementary slackness to obtain optimal solutions to a range of optimization problems. Using a relaxed version of complementary slackness, it also gives one of the most elegant techniques for obtaining efficient approximation algorithms for a range of problems. With the Hungarian algorithm in hand, we discuss a general reduction of the maximum-weight matching problem to the minimum-weight perfect matching problem. Thus, the Hungarian algorithm can also be used to find a maximum-weight matching in a bipartite graph. While the reduction from maximum-weight matching to minimum-weight perfect matching works also for non-bipartite graphs, the Hungarian algorithm is specific to bipartite graphs. It as possible to find a minimum-weight perfect matching in an arbitrary graph in the same running time as achieved by the Hungarian algorithm, but this requires ideas beyond the scope of this course. The second, optional half of these notes discusses how to obtain faster implementations of the Hungarian algorithm, which then also produces equally improved algorithms for the maximum-weight matching problem.

## 1 The Primal-Dual Schema

Remember complementary slackness. We proved that if we have a feasible solution of an LP and a feasible solution of its dual, then both solutions are optimal solutions of their respective LPs if and only they satisfy the complementary slackness conditions. This motivates the following beautiful idea for solving optimization problems that can be formulated as LPs: The algorithm starts with a usually infeasible primal solution and with a feasible dual solution that satisfy complementary slackness. It then iteratively updates the primal and dual solutions to

- Move the primal solution closer to feasibility,
- Keep the dual solution feasible, and
- Maintain complementary slackness.

At some point, the algorithm obtains a feasible primal solution. At this point, since the dual solution is feasible at all times and the primal and dual solutions satisfy complementary slackness at all times,

the primal solution is an optimal solution (as is the dual solution, but we usually care about the dual solution only as a helper in the search for an optimal primal solution).

You will sometimes hear the primal-dual schema described as maintaining an optimal but, until the algorithm terminates, infeasible primal solution and a feasible dual solution. I tend to describe it like this in class as well, because the combo of optimal primal solution and feasible dual solution is easy to remember. However, formally, it doesn't make much sense to talk about an optimal infeasible primal solution: if the solution is infeasible, that is, it violates the constraints of the LP, what prevents us from choosing the values assigned to the variables in the LP completely arbitrarily to achieve an arbitrarily good objective function value? The term "optimal" in this description should be understood to mean that the primal solution satisfies some condition with respect to the dual solution that guarantees that the primal solution is optimal if it is feasible.

Complementary slackness is one such condition, and many applications of the primal-dual schema employ optimality conditions directly derived from complementary slackness. The Hungarian algorithm does exactly this. However, technically, any condition will do that ensures that once the primal solution is feasible, it is an optimal solution. In this sense, the push-relabel algorithm for computing a maximum flow can also be viewed as an application of the primal-dual schema. We proved that a feasible flow $f$ is a maximum flow if there does not exist any $st$-path in the residual network $G^f$. We also proved that maintaining a valid height function ensures that there never exists an $st$-path in the residual network of the current preflow. Thus, the push-relabel algorithm can be seen as maintaining an "optimal" preflow $f$ in the sense that there does not exist an $st$-path in $G^f$. It then updates the flow to move it closer to feasibility and uses the height function to guide these updates to ensure that there never exists an $st$-path in $G^f$. Thus, once $f$ becomes a feasible flow, it is a maximum flow.[1]

## 2 Minimum-Weight Perfect Matching as an ILP

If $G = (U, W, E)$ is a bipartite graph, then it has a perfect matching only if $|U| = |W|$ because every vertex in $U$ must have its mate in $W$ and vice versa. We use $n = |U| = |W|$ throughout this topic. This is a necessary condition but not a sufficient one. Consider, for example, the bipartite graph $G = (U, W, \emptyset)$ with $|U| = |W|$. This graph has no perfect matching because there are no edges in $G$ at all. Thus, to keep things simple, we assume for now that the input is a *complete* bipartite graph $G = (U, W, E)$ with $|U| = |W|$. This guarantees that there always exists a perfect matching, $n!$ perfect matchings to be precise. The problem is not to decide whether there exists a perfect matching but to find one of minimum weight with respect to a weight function $w : E \to \mathbb{R}$ given as part of the input.

The Hungarian algorithm has a simple, purely combinatorial proof of correctness. The proof we discuss here is quite a bit more complicated but has the advantage that it makes it perfectly clear that the Hungarian algorithm is a direct application of the primal-dual schema. The combinatorial proof does not

[1] Remember that we proved that the dual of the LP formulation of the maximum flow problem is an LP formulation of the minimum cut problem, which gives us the Max-Flow Min-Cut Theorem as a direct consequence of strong LP duality. I tried to prove that the height function itself can be construed as a strange representation of the dual of the maximum flow LP itself. In essence, the height function needs to be interpreted as (a scaled version of) a feasible solution of the minimum cut LP, or at a minimum, it should be true that the cut defined by all vertices reachable from $s$ in $G^f$ should be a cut that satisfies complementary slackness with respect to the current preflow $f$. I didn't get this to work. Thus, the push-relabel algorithm should be viewed as employing the *spirit* of the primal-dual schema but probably not as a literal example of the primal-dual schema.

shed any light on why the vertex potentials the algorithm maintains constitute a feasible solution of the dual of the minimum-weight perfect matching problem and why the current matching and these vertex potentials satisfy complementary slackness at all times. Indeed, when I first learned about the Hungarian algorithm, its use of vertex potentials to guide the search for a perfect matching of minimum weight seemed to be magical and appear out of thin air. Deriving the algorithm directly from complementary slackness makes it much clearer how one might arrive at this algorithm without invoking magic.

To derive the Hungarian algorithm from complementary slackness, we need an ILP formulation of the minimum-weight perfect matching problem as a starting point. We associate a variable $x_e$ with every edge. An assignment $\hat{x}$ of values in $\{0,1\}$ to these variables naturally represents a subset of edges $M = \{e \in E \mid \hat{x}_e = 1\}$. We want to choose $M$ to be a minimum-weight perfect matching. Thus, we want to minimize the objective function

$$\sum_{e \in E} w_e x_e,$$

subject to appropriate constraints that guarantee that $M$ is a perfect matching. If $n = |U| = |W|$, then a perfect matching has size $n$:

$$\sum_{e \in E} x_e = n.$$

$M$ is a matching if it contains at most one of the edges incident to each vertex:

$$\begin{aligned} \sum_{v \in W} x_{u,v} &\leq 1 \qquad \forall u \in U, \\ \sum_{u \in U} x_{u,v} &\leq 1 \qquad \forall v \in W. \end{aligned}$$

This gives the following ILP formulation of the minimum-weight perfect matching problem:

$$\begin{aligned} \text{Minimize } & \sum_{e \in E} w_e x_e \\ \text{s.t. } & \sum_{e \in E} x_e \geq n \\ & \sum_{v \in W} x_{u,v} \leq 1 \qquad \forall u \in U \\ & \sum_{u \in U} x_{u,v} \leq 1 \qquad \forall v \in W \\ & x_e \in \{0,1\} \quad \forall e \in E. \end{aligned} \tag{1}$$

We turned the equality constraint $\sum_{e \in E} x_e = n$ into an inequality $\sum_{e \in E} x_e \geq n$ because the upper bound $\sum_{e \in E} x_e \leq n$ is redundant: If $\sum_{e \in E} x_e \geq n$ and $\sum_{w \in W} x_{(u,w)} \leq 1$, for all $u \in U$, then we must have $\sum_{e \in E} x_e = n$, $\sum_{w \in W} x_{(u,w)} = 1$, for all $u \in U$, and $\sum_{u \in U} x_{(u,w)} = 1$, for all $w \in W$.

The LP relaxation of (1) is

$$\begin{aligned} \text{Minimize } & \sum_{e \in E} q_e x_e \\ \text{s.t. } & \sum_{e \in E} x_e \geq n \\ & \sum_{v \in W} x_{u,v} \leq 1 \quad \forall u \in U \\ & \sum_{u \in U} x_{u,v} \leq 1 \quad \forall v \in W \\ & x_e \geq 0 \quad \forall e \in E. \end{aligned} \tag{2}$$

Note that we omitted the upper bound constraint $x_e \leq 1$, for every edge $e \in E$. This constraint is redundant because, for all $\{u, v\} \in E$, $x_{u,v} \leq 1$ follows from the constraints $\sum_{v' \in W} x_{u,v'} \leq 1$ and $x_{u,v'} \geq 0$, for all $\{u, v'\} \in E$.

The Hungarian Algorithm finds an *integral* optimal solution of (2). Thus, it is also an optimal solution of (1) and is therefore a minimum-weight perfect matching.

The dual of (2) is

$$\begin{aligned} \text{Maximize } & nz - \sum_{u \in U} y_u - \sum_{v \in W} y_v \\ \text{s.t. } & z - y_u - y_v \leq w_{u,v} \quad \forall \{u, v\} \in E \\ & y_u \geq 0 \qquad \forall u \in U \\ & y_v \geq 0 \qquad \forall v \in W \\ & z \geq 0. \end{aligned} \tag{3}$$

Here, $z$ is the dual variable corresponding to the first primal constraint, $y_u$ is the dual variable corresponding to the constraint $\sum_{v \in W} x_{u,v} \leq 1$ associated with the vertex $u \in U$, and $y_v$ is the dual variable corresponding to the constraint $\sum_{u \in U} x_{u,v} \leq 1$ associated with the vertex $v \in W$.

The goal of the Hungarian algorithm is to find an integral feasible primal solution $\hat{x}$ of (2) and a feasible dual solution $(\hat{z}, \hat{y})$ of (3) that satisfy complementary slackness. The complementary slackness conditions for the two LPs (2) and (3) are

$$\sum_{e \in E} x_e = n \qquad \text{or} \qquad z = 0 \tag{4}$$

$$\sum_{v \in W} x_{u,v} = 1 \qquad \text{or} \qquad y_u = 0 \qquad \forall u \in U \tag{5}$$

$$\sum_{u \in U} x_{u,v} = 1 \qquad \text{or} \qquad y_v = 0 \qquad \forall v \in W \tag{6}$$

$$z - y_u - y_v = w_{u,v} \qquad \text{or} \qquad x_{u,v} = 0 \qquad \forall \{u, v\} \in E. \tag{7}$$

As observed above, conditions (4) to (6) are satisfied by every feasible primal solution, so we can ignore them. This leaves us with (7).

Let us clean up the dual a little. Let us associate a new variable $\pi_u = z - y_u$ with every vertex $u \in U$

and a new variable $\pi_w = -y_w$ with every vertex $w \in W$. Substituting these variables into (3) gives the LP

$$\begin{aligned} \text{Maximize } & \sum_{u \in U} \pi_u + \sum_{v \in W} \pi_v \\ \text{s.t. } & \pi_u + \pi_v \leq w_{u,v} \quad \forall \{u, v\} \in E \\ & \pi_v \leq 0 \qquad \forall v \in W. \end{aligned} \tag{8}$$

With the same substitution, the complementary slackness condition (7) becomes

$$\pi_u + \pi_v = w_{u,v} \quad \text{or} \quad x_{u,v} = 0 \quad \forall \{u, v\} \in E.$$

We can consider $\pi$ to be a function assigning a potential $\pi_v$ to every vertex of $G$. Since we have the constraint $\pi_u + \pi_v \leq w_{u,v}$, for every edge $\{u, v\} \in E$, it is natural to call an edge $\{u, v\}$ **tight** if $\pi_u + \pi_v = w_{u,v}$. Since our primal LP encodes that the subset of edges $M \subseteq E$ we want to find is a perfect matching, we can interpret the complementary slackness condition as: Find a perfect matching $M \subseteq E$ and a potential function $\pi$ such that $M$ contains only tight edges.

**Lemma 1.** *A perfect matching $M$ in a bipartite graph $G = (U, W, E)$ is a minimum-weight perfect matching if and only if there exists a feasible potential function $\pi$ such that every edge in $M$ is tight with respect to $\pi$.*

## 3 The Hungarian Algorithm

Based on Lem. 1, the Hungarian algorithm maintains a matching $M$ and a feasible potential function $\pi$ such that every edge in $M$ is tight with respect to $\pi$. Then it iteratively updates the matching and the potential function. When updating the matching, the matching becomes bigger while ensuring that the matching contains only tight edges. Whenever we cannot grow the matching without adding a non-tight edge to it, we update the potential function to create more tight edges. We will show that after at most $n$ iterations of updating the potential function, we can grow the matching by one edge. Thus, after at most $n(n+1)$ iterations, we obtain a matching of size $n$, which is a perfect matching. Since all edges in $M$ are tight, Lem. 1 shows that $M$ is a minimum-weight perfect matching at this time. It remains to fill in the details of the Hungarian algorithm. Fig. 1 shows an example of running this algorithm.

Initially, the algorithm sets

$$\begin{aligned} & M = \emptyset, \\ & \pi_v = 0 && \forall v \in W, \text{ and} \\ & \pi_u = \min\{w_{u,v} \mid v \in W\} && \forall u \in U. \end{aligned}$$

This makes $M$ an infeasible primal solution because it is not a perfect matching yet. The potential function $\pi$ is a feasible dual solution because $\pi_v \leq 0$, for all $v \in W$, and, for every edge $\{u, v\}$,

$$\pi_u + \pi_v = \min\{w_{u,v'} \mid v' \in W\} + 0 \leq w_{u,v}.$$

Since $M = \emptyset$, $M$ and $\pi$ trivially satisfy the invariant that every edge in $M$ is tight with respect to $\pi$.

Each iteration of the algorithm constructs the subgraph $H = (V, E') \subseteq G$, where $E'$ is the set of all tight edges of $G$ with respect to the current potential function $\pi$. Since $G$ is bipartite, so is $H$. Thus,

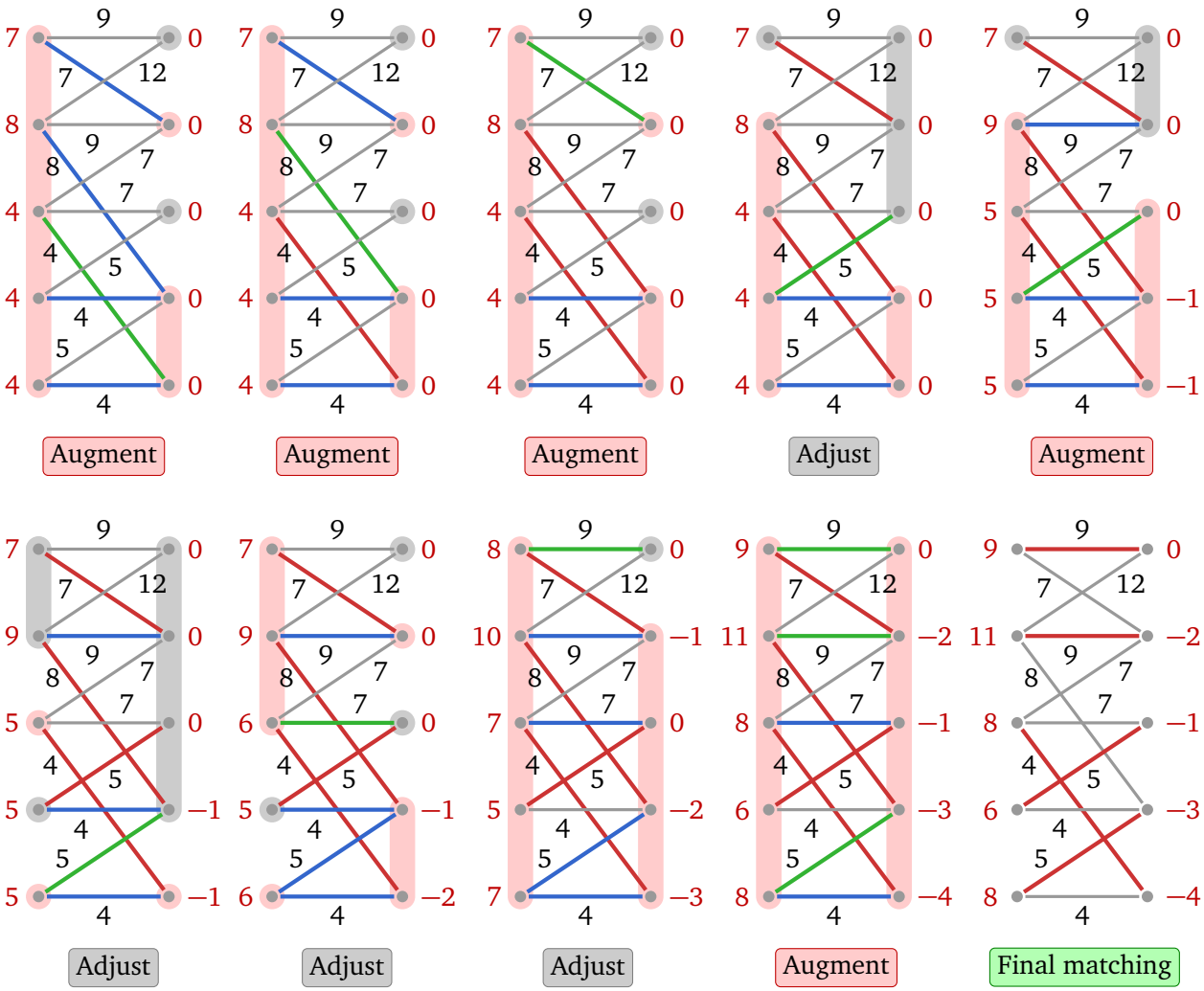


Figure 1: Illustration of the Hungarian Algorithm. Black edge labels are edge weights. Red vertex labels are vertex potentials. In each iteration, the sets $X$ and $Y$ are shaded red; the sets $\bar{X}$ and $\bar{Y}$ are shaded grey. Matched edges are red. Unmatched tight edges are blue. Non-tight edges are grey. Each iteration is labelled "Adjust" or "Augment". Each "Adjust" iteration adjusts the potential function by increasing the potentials of the vertices in $X$ and decreasing the potentials of the vertices in $Y$ by the slack of the tightest edge between $X$ and $\bar{Y}$, which is highlighted in green. Each "Augment" iteration augments the matching using an augmenting path of tight edges. The unmatched edges on the path are highlighted in green.

we can use alternating BFS in $H$ from the unmatched vertices in $U$ to try to find an augmenting path for $M$ in $H$. If we find such a path $P$, then $M \oplus P$ is a matching of size $|M \oplus P| = |M| + 1$. Moreover, $M \oplus P = (M \setminus P) \cup (P \setminus M) \subseteq M \cup P$. Every edge in $M$ is tight with respect to $\pi$. $P$ is a path in $H$, so every edge in $P$ is also tight with respect to $\pi$. Thus, every edge in $M \oplus P$ is tight with respect to $\pi$. Therefore, replacing $M$ with $M \oplus P$ grows $M$ by one edge and maintains the invariant that every edge in $M$ is tight with respect to $\pi$.

If we do not find an augmenting path for $M$ in $H$ but $M$ is not a perfect matching yet, then we update $\pi$ to create more tight edges. Let $F$ be the alternating forest found by the application of alternating BFS, let $X$ be the set of vertices in $U$ that belong to $F$, let $Y$ be the set of vertices in $W$ that belong to $F$, let $\bar{X} = U \setminus X$, let $\bar{Y} = W \setminus Y$, and let

$$\delta = \min \left\{ w_{u,v} - \pi_u - \pi_v \,\middle|\, u \in X, v \in \bar{Y} \right\}.$$

Then we replace $\pi$ with the potential function $\pi'$ defined as

$$\pi'_v = \begin{cases} \pi_v + \delta & \text{if } v \in X \\ \pi_v - \delta & \text{if } v \in Y \\ \pi_v & \text{otherwise.} \end{cases}$$

The next two lemmas prove that $\pi'$ is a feasible dual solution and that all edges in $M$ remain tight with respect to $\pi'$.

**Lemma 2.** *$\pi'$ is a feasible solution of* (8).

*Proof.* Since $\pi$ is a feasible solution of (8), every edge $\{u, v\}$ in $G$ satisfies $w_{u,v} - \pi_u - \pi_v \geq 0$. Thus, $\delta \geq 0$. Therefore, every vertex $v \in W$ satisfies $\pi'_v \leq \pi_v \leq 0$. Thus, $\pi'$ satisfies the non-positivity constraints of all variables $\pi_v$, $v \in W$.

For every edge $\{u, v\} \in G$, we have $\pi'_u + \pi'_v \leq \pi_u + \pi_v \leq w_{u,v}$ unless $u \in X$ and $v \in \bar{Y}$. If $u \in X$ and $v \in \bar{Y}$, then

$$\pi'_u + \pi'_v = \pi_u + \delta + \pi_v = \pi_u + \pi_v + \min \left\{ w_{u',v'} - \pi_{u'} - \pi_{v'} \,\middle|\, u' \in X, v' \in \bar{Y} \right\} \leq \pi_u + \pi_v + w_{u,v} - \pi_u - \pi_v = w_{u,v}.$$

Thus, $\pi'$ satisfies the constraints $\pi_u + \pi_v \leq w_{u,v}$, for all edges of $G$. This makes $\pi'$ a feasible solution of (8). □

**Lemma 3.** *All edges in $M$ are tight with respect to $\pi'$.*

*Proof.* Consider any edge $\{u, v\} \in M$. This edge is tight with respect to $\pi$. Thus, if $u \notin \bar{X}$ or $v \notin Y$, then

$$\pi'_u + \pi'_v \geq \pi_u + \pi_v = w_{u,v}$$

because, as observed in the proof of Lem. 2, $\delta \geq 0$. Since $\pi'$ is a feasible solution of (8), by Lem. 2, we also have

$$\pi'_u + \pi'_v \leq w_{u,v}$$

Thus,

$$\pi'_u + \pi'_v = w_{u,v},$$

that is, the edge is tight with respect to $\pi'$.

To finish the proof of the lemma, we prove that there is no edge $\{u, v\} \in M$ with $u \in \bar{X}$ and $v \in Y$. Assume for the sake of contradiction that there exists such an edge. Since $v \in Y$, $P_v$ is an alternating path from $r_v$ to $v$. Since $r_v \in U$, $v \in W$, $r_v$ is unmatched, and $G$ is bipartite, the last edge in $P_v$ is not in $M$. Since $\{u, v\} \in M$, this makes $P = P_v \circ \langle u \rangle$ an alternating path from $r_v$ to $u$. Moreover, this is an alternating path from $r_v$ to $u$ in $H$ because $P_v$, being a path in $F$, is a path in $H$, and $\{u, v\}$, being in $M$, is tight with respect to $\pi$ and, therefore, also in $H$. As we proved in the previous topic, alternating BFS from the unmatched vertices in $U$ discovers all vertices reachable from unmatched vertices in $U$ via alternating paths. Thus, $u \in X$, which is the desired contradiction. □

If we split the iterations of the algorithm into two types, those that grow the matching and those that adjust the potential function, then it it clear that the algorithm terminates after at most $n$ iterations that grow the matching, because we obtain a perfect matching after the $n$th such iteration. To bound the running time of the algorithm, we need to bound the number of iterations that adjust the potential function. We prove that between any two consecutive iterations that grow the matching, there are at most $n$ iterations that adjust the potential function. The key do doing so is the following lemma:

**Lemma 4.** *Let $\pi$ and $\pi'$ be the potential functions before and after an iteration that updates the vertex potentials, let $H$ and $H'$ by the tight subgraphs of $G$ with respect to $\pi$ and $\pi'$, respectively, and let $Y$ and $Y'$ be the vertices in $W$ reachable from unmatched vertices in $U$ via alternating paths in $H$ and $H'$, respectively. Then $Y' \supset Y$.*

*Proof.* Consider some vertex $v$ reachable from an unmatched vertex $u \in U$ via an alternating path $P = \langle u = v_0, \ldots, v_k = v \rangle$. Then each vertex $v_i \in P$ is reachable from $u$ via the alternating path $\langle v_0, \ldots, v_i \rangle$. Therefore, $v_0, \ldots, v_k \subseteq X \cup Y$. More specifically, since $G$ is bipartite and $u = v_0 \in U$, we have $v_i \in X$ if $i$ is even, and $v_i \in Y$ if $i$ is odd.

Since $P$ is a path in $H$, all its edges are tight with respect to $\pi$. Since $\pi'_z = \pi_z + \delta$ if $z \in X$, and $\pi'_z = \pi_z - \delta$ if $z \in Y$, all edges in $P$ are also tight with respect to $\pi'$. This shows that $P$ is also an alternating path from $u$ to $v$ in $H'$. Since this is true for every vertex $v \in X \cup Y$, this shows that $X \subseteq X'$ and $Y \subseteq Y'$.

Now consider the vertices $u \in X$ and $v \in \bar{Y}$ such that $\delta = w_{u,v} - \pi_u - \pi_v$. By the definition of $\delta$, such a pair of vertices exists. Since $\pi'_v = \pi_v$ and $\pi'_u = \pi_u + \delta$, we have $\pi'_u + \pi'_v = w_{u,v}$, that is, the edge $\{u, v\}$ is tight with respect to $\pi'$. Moreover, all edges in $M$ are tight with respect to $\pi$, therefore belong to $H$, and alternating BFS either adds both endpoints of an edge in $M$ to $F$ or neither. Thus, every edge in $M$ either has its endpoints in $X$ and $Y$ or in $\bar{X}$ and $\bar{Y}$. Since $u \in X$ and $v \in \bar{Y}$, this shows that $\{u, v\} \notin M$.

Since $u \in X$, we have $u \in F$. The alternating path $P_u$ from $r_u$ to $u$ in $F$ is an alternating path from $r_u$ to $u$ in $H$. As just observed, it as also an alternating path from $r_u$ to $u$ in $H'$. Since $r_u$ is unmatched, $P_u$ starts with an unmatched edge. Since $r_u, u \in U$, $P_u$ has even length. Thus, the last edge in $P_u$ is in $M$. Finally observe that all vertices in $P_u$ belong to $X \cup Y$. Therefore, $v \notin P_u$. Since $\{u, v\} \notin M$, this makes $P_u \circ \langle v \rangle$ an alternating path from $r_u$ to $v$ in $H'$, that is, $v \in Y'$. Since $v \notin Y$ and we have already shown that $Y \subseteq Y'$, this shows that $Y \subset Y'$. □

By Lem. 4, every time we update $\pi$, the set $Y$ grows by at least one vertex. Thus, we can update $\pi$ at

most $n$ times before $Y = W$. On the other hand, as long as $M$ is not a perfect matching, there exists an unmatched vertex in $W$. Thus, after updating $\pi$ at most $n$ times, $Y$ contains an unmatched vertex in $W$. However, $Y$ is the set of vertices in $W$ that belong to the alternating forest $F$ in each iteration. Therefore, when this happens, alternating BFS finds an alternating path $P_v$ from an unmatched vertex $r_v \in U$ to an unmatched vertex $v \in W$, and the current iteration grows the matching by one edge. This immediately implies that the algorithm finds a perfect matching after at most $(n+1)n$ iterations. We are ready to prove our first main result in this topic:

**THEOREM 5.** *A minimum-weight perfect matching in a bipartite graph can be found in $O\left(n^4\right)$ time.*

*Proof.* Lems. 2 and 3 prove that $\pi$ is always a feasible potential function and that all edges in $M$ are tight with respect to $\pi$. Thus, by Lem. 1, once the algorithm obtains a perfect matching and returns it, this matching is a minimum-weight perfect matching.

We have just argued that the algorithm finds such a matching after at most $(n+1)n$ iterations. Thus, to argue that the algorithm runs in $O\left(n^4\right)$ time, we need to argue that every iteration takes $O\left(n^2\right)$ time. Note that each iteration needs to construct $H$, run alternating BFS in $H$, test whether $F$ contains an unmatched vertex in $v \in W$, and either compute $M \oplus P_v$, if such a vertex exists, or update $\pi$, if it doesn't. To update $\pi$, it needs to inspect all edges between vertices in $X$ and $\bar{Y}$ to determine $\delta$, and then it needs to update the potentials of all vertices in $X$ and $Y$. We observed that alternating BFS takes linear time in the size of the graph to which we apply it. All the other steps clearly also take linear time in the number of vertices or edges of $G$. Since $G$ is a complete bipartite graph with $2n$ vertices, it has $n^2$ edges. Thus, each iteration does indeed take $O\left(n^2\right)$ time, and the whole algorithm runs in $O\left(n^4\right)$ time. □

## 4 MAXIMUM-WEIGHT MATCHING

Having shown that we can find a minimum-weight perfect matching in a bipartite graph in $O\left(n^4\right)$ time, we are ready to prove that a maximum-weight matching can be found in the same amount of time. More generally, we prove that

**PROPOSITION 6.** *If there exists an algorithm that computes a minimum-weight perfect matching in any arbitrary graph in $T(n,m)$ time, then there exists an algorithm that computes a maximum-weight matching in any arbitrary graph in $O(n+m+T(n,m))$ time.*

*Proof.* First observe that computing a maximum-weight matching in a graph $G$ is the same as computing a minimum-weight matching in $G$ after negating all edge weights. To compute a minimum-weight matching in $G$, we construct a graph $G''$ composed of $G$ and an identical copy $G'$ of $G$. In addition, for every vertex $u \in G$, $G''$ has an edge $(u,u')$ of weight zero, where $u'$ is the copy of $u$ in $G'$. This is illustrated in Fig. 2. $G''$ has a trivial perfect matching $\{(u,u') \mid u \in G\}$. Since $G''$ has $2n$ vertices and $2m+n = O(m)$ edges, we can compute a minimum-weight perfect matching $M''$ of $G''$ in $O(T(n,m))$ time. As we show next, the matching $M = \{(u,v) \in M'' \mid u,v \in G\}$ is a minimum-weight matching of $G$. Since the construction of $G''$ from $G$ can clearly be carried out in $O(n+m)$ time, as can the construction of $M$ from $M''$, this shows that a minimum-weight matching in $G$ can be computed in $O(n+m+T(n,m))$ time.

Let $M^*$ be a minimum-weight matching of $G$ and consider the two matchings $M$ and $M' = \{(u',v') \in$

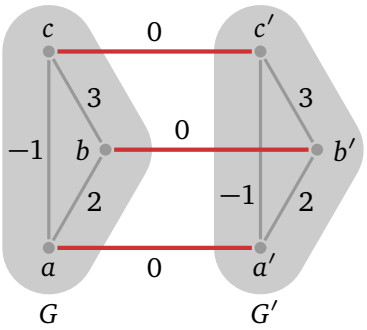


Figure 2: Construction of the graph $G''$ from two copies of $G$. A trivial perfect matching of $G''$ is shown in red.

$M'' \mid u', v' \in G'\}$. Since $M^*$ is a minimum-weight matching of $G$, $M$ is a matching of $G$, and $M'$ is a matching of $G'$, we have $q(M) \geq q(M^*)$ and $q(M') \geq q(M^*)$. If $q(M) > q(M^*)$, then observe that $q(M'') = q(M) + q(M') > 2q(M^*)$ because every edge between $G$ and $G'$ in $G''$ has weight 0. On the other hand, the matching $M''' = \{(u, v), (u', v') \mid (u, v) \in M^*\} \cup \{(u, u') \mid u \in G \text{ is unmatched by } M^*\}$ is a perfect matching of $G''$ of weight $2q(M^*)$. Thus, $M''$ is not a minimum-weight perfect matching of $G''$, a contradiction. This shows that $q(M) \leq q(M^*)$. Since $q(M) \geq q(M^*)$, we therefore have $q(M) = q(M^*)$, that is, $M$ is a minimum-weight matching of $G$. □

Note that Prop. 6 provides a reduction from maximum-weight matching to minimum-weight perfect matching for arbitrary graphs, not only for bipartite graphs, but it also requires the existence of an algorithm that can compute a minimum-weight perfect matching in an arbitrary graph. We only discussed how to find minimum-weight perfect matchings in bipartite graphs. The next proposition shows that if we want to compute a maximum-weight perfect matching in a bipartite graph, we only need an algorithm to compute a minimum-weight perfect matching in bipartite graphs.

**Proposition 7.** *If there exists an algorithm that computes a minimum-weight perfect matching in any bipartite graph in $T(n, m)$ time, then there exists an algorithm that computes a maximum-weight matching in any bipartite graph in $O(n + m + T(n, m))$ time.*

*Proof.* We use the same reduction from maximum-weight matching to minimum-weight perfect matching as in the proof of Prop. 7. If $G = (U, W, E)$ is bipartite, then so is $G' = (U', W', E')$, and it is easily verified that every edge of $G''$ has one endpoint in $U \cup W'$ and the other in $U' \cup W$. Thus, $G''$ is bipartite, and we can use a minimum-weight perfect matching algorithm for bipartite graphs to compute such a matching in $G''$. □

Since the Hungarian algorithm computes a minimum-weight perfect matching in $O\left(n^4\right)$ time, we immediately obtain the following corollary:

**Corollary 8.** *A maximum-weight matching in a bipartite graph can be computed in $O\left(n^4\right)$ time.*

# 5 Faster Minimum-Weight Perfect Matching and Maximum-Weight Matching*

Our goal in this section is to reduce the running time of the Hungarian Algorithm to $O\left(n^3\right)$. Recall that the current running time of $O\left(n^4\right)$ is the result of the algorithm executing up to $(n+1)n$ iterations, each of which costs $O\left(n^2\right)$ time. Also recall that there are $n$ iterations that grow the matching, and the remaining up to $n^2$ iterations adjust the potential function to allow the matching to grow. Our faster implementation of the Hungarian algorithm groups these iterations into phases. Each phase ends with an iteration that grows the matching. All iterations in a phase before the last iteration are iterations that adjust the potential function. Thus, the algorithm has $n$ phases. The key to obtaining a faster algorithm then is to show that we need to find an alternating forest only once per phase, instead of once per iteration. Since the $O\left(n^2\right)$ cost of computing an alternating forest is the most costly part of the algorithm, performing this step only once per phase, $n$ times in total, instead of once per iteration, reduces the running time of the algorithm to $O\left(n^3\right)$.

The reason why we can get away with building the alternating forest once per phase is stated in the last paragraph of the proof of Lem. 4. This paragraph observes that because every edge in $F$ connects a vertex in $X$ with a vertex in $Y$, each such edge remains tight. This implies that every time we update $\pi$, we do not need to compute $F$ from scratch but can simply add the vertices reachable via newly tight edges to the forest obtained before updating $\pi$. Thus, we can think of the implementation of a phase as running a single search for tight alternating paths (not necessarily in breadth-first order, which is important only if we want to find *shortest* augmenting pathe) until we find an unmatched vertex in $W$, and updating $\pi$ *on the fly* as necessary to allow us to reach such a vertex. To implement this, we need a few simple pieces of information.

Each phase explicitly maintains the sets $X$, $Y$, $\bar{X}$, and $\bar{Y}$ while running the alternating search. Initially, all unmatched vertices in $U$ are roots of the forest $F$ and no other vertices are in $F$ yet, because we have not explored any edges yet. Thus, $X$ contains all unmatched vertices in $U$, $\bar{X} = U \setminus X$, $Y = \emptyset$, and $\bar{Y} = W$.

For each vertex $v \in X \cup Y$, the algorithm stores its parent $p_v$ in $F$. If $v$ is an unmatched vertex in $U$, then it has no parent in $F$, so $p_v = \text{NULL}$. Once we add an unmatched vertex $v \in W$ to $F$, we can use these parent pointers starting from $v$ to collect the edges in $P_v$ and "flip" their membership in $M$.

For each vertex $v \in \bar{Y}$, we store two pieces of information:

$$\delta_v = \min\{w_{u,v} - \pi_u - \pi_v \mid u \in X\}$$

and the vertex $p_v \in X$ such that $w_{p_v,v} - \pi_{p_v} - \pi_v = \delta_v$.

Finally, we maintain a queue $Q$ of all those vertices $v \in \bar{Y}$ with $\delta_v = 0$. We use this queue to implement the construction of an alternating forest in the tight subgraph $H$ of $G$. As long as $Q$ is not empty, there exists a vertex $v \in Q \subseteq \bar{Y}$ such that the edge $\{p_v, v\}$ is tight. Since $p_v \in X \subseteq U$, $P_{p_v}$ is an alternating path of even length from $r_{p_v}$ to $p_v$. In particular, the last edge of $P_v$ is in $M$. Since $v \in \bar{Y}$, that is, $v \notin F$, this last edge is not the edge $\{p_v, v\}$. Since $M$ is a matching, this implies that $\{p_v, v\} \notin M$ and $P_v = P_{p_v} \circ \langle v \rangle$ is a tight alternating path from $r_v = r_{p_v}$ to $v$. Thus, we can add $v$ to $F$ by moving $v$ from $\bar{Y}$ to $Y$ and deleting $v$ from $Q$. We do not update $p_v$ because $p_v$ *is* now the parent of $v$ in $F$.

If $v$ is unmatched, then $P_v$ is an augmenting path, we replace $M$ with $M \oplus P_v$, and the phase ends. Otherwise, there exists an edge $\{u, v\} \in M$, which is tight because all edges in $M$ are tight. Thus,

$P_u = P_v \circ \langle u \rangle$ is a tight alternating path from $r_v$ to $u$. We set $p_u = v$ to make $u$ a child of $v$ in $F$ and move $u$ from $\bar{X}$ to $X$. This requires us to update the $\delta_{v'}$ values of all vertices in $\bar{Y}$ as well as the queue $Q$. For each vertex $v' \in \bar{Y}$, the edge $\{u, v'\}$ may be the new tightest edge connecting $v'$ to a vertex in $X$. This is the case if $w_{u,v'} - \pi_u - \pi_{v'} < \delta_{v'}$. In this case, we set $\delta_{v'} = w_{u,v'} - \pi_u - \pi_{v'}$ and $p_{v'} = u$. If $\delta_{v'} = 0$ now (but not before), then we add $v'$ to $Q$.

This process ends when we add an unmatched vertex $v \in W$ to $F$ or the queue $Q$ becomes empty before discovering such a vertex $v \in W$. In the latter case, we need to update the vertex potentials to create new tight edges that can be explored. We inspect all vertices in $\bar{Y}$, and calculate

$$\delta = \min\left\{\delta_v \,\middle|\, v \in \bar{Y}\right\},$$

add $\delta$ to $\pi_u$, for all $u \in X$, and subtract $\delta$ from $\pi_v$, for all $v \in Y$. Since

$$\delta_v = \min\{w_{u,v} - \pi_u - \pi_v \mid u \in X\},$$

for all $v \in \bar{Y}$, increasing $\pi_u$ by $\delta$, for all $u \in X$, decreases $\delta_v$ by $\delta$, for all $v \in \bar{Y}$. Thus, we iterate over the vertices in $\bar{Y}$ a second time, decrease each $\delta_v$ value by $\delta$, and add each vertex $v \in \bar{Y}$ that now satisfies $\delta_v = 0$ to $Q$. Note that the definition of $\delta$ ensures that there exists at least one such vertex, that is, each time we update the vertex potentials, we are able to add at least one more vertex in $\bar{Y}$ to $F$.

This algorithm implements exactly the same search for a minimum-weight perfect matching as the algorithm in the previous section, with the one difference that it does not visit vertices in breadth-first order. However, the correctness proof of the algorithm in the previous section does not rely on the augmenting paths it finds being shortest paths. Thus, the algorithm just described does find a minimum-weight perfect matching. To prove the following theorem then, we need to argue that it runs in $O\left(n^3\right)$ time.

**Theorem 9.** *A minimum-weight perfect matching or maximum-weight matching in a complete bipartite graph can be computed in $O\left(n^3\right)$ time.*

*Proof.* Since each phase grows the matching by one edge, there are $n$ phases. Thus, it suffices to prove that each phase takes $O\left(n^2\right)$ time.

Initializing $X$, $\bar{X}$, $Y$, and $\bar{Y}$ at the beginning of a phase is easily done in linear time. Next, we iterate over the vertices in $\bar{Y}$. For each $v \in \bar{Y}$, we iterate over the vertices in $X$ to compute $\delta_v$ and $p_v$. This clearly takes $O(n)$ time per vertex in $\bar{Y}$, $O\left(n^2\right)$ time for all vertices in $\bar{Y}$. The queue $Q$ can be initialized in the process by adding every vertex $v \in \bar{Y}$ with $\delta_v = 0$ to $Q$.

As long as $Q$ is non-empty, we add some vertex $v$ in $Q$ to $Y$. This takes constant time per vertex. If $v$ is matched, we also move its mate from $\bar{X}$ to $X$. This also takes constant time. In addition, we inspect all edges incident to $u$ to identify those vertices $v' \in \bar{Y}$ whose $\delta_{v'}$ and $p_{v'}$ values need to be updated. This takes constant time per edge incident to $u$, $O(n)$ time in total. In summary, every time we remove a vertex from $Q$, we add one or two vertices to $F$, and doing so costs at most $O(n)$ per vertex. Since we can add at most $2n$ vertices to $F$, the cost of adding vertices to $F$ is $O\left(n^2\right)$ per phase.

Finally, updating vertex potentials requires finding the minimum of all $\delta_v$ values, for all $v \in \bar{Y}$, and then updating $\pi_v$, for all $v \in X \cup Y$, and $\delta_v$, for all $v \in V$. This takes $O(n)$ time. We also need to add all those vertices in $\bar{Y}$ to $Q$ for which $\delta_v$ drops to 0. This costs constant time per vertex, $O(n)$ time for all vertices in $\bar{Y}$. Thus, updating vertex potentials takes linear time every time the queue $Q$ becomes

empty during a phase. Every time this happens, at least one edge between $X$ and $\bar{Y}$ becomes tight with respect to the updated vertex potentials, allowing us to move at least one vertex from $\bar{Y}$ to $Y$. Thus, after updating the vertex potentials at most $n$ times, all vertices in $W$ are in $F$, that is, we must be able to reach an unmatched vertex in $W$ via a tight alternating path. Thus, we update vertex potentials at most $n$ times per phase. The total cost of updating vertex potentials per phase is thus $O\left(n^2\right)$.

Since all parts of a phase can be implemented in $O\left(n^2\right)$ time and the algorithm executes $n$ phases, the whole algorithm runs in $O\left(n^3\right)$ time.

The Hungarian algorithm computes a minimum-weight perfect matching. By Prop. 7, we can find a maximum-weight matching in the same amount of time. □

The Hungarian Algorithm does not use the fact that the input graph is complete in any way. We even used an incomplete graph to illustrate the algorithm in Fig. 1. Completeness only guarantees that the graph has a perfect matching. Thus, we can use the Hungarian algorithm to try to find a minimum-weight perfect matching (or a maximum-weight matching via the reduction from Prop. 7) even in not necessarily complete bipartite graphs. One way to do this is to find a maximum matching first, which can be done in $O\left(m\sqrt{n}\right)$ time. If this maximum matching is not perfect, then there is no perfect matching. Otherwise, there exists a perfect matching, and we can now run the Hungarian algorithm to find a minimum-weight perfect matching in $O\left(n^3\right)$ time. Exer. 1 provides the means to avoid computing a maximum matching first, by giving a condition that can be used to test for the non-existence of a perfect matching while the Hungarian algorithm runs.

Now, if the graph is complete, then it has $n^2$ edges. Thus, the cost per phase of the Hungarian algorithm is in fact linear in the size of the graph. If the number of edges is $m \in o\left(n^2\right)$ though, then there is no obvious reason why it should take $O\left(n^2\right)$ time to find a tight augmenting path in each phase. Can this cost be reduced to $O(m)$, resulting in an $O(nm)$-time minimum-weight perfect matching algorithm? Almost. By using appropriate data structures, the cost per phase can be reduced to $O(n \lg n + m)$, which gives a minimum-weight perfect matching algorithm with running tim $O\left(n^2 \lg n + nm\right)$. This algorithm uses the same strategy as the implementation of the Hungarian algorithm disussed in this section, only it maintains vertex potentials and the slack values $\delta_v$ for the vertices in $\bar{Y}$ in a way that the minimum $\delta_v$ value can be determined in $O(\lg n)$ time and all $\delta_v$ values and vertex potentials of vertices in $X \cup Y$ can be updated in bulk in constant (!) time. The main data structure to achieve this is a priority queue storing the $\delta_v$ values plus a range-sum data structure storing the vertex potentials. A detailed discussion of this faster implementation of the Hungarian algorithm is beyond the scope of this course.

## EXERCISES

**EXERCISE 1.** Consider running the $O\left(n^3\right)$-time implementation of the Hungarian algorithm on an arbitrary bipartite graph $G$. Prove that $G$ has no perfect matching if and only if the algorithm reaches a situation where all vertices in $Y$ are matched and none of the vertices in $\bar{Y}$ has a neighbour in $X$.