MAXIMUM MATCHINGS

A matching is a set of edges, so its **size** is the number of edges. We can seek a large matching by iteratively selecting edges whose endpoints are not used by the edges already selected, until no more are available. This yields a maximal matching but maybe not a maximum matching.

3.1.4. Definition. A **maximal matching** in a graph is a matching that cannot be enlarged by adding an edge. A **maximum matching** is a matching of maximum size among all matchings in the graph.

A matching M is maximal if every edge not in M is incident to an edge already in M. Every maximum matching is a maximal matching, but the converse need not hold.

3.1.5. Example. *Maximal $\neq$ maximum.* The smallest graph having a maximal matching that is not a maximum matching is P_4. If we take the middle edge, then we can add no other, but the two end edges form a larger matching. Below we show this phenomenon in P_4 and in P_6. ■

In Example 3.1.5, replacing the bold edges by the solid edges yields a larger matching. This gives us a way to look for larger matchings.

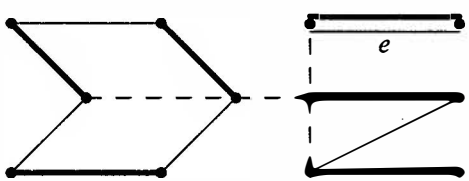
3.1.6. Definition. Given a matching M, an M-**alternating path** is a path that alternates between edges in M and edges not in M. An M-alternating path whose enpoints are unsaturated by M is an M-**augmenting path**.

Given an M-augmenting path P, we can replace the edges of M in P with the other edges of P to obtain a new matching M' with one more edge. Thus when M is a maximum matching, there is no M-augmenting path.

In fact, we prove next that maximum matchings are characterized by the absence of augmenting paths. We prove this by considering two matchings and examining the set of edges belonging to exactly one of them. We define this operation for any two graphs with the same vertex set. (The operation is defined in general for any two sets; see Appendix A.)

3.1.7. Definition. If G and H are graphs with vertex set V, then the **symmetric difference** $G \triangle H$ is the graph with vertex set V whose edges are all those edges appearing in exactly one of G and H. We also use this notation for sets of edges; in particular, if M and M' are matchings, then $M \triangle M' = (M - M') \cup (M' - M)$.

3.1.8. Example. In the graph below, M is the matching with five solid edges, M' is the one with six bold edges, and the dashed edges belong to neither M nor M'. The two matchings have one common edge e; it is not in their symmetric difference. The edges of $M \triangle M'$ form a cycle of length 6 and a path of length 3. ■



3.1.9. Lemma. Every component of the symmetric difference of two matchings is a path or an even cycle.

Proof: Let M and M' be matchings, and let $F = M \triangle M'$. Since M and M' are matchings, every vertex has at most one incident edge from each of them. Thus F has at most two edges at each vertex. Since $\Delta(F) \leq 2$, every component of F is a path or a cycle. Furthermore, every path or cycle in F alternates between edges of $M - M'$ and edges of $M' - M$. Thus each cycle has even length, with an equal number of edges from M and from M'. ■

3.1.10. Theorem. (Berge [1957]) A matching M in a graph G is a maximum matching in G if and only if G has no M-augmenting path.

Proof: We prove the contrapositive of each direction; G has a matching larger than M if and only if G has an M-augmenting path. We have observed that an M-augmenting path can be used to produce a matching larger than M.

For the converse, let M' be a matching in G larger than M; we construct an M-augmenting path. Let $F = M \triangle M'$. By Lemma 3.1.9, F consists of paths and even cycles; the cycles have the same number of edges from M and M'. Since $|M'| > |M|$, F must have a component with more edges of M' than of M. Such a component can only be a path that starts and ends with an edge of M'; thus it is an M-augmenting path in G. ∎

HALL'S MATCHING CONDITION

When we are filling jobs with applicants, there may be many more applicants than jobs; successfully filling the jobs will not use all applicants. To model this problem, we consider an X, Y-bigraph (bipartite graph with bipartition X, Y—Definition 1.2.17), and we seek a matching that saturates X.

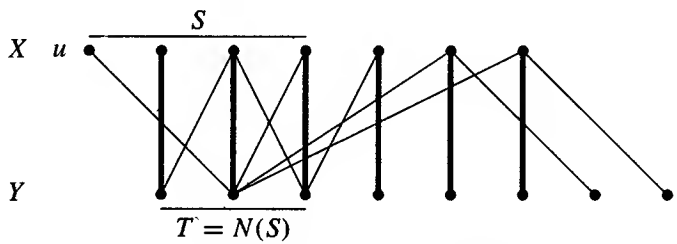
If a matching M saturates X, then for every $S \subseteq X$ there must be at least $|S|$ vertices that have neighbors in S, because the vertices matched to S must be chosen from that set. We use $N_G(S)$ or simply $N(S)$ to denote the set of vertices having a neighbor in S. Thus $|N(S)| \geq |S|$ is a necessary condition.

The condition "For all $S \subseteq X$, $|N(S)| \geq |S|$" is **Hall's Condition**. Hall proved that this obvious necessary condition is also sufficient (TONCAS).

3.1.11. Theorem. (Hall's Theorem—P. Hall [1935]) An X, Y-bigraph G has a matching that saturates X if and only if $|N(S)| \geq |S|$ for all $S \subseteq X$.

Proof: *Necessity.* The $|S|$ vertices matched to S must lie in $N(S)$.

Sufficiency. To prove that Hall's Condition is sufficient, we prove the contrapositive. If M is a maximum matching in G and M does not saturate X, then we obtain a set $S \subseteq X$ such that $|N(S)| < |S|$. Let $u \in X$ be a vertex unsaturated by M. Among all the vertices reachable from u by M-alternating paths in G, let S consist of those in X, and let T consist of those in Y (see figure below with M in bold). Note that $u \in S$.



We claim that M matches T with $S - \{u\}$. The M-alternating paths from u reach Y along edges not in M and return to X along edges in M. Hence every vertex of $S - \{u\}$ is reached by an edge in M from a vertex in T. Since there is no M-augmenting path, every vertex of T is saturated; thus an M-alternating

3.2. Algorithms and Applications

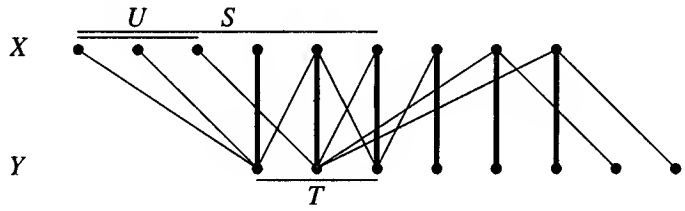
MAXIMUM BIPARTITE MATCHING

To find a maximum matching, we iteratively seek augmenting paths to enlarge the current matching. In a bipartite graph, if we don't find an augmenting path, we will find a vertex cover with the same size as the current matching, thereby proving that the current matching has maximum size. This yields both an algorithm to solve the maximum matching problem and an algorithmic proof of the König–Egerváry Theorem.

Given a matching M in an X, Y-bigraph G, we search for M-augmenting paths from each M-unsaturated vertex in X. We need only search from vertices in X, because every augmenting path has odd length and thus has ends in both X and Y. We will search from the unsaturated vertices in X simultaneously. Starting with a matching of size 0, $\alpha'(G)$ applications of the Augmenting Path Algorithm produce a maximum matching.

3.2.1. Algorithm. (Augmenting Path Algorithm).
Input: An X, Y-bigraph G, a matching M in G, and the set U of M-unsaturated vertices in X.
Idea: Explore M-alternating paths from U, letting $S \subseteq X$ and $T \subseteq Y$ be the sets of vertices reached. *Mark* vertices of S that have been explored for path extensions. As a vertex is reached, record the vertex from which it is reached.
Initialization: $S = U$ and $T = \emptyset$.
Iteration: If S has no unmarked vertex, stop and report $T \cup (X - S)$ as a minimum cover and M as a maximum matching. Otherwise, select an unmarked $x \in S$. To explore x, consider each $y \in N(x)$ such that $xy \notin M$. If y is unsaturated, terminate and report an M-augmenting path from U to y. Otherwise, y is matched to some $w \in X$ by M. In this case, include y in T (reached from x)

and include w in S (reached from y). After exploring all such edges incident to x, mark x and iterate. ∎



When exploring x in the iterative step, we may reach a vertex $y \in T$ that we have reached previously. Recording x as the previous vertex on the path may change which M-augmenting path we report, but it won't change whether such a path exists.

3.2.2. Theorem. Repeatedly applying the Augmenting Path Algorithm to a bipartite graph produces a matching and a vertex cover of equal size.

Proof: We need only verify that the Augmenting Path Algorithm produces an M-augmenting path or a vertex cover of size $|M|$. If the algorithm produces an M-augmenting path, we are finished. Otherwise, it terminates by marking all vertices of S and claiming that $R = T \cup (X - S)$ is a vertex cover of size $|M|$. We must prove that R is a vertex cover and has size $|M|$.

To show that R is a vertex cover, it suffices to show that there is no edge joining S to $Y - T$. An M-alternating path from U enters X only on an edge of M. Hence every vertex x of $S - U$ is matched via M to a vertex of T, and there is no edge of M from S to $Y - T$. Also there is no such edge outside M. When the path reaches $x \in S$, it can continue along any edge not in M, and exploring x puts all other neighbors of x into T. Since the algorithm marks all of S before terminating, all edges from S go to T.

Now we study the size of R. The algorithm puts only saturated vertices in T; each $y \in T$ is matched via M to a vertex of S. Since $U \subseteq S$, also each vertex of $X - S$ is saturated, and the edges of M incident to $X - S$ cannot involve T. Hence they are different from the edges saturating T, and we find that M has at least $|T| + |X - S|$ edges. Since there is no matching larger than this vertex cover, we have $|M| = |T| + |X - S| = |R|$. ∎

In addition to studying the correctness of algorithms, we are concerned about the time (number of computational steps) they use. We measure this as a function of the size of the input. For graph problems, we usually use the order $n(G)$ and/or size $e(G)$ to measure the input size.

3.2.3. Definition. The **running time** of an algorithm is the maximum number of computational steps used, expressed as a function of the size of the input. A **good algorithm** is one that has polynomial running time.

Running time is often expressed as "$O(f)$", where f is a function of the

size of the input. Here $O(f)$ denotes the set of functions g such that $|g(x)|$ is bounded by a constant multiple of $|f(x)|$ when x is sufficiently large (that is, there exist c, a such that $|g(x)| \le c\,|f(x)|$ when $|x| \ge a$).

Many problems we study in Chapters 1-4 have good algorithms; other notions of complexity (Appendix B) need not trouble us yet. Since we don't know how long a particular operation may take on a particular computer, constant factors in running time have little meaning. Hence the "Big Oh" notation $O(f)$ is convenient. When f is a quadratic polynomial, we typically abuse notation by writing $O(n^2)$ instead of $O(f)$ to describe functions that grow at most quadratically in terms of n.

3.2.4. Remark. Let G be an X, Y-bigraph with n vertices and m edges. Since $\alpha'(G) \le n/2$, we find a maximum matching in G by applying Algorithm 3.2.1 at most $n/2$ times. Each application explores a vertex of X at most once, just before marking it; thus it considers each edge at most once. If the time for one edge exploration is bounded by a constant, then this algorithm to find a maximum matching runs in time $O(nm)$. Theorem 3.2.22 presents a faster algorithm, with running time $O(\sqrt{n}m)$. Section 3.3 discusses a good algorithm for maximum matching in general graphs. ∎

WEIGHTED BIPARTITE MATCHING

Our results on maximum matching generalize to weighted X, Y-bigraphs, where we seek a matching of maximum total weight. If our graph is not all of $K_{n,n}$, then we insert the missing edges and assign them weight 0. This does not affect the numbers we can obtain as the weight of a matching. Thus we assume that our graph is $K_{n,n}$.

Since we consider only nonnegative edge weights, some maximum weighted matching is a perfect matching; thus we seek a perfect matching. We solve both the maximum weighted matching problem and its dual.

3.2.5. Example. *Weighted bipartite matching and its dual.* A farming company owns n farms and n processing plants. Each farm can produce corn to the capacity of one plant. The profit that results from sending the output of farm i to plant j is $w_{i,j}$. Placing weight $w_{i,j}$ on edge $x_i y_j$ gives us a weighted bipartite graph with partite sets $X = \{x_1, \ldots, x_n\}$ and $Y = \{y_1, \ldots, y_n\}$. The company wants to select edges forming a matching to maximize total profit.

The government claims that too much corn is being produced, so it will pay the company not to process corn. The government will pay u_i if the company agrees not to use farm i and v_j if it agrees not to use plant j. If $u_i + v_j < w_{i,j}$, then the company makes more by using the edge $x_i y_j$ than by taking the government payments for those vertices. In order to stop all production, the government must offer amounts such that $u_i + v_j \ge w_{i,j}$ for all i, j. The government wants to find such values to minimize $\sum u_i + \sum v_j$. ∎

3.2.6. Definition. A **transversal** of an n-by-n matrix consists of n positions, one in each row and each column. Finding a transversal with maximum sum is the **Assignment Problem**. This is the matrix formulation of the **maximum weighted matching** problem, where nonnegative weight $w_{i,j}$ is assigned to edge $x_i y_j$ of $K_{n,n}$ and we seek a perfect matching M to maximize the total weight $w(M)$.

With these weights, a **(weighted) cover** is a choice of labels $u_i, \ldots, u_n$ and $v_j, \ldots, v_n$ such that $u_i + v_j \geq w_{i,j}$ for all i, j. The **cost** $c(u, v)$ of a cover (u, v) is $\sum u_i + \sum v_j$. The **minimum weighted cover** problem is that of finding a cover of minimum cost.

Note that the problem of minimum weight perfect matching can be solved using maximum weight matching; simply replace each weight $w_{i,j}$ with $M - w_{i,j}$ for some large number M.

The next lemma shows that the weighted matching and weighted cover problems are dual problems.

3.2.7. Lemma. For a perfect matching M and cover (u, v) in a weighted bipartite graph G, $c(u, v) \geq w(M)$. Also, $c(u, v) = w(M)$ if and only if M consists of edges $x_i y_j$ such that $u_i + v_j = w_{i,j}$. In this case, M and (u, v) are optimal.

Proof: Since M saturates each vertex, summing the constraints $u_i + v_j \geq w_{i,j}$ that arise from its edges yields $c(u, v) \geq w(M)$ for every cover (u, v). Furthermore, if $c(u, v) = w(M)$, then equality must hold in each of the n inequalities summed. Finally, since $c(u, v) \geq w(M)$ for every matching and every cover, $c(u, v) = w(M)$ implies that there is no matching with weight greater than $c(u, v)$ and no cover with cost less than $w(M)$. ∎

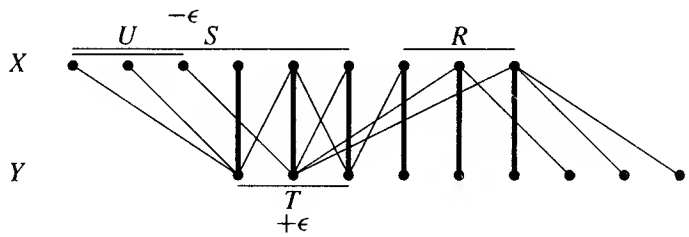
A matching and a cover have the same value only when the edges of the matching are covered with equality. This leads us to an algorithm.

3.2.8. Definition. The **equality subgraph** $G_{u,v}$ for a cover (u, v) is the spanning subgraph of $K_{n,n}$ having the edges $x_i y_j$ such that $u_i + v_j = w_{i,j}$.

If $G_{u,v}$ has a perfect matching, then its weight is $\sum u_i + \sum v_j$, and by Lemma 3.2.7 we have the optimal solution. Otherwise, we find a matching M and a vertex cover Q of the same size in $G_{u,v}$ (by using the Augmenting Path Algorithm, for example). Let $R = Q \cap X$ and $T = Q \cap Y$. Our matching of size $|Q|$ consists of $|R|$ edges from R to $Y - T$ and $|T|$ edges from T to $X - R$, as shown below. To seek a larger matching in the equality subgraph, we change (u, v) to introduce an edge from $X - R$ to $Y - T$ while maintaining equality on all edges of M.

A cover requires $u_i + v_j \geq w_{i,j}$ for all i, j; the difference $u_i + v_j - w_{i,j}$ is the **excess** for i, j. Edges joining $X - R$ and $Y - T$ are not in $G_{u,v}$ and have positive excess. Let ϵ be the minimum excess on the edges from $X - R$ to $Y - T$. Reducing u_i by ϵ for all $x_i \in X - R$ maintains the cover condition for these edges while bringing at least one into the equality subgraph. To maintain the cover condition for the edges from $X - R$ to T, we also increase v_j by ϵ for $y_j \in T$.

We repeat the procedure with the new equality subgraph; eventually we obtain a cover whose equality subgraph has a perfect matching. The resulting algorithm was named the **Hungarian Algorithm** by Kuhn in honor of the work of König and Egerváry on which it is based.



3.2.9. Algorithm. (Hungarian Algorithm—Kuhn [1955], Munkres [1957]).
Input: A matrix of weights on the edges of $K_{n,n}$ with bipartition X, Y.
Idea: Iteratively adjusting the cover (u, v) until the equality subgraph $G_{u,v}$ has a perfect matching.
Initialization: Let (u, v) be a cover, such as $u_i = \max_j w_{i,j}$ and $v_j = 0$.
Iteration: Find a maximum matching M in $G_{u,v}$. If M is a perfect matching, stop and report M as a maximum weight matching. Otherwise, let Q be a vertex cover of size $|M|$ in $G_{u,v}$. Let $R = X \cap Q$ and $T = Y \cap Q$. Let

$$\epsilon = \min\{u_i + v_j - w_{i,j} : x_i \in X - R,\ y_j \in Y - T\}.$$

Decrease u_i by ϵ for $x_i \in X - R$, and increase v_j by ϵ for $y_j \in T$. Form the new equality subgraph and repeat. ■

We have presented the algorithm using bipartite graphs, but repeatedly drawing a changing equality subgraph is awkward. Therefore, we compute with matrices. The initial weights form a matrix A with $w_{i,j}$ in position i, j. We associate the vertices and the labels (u, v) with the rows and columns, which serve as X and Y, respectively. We subtract $w_{i,j}$ from $u_i + v_j$ to obtain the **excess matrix**: $c_{i,j} = u_i + v_j - w_{i,j}$. The edges of the equality subgraph correspond to 0s in the excess matrix.

3.2.10. Example. *Solving the Assignment Problem.* The first matrix below is the matrix of weights. The others display a cover (u, v) and the corresponding excess matrix. We underscore entries in the excess matrix to mark a maximum matching M of $G_{u,v}$, which appears as bold edges in the equality subgraphs drawn for the first two excess matrices. (Drawing the equality subgraphs is not necessary.) A matching in $G_{u,v}$ corresponds to a set of 0s in the excess matrix with no two in any row or column; call this a **partial transversal**.

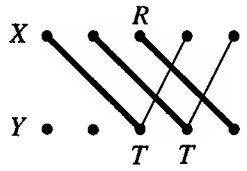
A set of rows and columns covering the 0s in the excess matrix is a **covering set**; this corresponds to a vertex cover in $G_{u,v}$. A covering set of size less than n yields progress toward a solution, since the next weighted cover costs less. We study the 0s in the excess matrix and find a partial transversal and a covering set of the same size. In a small matrix, we can do this by inspection.

We underscore the 0s of a partial transversal, and we use Rs and Ts to label the rows and columns of the covering set. At each iteration, we compute the minimum excess on the positions *not* in a covered row or column (in rows $X - R$ and columns $Y - T$). These uncovered positions have positive excess (the corresponding edges are not in the equality subgraph). The value ϵ defined in Algorithm 3.2.9 is the minimum of these excesses. We reduce the label u_i by ϵ on rows not in R and increase the label v_j by ϵ on columns in T.

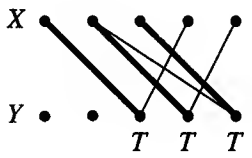
In the example below, the covering set used in the first iteration reduces the cost of the cover but does not augment the maximum matching in the equality subgraph. The second iteration produces a perfect matching. Using the last three columns as a covering set in the first iteration would augment the matching immediately.

The transversal of 0s after the final iteration identifies a perfect matching whose total weight equals the cost of the final cover. The corresponding edges have weights 5, 4, 6, 8, 8 in the original data, which sum to 31. The labels 4, 5, 7, 4, 6 and 0, 0, 2, 2, 1 in the final cover satisfy each edge exactly and also sum to 31. The value of the optimal solution is unique, but the solution itself is not; this example has many maximum weight matchings and many minimum cost covers, but all have total weight 31. ∎

$$\begin{pmatrix} 4 & 1 & 6 & 2 & 3 \\ 5 & 0 & 3 & 7 & 6 \\ 2 & 3 & 4 & 5 & 8 \\ 3 & 4 & 6 & 3 & 4 \\ 4 & 6 & 5 & 8 & 6 \end{pmatrix} \rightarrow \begin{array}{c} \begin{matrix} 0 & 0 & 0 & 0 & 0 \end{matrix} \\ \begin{matrix} 6 \\ 7 \\ 8 \\ 6 \\ 8 \end{matrix} \begin{pmatrix} 2 & 5 & \underline{0} & 4 & 3 \\ 2 & 7 & 4 & \underline{0} & 1 \\ 6 & 5 & 4 & 3 & \underline{0} \\ 3 & 2 & 0 & 3 & 2 \\ 4 & 2 & 3 & 0 & 2 \end{pmatrix} R \\ \begin{matrix} & & T & T & \end{matrix} \end{array}$$



$$\begin{array}{c} \begin{matrix} 0 & 0 & 1 & 1 & 0 \end{matrix} \\ \begin{matrix} 5 \\ 6 \\ 8 \\ 5 \\ 7 \end{matrix} \begin{pmatrix} 1 & 4 & \underline{0} & 4 & 2 \\ 1 & 6 & 4 & \underline{0} & 0 \\ 6 & 5 & 5 & 4 & \underline{0} \\ 2 & 1 & 0 & 3 & 1 \\ 3 & 1 & 3 & 0 & 1 \end{pmatrix} \\ \begin{matrix} & & T & T & T \end{matrix} \end{array}$$



$$\rightarrow \begin{array}{c} \begin{matrix} 0 & 0 & 2 & 2 & 1 \end{matrix} \\ \begin{matrix} 4 \\ 5 \\ 7 \\ 4 \\ 6 \end{matrix} \begin{pmatrix} 0 & 3 & \underline{0} & 4 & 2 \\ \underline{0} & 5 & 4 & 0 & 0 \\ 5 & 4 & 5 & 4 & \underline{0} \\ 1 & \underline{0} & 0 & 3 & 1 \\ 2 & 0 & 3 & \underline{0} & 1 \end{pmatrix} \end{array}$$

3.2.11. Theorem. The Hungarian Algorithm finds a maximum weight matching and a minimum cost cover.

Proof: The algorithm begins with a cover. It can terminate only when the equality subgraph has a perfect matching, which guarantees equal value for the current matching and cover. Suppose that (u, v) is the current cover and that the equality subgraph has no perfect matching. Let (u', v') denote the new lists of numbers assigned to the vertices. Because ϵ is the minimum of a nonempty finite set of positive numbers, $\epsilon > 0$.

We verify first that (u', v') is a cover. The change of labels on vertices of $X - R$ and T yields $u'_i + v'_j = u_i + v_j$ for edges $x_i y_j$ from $X - R$ to T or from R to $Y - T$. If $x_i \in R$ and $y_j \in T$, then $u'_i + v'_j = u_i + v_j + \epsilon$, and the weight remains covered. If $x_i \in X - R$ and $y_j \in Y - T$, then $u'_i + v'_j$ equals $u_i + v_j - \epsilon$, which by the choice of ϵ is at least $w_{i,j}$.

The algorithm terminates only when the equality subgraph has a perfect matching, so it suffices to show that it does terminate. Suppose that the weights $w_{i,j}$ are rational. Multiplying the weights by their least common denominator yields an equivalent problem with integer weights. We can now assume that the labels in the current cover also are integers. Thus each excess is also an integer, and at each iteration we reduce the cost of the cover by an integer amount. Since the cost starts at some value and is bounded below by the weight of a perfect matching, after finitely many iterations we have equality.

For real-valued weights in general, see Remark 3.2.12). ∎

3.2.12.* Remark. When the weights are real numbers, the algorithm still works if we obtain vertex covers in the equality subgraph more carefully. We show that the algorithm terminates within n^2 iterations. Because the edges of M remain in the new equality subgraph, the size of the current matching never decreases. Since the size of the matching can increase at most n times, it suffices to show that it must increase within n iterations.

If we find the maximum matching M by iterating the Augmenting Path Algorithm, then the last iteration presents us with a vertex cover. We find it by exploring M-alternating paths from the set U of M-unsaturated vertices in X. With S and T denoting the sets of vertices reachable in X and T, we obtain the vertex cover $R \cup T$, where $R = X - S$.

Applying a step of the Hungarian Algorithm using the vertex cover $R \cup T$ maintains equality on M and all the edges in M-alternating paths from U. Edges from T to R disappear from the equality subgraph, but we don't care because they don't appear in M-alternating paths from U. Introducing an edge from S to $Y - T$ either creates an M-augmenting path or increases T while leaving U unchanged. Since we can increase T at most n times, we obtain a larger matching in the equality subgraph within n iterations. ∎

3.2.13.* Remark. The maximum matching and vertex cover problems in bipartite graphs are special cases of the weighted problems. Given a bipartite graph G, form a weighted graph with weight 1 on the edges of G and weight 0 on the edges of $K_{n,n}$. The maximum weight of a matching is $\alpha'(G)$.

Given integer weights, the Hungarian algorithm always maintains integer labels in the weighted cover. Hence in this weighted cover problem we may restrict the values (labels) used to be integers. Further thought shows that these integers will always be 0 or 1.

The vertices receiving label 1 must cover the weight on the edges of G, so they form a vertex cover for G. Minimizing the sum of labels under the integer restriction is equivalent to finding the minimum number of vertices in a vertex cover for G. Hence the answer to the weighted cover problem is $\beta(G)$. ∎

3.2.14.* Application. *Street Sweeping and the Transportation Problem.* A cleaning machine sweeping a curb must move in the same direction as traffic. This yields a digraph; a two-way street generates two oppositely directed edges, while a one-way street generates two edges in the same direction. We consider a simple version of the **Street Sweeping Problem**, discussed in more detail in Roberts [1978] as based on Tucker–Bodin [1976].

In New York City, parking is prohibited from some curbs each day to allow for street sweeping. For each day, this defines a **sweep subgraph** G of the full digraph H of curbs, consisting of those available for sweeping. Each $e \in E(H)$ has a **deadheading time** $t(e)$ needed to travel it without sweeping.

The question is how to sweep G while minimizing the total deadheading time spent without sweeping. This is a generalization of a directed version of the Chinese Postman Problem. If indegree equals outdegree at each vertex of G, then no deadheading is needed. Otherwise, we duplicate edges of G or add edges from H to obtain an Eulerian digraph G' containing G.

Let X be the set of vertices with excess indegree; let $\sigma(x) = d_G^-(x) - d_G^+(x)$ for $x \in X$. Let Y be the set with excess outdegree; let $\partial(y) = d_G^-(y) - d_G^+(y)$ for $y \in Y$. Note that $\sum_{x \in X} \sigma(x) = \sum_{y \in Y} \partial(y)$. To obtain G' from G, we must add $\sigma(x)$ edges with tails at $x \in X$ and $\partial(y)$ edges with heads at $y \in Y$. Since G' needs net outdegree 0 at each vertex, the additions form paths from X to Y. The cost $c(xy)$ of an x, y-path is the distance from x to y in the weighted digraph H, which can be found by Dijkstra's Algorithm.

This yields the **Transportation Problem**. Given supply $\sigma(x)$ for $x \in X$, demand $\partial(y)$ for $y \in Y$, cost $c(xy)$ per unit sent from x to y, and $\sum \sigma(x) = \sum \partial(y)$, we want to satisfy the demands at least total cost. A version of the problem was introduced by Kantorovich [1939]; the form above arose (with a constructive solution) in Hitchcock [1941] (see also Koopmans [1947]). The problem is discussed at length in Ford–Fulkerson [1962, p93–130].

When the supplies and demands are rational, the Assignment Problem can be applied. First scale up to obtain integer supplies and demands. Next define a matrix with $\sum \sigma(x)$ rows and columns. For each $x \in X$, create $\sigma(x)$ rows. For each $y \in Y$, create $\delta(y)$ columns. When row i and column j represent x and y, let $w_{i,j} = M - c(xy)$, where $M = \max_{x,y} c(xy)$. A maximum weight matching now yields a minimum cost solution to the Transportation Problem. A generalization of the Transportation Problem appears in Section 4.3. ■