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 $w(M)$. ∎

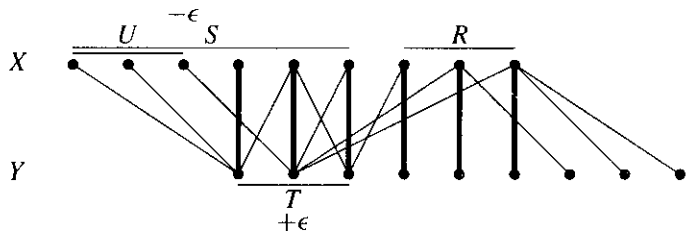
A matching and a cover have the same value only when the edges of the matching are covered with equality. This leads us to an algorithm.

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

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

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**.

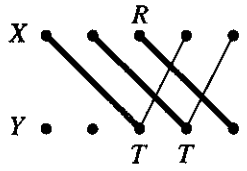
A set of rows and columns covering the 0s in the excess matrix is a **covering set**; this corresponds to a vertex cover in $G_{u,v}$. A covering set of size less than n yields progress toward a solution, since the next weighted cover costs less. We study the 0s in the excess matrix and find a partial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

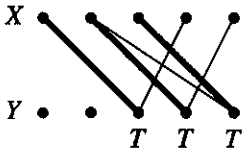
In the example below, the covering set used in the first iteration reduces the cost of the cover but does not augment the maximum matching in the equality subgraph. The second iteration produces a perfect matching. Using the last three columns as a covering set in the first iteration would augment the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

STABLE MATCHINGS (optional)

Instead of optimizing total weight for a matching, we may try to optimize using preferences. Given n men and n women; we want to establish n "stable" marriages. If man x and woman a are paired with other partners, but x prefers a to his current partner and a prefers x to her current partner, then they might leave their current partners and switch to each other. In this situation we say that the unmatched pair (x, a) is an **unstable pair**.

3.2.15. Definition. A perfect matching is a **stable matching** if it yields no unstable unmatched pair.

3.2.16. Example. Given men x, y, z, w, women a, b, c, d, and preferences listed below, the matching $\{xa, yb, zd, wc\}$ is a stable matching. ■

Men $\{x, y, z, w\}$	Women $\{a, b, c, d\}$
$x : a > b > c > d$	$a : z > x > y > w$
$y : a > c > b > d$	$b : y > w > x > z$
$z : c > d > a > b$	$c : w > x > y > z$
$w : c > b > a > d$	$d : x > y > z > w$

In their paper "College admissions and the stability of marriage", Gale and Shapley proved that a stable matching always exists and can be found using a relatively simple algorithm. In the algorithm, men and women do not play symmetric roles; we will discuss this importance of this difference later. The algorithm below generates the matching of Example 3.2.16.

3.2.17. Algorithm. (Gale–Shapley Proposal Algorithm)
Input: Preference rankings by each of n men and n women.
Idea: Produce a stable matching using proposals by maintaining information about who has proposed to whom and who has rejected whom.
Iteration: Each man proposes to the highest woman on his preference list who has not previously rejected him. If each woman receives exactly one proposal, stop and use the resulting matching. Otherwise, every woman receiving more than one proposal rejects all of them except the one that is highest on her preference list. Every woman receiving a proposal says "maybe" to the most attractive proposal received. ■

3.2.18. Theorem. (Gale–Shapley [1962]) The Proposal Algorithm produces a stable matching.

Proof: The algorithm terminates (with some matching), because on each nonterminal iteration, the total length of the lists of potential mates for the men decreases. This can happen only n^2 times.

Key Observation: the sequence of proposals made by each man is nonincreasing in his preference list, and the sequence of men to whom a woman says "maybe" is nondecreasing in her preference list, culminating in the man assigned. This holds because men propose repeatedly to the same woman until rejected, and women say "maybe" to the same man until a better offer arrives.

If the result is not stable, then there is an unstable unmatched pair (x, a), with x matched to b and y matched to a. By the key observation, x never proposed to a during the algorithm, since a received a mate less desirable than x. The key observation also implies that x would not have proposed to b without earlier proposing to a. This contradiction confirms the stability of the result. ■

The asymmetry of the proposal algorithm suggests asking which sex is happier. When the first choices of the men are distinct, they all get their first

choice, and the women are stuck with whomever proposed. When the algorithm runs with women proposing, every woman is at least as happy as when men do the proposing, and every man is at least as unhappy. In Example 3.2.16, running the algorithm with women proposing immediately yields the matching $\{xd, yb, ca, wc\}$, in which all women are matched to their first choices. In fact, among all stable matchings, every man is happiest in the one produced by the male-proposal algorithm, and every woman is happiest under the female-proposal algorithm (Exercise 11). Societal conventions thus favor men.

The algorithm is used in another setting. Each year, the graduates of medical schools submit preference lists of hospitals where they wish to be residents. The hospitals have their own preferences; we model a hospital with multiple openings as several hospitals with the same preference list. Chaos in the market for residents (then called interns) forced hospitals to devise and implement the algorithm ten years before the Gale–Shapley paper defined and solved the problem! The result was the National Resident Matching Program, a nonprofit corporation established in 1952 to provide a uniform appointment date and matching procedure.

Who is happier with the outcome? Since the medical organizations ran the algorithm, it is not surprising that initially they did the proposing and were happier with the outcome. The distinction is even clearer in another setting; students applying for jobs have preferences, but the employers make the proposals, called "job offers". Unhappiness with the NRMP caused the system to be changed in 1998 to a student-proposing algorithm. In 1998 the system processed 35,823 applicants for 22,451 positions. Additional details about the system can be found at nrmp.aamc.org/nrmp/mainguid/ on the World Wide Web.

There may be stable matchings other than those found by the two versions of the proposal algorithm. To seek a "fair" stable matching, we could give each person a number of points with which to rate preferences. The weight for the pair xa is then the sum of the points that x gives to a and a gives to x. The Hungarian Algorithm would yield a matching of maximum total weight, but this might not be a stable matching (Exercise 10). Other approaches appear in the books Knuth [1976] and Gusfield–Irving [1989], which discuss stable marriages and related topics.

FASTER BIPARTITE MATCHING (optional)

We began this section with an algorithm for finding maximum matchings in bipartite graphs. The running time can be improved by seeking augmenting paths in a clever order; when short augmenting paths are available, we needn't explore many edges to find one. Using a Breadth-First Search simultaneously from all the unsaturated vertices of X, we can find many paths of the same length with one examination of the edge set. Hopcroft and Karp [1973] proved that subsequent augmentations must use longer paths, so the searches can be grouped in phases finding paths of the same lengths. They combined these

ideas to show that few phases are needed, enabling maximum matchings in n-vertex bipartite graphs to be found in $O(n^{2.5})$ time.

3.2.19. Remark. If M is a matching of size r and M^* is a matching of size $s > r$, then there exist at least $s - r$ vertex-disjoint M-augmenting paths. At least this many such paths can be found in $M \triangle M^*$. ∎

The next lemma implies that the sequence of path lengths in successive shortest augmentations is nondecreasing. Here we treat paths as sets of edges, and cardinality indicates number of edges.

3.2.20. Lemma. If P is a shortest M-augmenting path and P' is $M \triangle P$-augmenting, then $|P'| \geq |P| + 2|P \cap P'|$ (treating P as an edge set).

Proof: Note that $M \triangle P$ is the matching obtained by using P to augment M. Let N be the matching $(M \triangle P) \triangle P'$ obtained by using P' to augment $M \triangle P$. Since $|N| = |M| + 2$, Remark 3.2.19 guarantees that $M \triangle N$ contains two disjoint M-augmenting paths P_1 and P_2. Each of these is at least as long as P, since P is a shortest M-augmentating path.

Since N is obtained from M by switching the edges in P and then switching the edges in P', an edge belongs to exactly one of M and N if and only if it belongs to exactly one of P and P'. Therefore, $M \triangle N = P \triangle P'$. This yields $|P \triangle P'| \geq |P_1| + |P_2| \geq 2|P|$. Thus

$$2|P| \leq |P \triangle P'| = |P| + |P'| - 2|P \cap P'|.$$

We conclude that $|P'| \geq |P| + 2|P \cap P'|$. ∎

3.2.21. Lemma. If $P_1, P_2, \cdots$ is a list of successive shortest augmentations, then the augmentations of the same length are vertex-disjoint paths.

Proof: We use the method of contradiction. Let P_k, P_l with $l > k$ be a closest pair in the list that have the same size but are not vertex-disjoint. By Lemma 3.2.20, the lengths of successive shortest augmenting paths are nondecreasing, so $P_k, \ldots, P_l$ all have the same length. Since P_k, P_l is a closest intersecting pair with the same length, the paths $P_{k+1}, \ldots, P_l$ are pairwise disjoint.

Let M' be the matching given by the augmentations $P_1, \ldots, P_k$. Since $P_{k+1}, \ldots, P_l$ are pairwise disjoint, P_l is an M'-augmenting path. By Lemma 3.2.20, $|P_l| \geq |P_k| + |P_l \cap P_k|$. Since $|P_l| = |P_k|$, there is no common edge.

On the other hand, there must be a common edge. Each vertex of P_k is saturated in M' using an edge of P_k, and every vertex of an M'-augmenting path P_l that is saturated in M' (such as a vertex common to P_l and P_k) must contribute its saturated edge to P_l.

The contradiction implies that there is no such pair P_k, P_l. ∎

3.2.22. Theorem. (Hopcroft–Karp [1973]) The breadth-first phased maximum matching algorithm runs in $O(\sqrt{n}m)$ time on bipartite graphs with n vertices and m edges.

Proof: By Lemmas 3.2.20–3.2.21, searching simultaneously from all unsaturated vertices of X for shortest augmentations yields vertex-disjoint paths, after which all other augmenting paths are longer. Hence the augmentations of each length are found in one examination of the edge set, running in time $O(m)$. It suffices to prove that there are at most $2\lfloor\sqrt{n/2}\rfloor + 2$ phases.

List the augmenting paths as $P_1, \ldots, P_s$ in order by length, with $s = \alpha'(G) \leq n/2$. Since paths of the same length are vertex-disjoint, each P_{i+1} is an augmenting path for the matching M_i formed by using $P_1, \ldots, P_i$. It suffices to prove the more general statement that whenever $P_1, \ldots, P_s$ are successive shortest augmenting paths that build a maximum matching, the number of distinct lengths among these paths is at most $2\lfloor\sqrt{s}\rfloor + 2$.

Let $r = \lfloor s - \sqrt{s}\rfloor$. Because $|M_r| = r$ and the maximum matching has size s, Remark 3.2.19 yields at least $s - r$ vertex-disjoint M_r-augmenting paths. The shortest of these paths uses at most $\lfloor r/(s-r)\rfloor$ edges from M_r. Hence $|P_{r+1}| \leq 2\lfloor r/(s-r)\rfloor + 1$. Since $\lfloor r/(s-r)\rfloor < \lfloor s/\lceil\sqrt{s}\rceil\rfloor \leq \lfloor\sqrt{s}\rfloor$, the paths up to P_r provide all but the last $\lceil\sqrt{s}\rceil$ augmentations using length at most $2\lfloor\sqrt{s}\rfloor + 1$. There are at most $\lfloor\sqrt{s}\rfloor + 1$ distinct odd integers up to this value, and even if the last $\lceil\sqrt{s}\rceil$ paths have distinct lengths, they provide at most $\lfloor\sqrt{s}\rfloor + 1$ additional lengths, so altogether we use at most $2\lfloor\sqrt{s}\rfloor + 2$ distinct lengths. ∎

Even and Tarjan [1975] extended this to solve in time $O(\sqrt{n}m)$ a more general problem that includes maximum bipartite matching.

EXERCISES

3.2.1. (−) Using nonnegative edge weights, construct a 4-vertex weighted graph in which the matching of maximum weight is not a matching of maximum size.

3.2.2. (−) Show how to use the Hungarian Algorithm to test for the existence of a perfect matching in a bipartite graph.

3.2.3. (∗−) Give an example of the stable matching problem with two men and two women in which there is more than one stable matching.

3.2.4. (∗−) Determine the stable matchings resulting from the Proposal Algorithm run with men proposing and with women proposing, given the preference lists below.

Men $\{u, v, w, x, y, z\}$	Women $\{a, b, c, d, e, f\}$
$u: a > b > d > c > f > e$	$a: z > x > y > u > v > w$
$v: a > b > c > f > e > d$	$b: y > z > w > x > v > u$
$w: c > b > d > a > f > e$	$c: v > x > w > y > u > z$
$x: c > a > d > b > e > f$	$d: w > y > u > x > z > v$
$y: c > d > a > b > f > e$	$e: u > v > x > w > y > z$
$z: d > e > f > c > b > a$	$f: u > w > x > v > z > y$

• • • • •

3.2.5. Find a transversal of maximum total sum (weight) in each matrix below. Prove that there is no larger weight transversal by exhibiting a solution to the dual problem. Explain why this proves that there is no larger transversal.

(a)	(b)	(c)
4 4 4 3 6	7 8 9 8 7	1 2 3 4 5
1 1 4 3 4	8 7 6 7 6	6 7 8 7 2
1 4 5 3 5	9 6 5 4 6	1 3 4 4 5
5 6 4 7 9	8 5 7 6 4	3 6 2 8 7
5 3 6 8 3	7 6 5 5 5	4 1 3 5 4

3.2.6. Find a minimum-weight transversal in the matrix below, and use duality to prove that the solution is optimal. (Hint: Use a transformation of the problem.)

$$\begin{pmatrix} 4 & 5 & 8 & 10 & 11 \\ 7 & 6 & 5 & 7 & 4 \\ 8 & 5 & 12 & 9 & 6 \\ 6 & 6 & 13 & 10 & 7 \\ 4 & 5 & 7 & 9 & 8 \end{pmatrix}$$

3.2.7. *The Bus Driver Problem.* Let there be n bus drivers, n morning routes with durations $x_1, \ldots, x_n$, and n afternoon routes with durations $y_1, \ldots, y_n$. A driver is paid overtime when the morning route and afternoon route exceed total time t. The objective is to assign one morning run and one afternoon run to each driver to minimize the total amount of overtime. Express this as a weighted matching problem. Prove that giving the ith longest morning route and ith shortest afternoon route to the same driver, for each i, yields an optimal solution. (Hint: Do not use the Hungarian Algorithm; consider the special structure of the matrix.) (R.B. Potts)

3.2.8. Let the entries in matrix A have the form $w_{i,j} = a_i b_j$, where $a_1, \ldots, a_n$ are numbers associated with the rows and $b_1, \ldots, b_n$ are numbers associated with the columns. Determine the maximum weight of a transversal of A. What happens when $w_{i,j} = a_i + b_j$? (Hint: In each case, guess the general pattern by examining the solution when $n = 2$.)

3.2.9. (*) A mathematics department offers k seminars in different topics to its n students. Each student will take one seminar; the ith seminar will have k_i students, where $\sum k_i = n$. Each student submits a preference list ranking the k seminars. An assignment of the students to seminars is *stable* if no two students can both obtain more preferable seminars by switching their assignments. Show how to find a stable assignment using weighted bipartite matching. (Isaak)

3.2.10. (*) Consider n men and n women, each assigning $n - i$ points to the ith person in his or her preference list. Let the weight of a pair be the sum of the points assigned by those two people. Construct an example where no maximum weight matching is a stable matching.

3.2.11. (*!) Prove that if man x is paired with woman a in some stable matching, then a does not reject x in the Gale–Shapley Proposal Algorithm with men proposing. Conclude that among all stable matchings, *every* man is happiest in the matching produced by this algorithm. (Hint: Consider the first occurrence of such a rejection.)

3.2.12. (*) In the Stable Roommates Problem, each of $2n$ people has a preference ordering on the other $2n - 1$ people. A stable matching is a perfect matching such that no

unmatched pair prefers each other to their current roommates. Prove that there is no stable matching when the preferences are those below. (Gale–Shapley [1962])

$$a : b > c > d$$
$$b : c > a > d$$
$$c : a > b > d$$
$$d : a > b > c$$

3.2.13. (∗) In the stable roommates problem, suppose that each individual declares a top portion of the preference list as "acceptable". Define the *acceptability graph* to be the graph whose vertices are the people and whose edges are the pairs of people who rank each other as acceptable. Prove that all sets of rankings with acceptability graph G lead to a stable matching if and only if G is bipartite. (Abeledo–Isaak [1991]).

3.3. Matchings in General Graphs

When discussing perfect matchings in graphs, it is natural to consider more general spanning subgraphs.

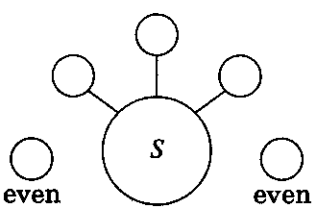
3.3.1. Definition. A **factor** of a graph G is a spanning subgraph of G. A **k-factor** is a spanning k-regular subgraph. An **odd component** of a graph is a component of odd order; the number of odd components of H is $o(H)$.

3.3.2. Remark. A 1-factor and a perfect matching are almost the same thing. The precise distinction is that "1-factor" is a spanning 1-regular subgraph of G, while "perfect matching" is the set of edges in such a subgraph.

A 3-regular graph that has a perfect matching decomposes into a 1-factor and a 2-factor. ∎

TUTTE'S 1-FACTOR THEOREM

Tutte found a necessary and sufficient condition for which graphs have 1-factors. If G has a 1-factor and we consider a set $S \subseteq V(G)$, then every odd component of $G - S$ has a vertex matched to something outside it, which can only belong to S. Since these vertices of S must be distinct, $o(G - S) \leq |S|$.



The condition "For all $S \subseteq V(G)$, $o(G - S) \leq |S|$" is **Tutte's Condition**. Tutte proved that this obvious necessary condition is also sufficient (TONCAS).

Many proofs are known, such as Exercise 13 and Exercise 27. We present the proof by Lovász using the ideas of symmetric difference and extremality.

3.3.3. Theorem. (Tutte [1947]) A graph G has a 1-factor if and only if $o(G - S) \leq |S|$ for every $S \subseteq V(G)$.

Proof: (Lovász [1975]). *Necessity*. The odd components of $G - S$ must have vertices matched to distinct vertices of S.

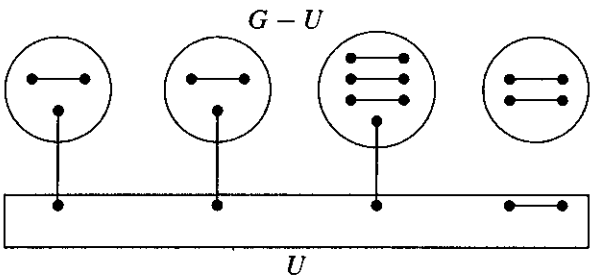
Sufficiency. When we add an edge joining two components of $G - S$, the number of odd components does not increase (odd and even together become one odd component, two components of the same parity become one even component). Hence Tutte's Condition is preserved by addition of edges: if $G' = G + e$ and $S \subseteq V(G)$, then $o(G' - S) \leq o(G - S) \leq |S|$. Also, if $G' = G + e$ has no 1-factor, then G has no 1-factor.

Therefore, the theorem holds unless there exists a simple graph G such that G satisfies Tutte's Condition, G has no 1-factor, and adding any missing edge to G yields a graph with a 1-factor. Let G be such a graph. We obtain a contradiction by showing that G actually does contain a 1-factor.

Let U be the set of vertices in G that have degree $n(G) - 1$.

Case 1: $G - U$ consists of disjoint complete graphs. In this case, the vertices in each component of $G - U$ can be paired in any way, with one extra in the odd components. Since $o(G - U) \leq |U|$ and each vertex of U is adjacent to all of $G - U$, we can match the leftover vertices to vertices of U.

The remaining vertices are in U, which is a clique. To complete the 1-factor, we need only show that an even number of vertices remain in U. We have matched an even number, so it suffices to show that $n(G)$ is even. This follows by invoking Tutte's Condition for $S = \emptyset$, since a graph of odd order would have a component of odd order.

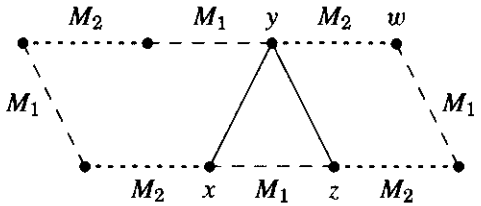


Case 2: $G - U$ is not a disjoint union of cliques. In this case, $G - U$ has two vertices at distance 2; these are nonadjacent vertices x, z with a common neighbor $y \notin U$ (Exercise 1.2.23a). Furthermore, $G - U$ has another vertex w not adjacent to y, since $y \notin U$. By the choice of G, adding an edge to G creates a 1-factor; let M_1 and M_2 be 1-factors in $G + xz$ and $G + yw$, respectively. It suffices to show that $M_1 \triangle M_2$ contains a 1-factor avoiding xz and yw, because this will be a 1-factor in G.

Let $F = M_1 \triangle M_2$. Since $xz \in M_1 - M_2$ and $yw \in M_2 - M_1$, both xz and yw are in F. Since every vertex of G has degree 1 in each of M_1 and M_2, every vertex of G has degree 0 or 2 in F. Hence the components of F are even cycles and isolated vertices (see Lemma 3.1.9). Let C be the cycle of F containing xz.

If C does not also contain yw, then the desired 1-factor consists of the edges of M_2 from C and all of M_1 not in C.

If C contains both yw and xz, as shown below, then to avoid them we use yx or yz. In the portion of C starting from y along yw, we use edges of M_1 to avoid using yw. When we reach $\{x, z\}$, we use zy if we arrive at z (as shown); otherwise, we use xy. In the remainder of C we use the edges of M_2. We have produced a 1-factor of C that does not use xz or yw. Combined with M_1 or M_2 outside C, we have a 1-factor of G. ∎



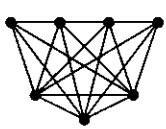
3.3.4. Remark. Like other characterization theorems (such as Theorem 1.2.18 and Theorem 3.1.11)), Theorem 3.3.3 yields short verifications both when the property holds *and* when it doesn't. We prove that G has a 1-factor exists by exhibiting one. When it doesn't exist, Theorem 3.3.3 guarantees that we can exhibit a set whose deletion leaves too many odd components. ∎

3.3.5. Remark. For a graph G and any $S \subseteq V(G)$, counting the vertices modulo 2 shows that $|S| + o(G-S)$ has the same parity as $n(G)$. Thus also the difference $o(G - S) - |S|$ has the same parity as $n(G)$. We conclude that if $n(G)$ is even and G has no 1-factor, then $o(G - S)$ exceeds $|S|$ by at least 2 for some S. ∎

For non-bipartite graphs (such as odd cycles), there may be a gap between $\alpha'(G)$ and $\beta(G)$ (see also Exercise 10). Nevertheless, another minimization problem yields a min-max relation for $\alpha'(T)$ in general graphs. This min-max relation generalizes Remark 3.3.5. The proof uses a graph transformation that involves a general graph operation.

3.3.6. Definition. The **join** of simple graphs G and H, written $G \vee H$, is the graph obtained from the disjoint union $G + H$ by adding the edges $\{xy : x \in V(G), y \in V(H)\}$.

$P_4 \vee K_3$

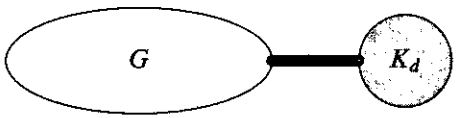


3.3.7. Corollary. (Berge–Tutte Formula—Berge [1958]) The largest number of vertices saturated by a matching in G is $\min_{S\subseteq V(G)}\{n(G) - d(S)\}$, where $d(S) = o(G - S) - |S|$.

Proof: Given $S \subseteq V(G)$, at most $|S|$ edges can match vertices of S to vertices in odd components of $G - S$, so every matching has at least $o(G - S) - |S|$ unsaturated vertices. We want to achieve this bound.

Let $d = \max\{o(G - S) - |S| : S \subseteq V(G)\}$. The case $S = \emptyset$ yields $d \geq 0$. Let $G' = G \vee K_d$. Since $d(S)$ has the same parity as $n(G)$ for each S, we know that $n(G')$ is even. If G' satisfies Tutte's Condition, then we obtain a matching of the desired size in G from a perfect matching in G', because deleting the d added vertices eliminates edges that saturate at most d vertices of G.

The condition $o(G' - S') \leq |S'|$ holds for $S' = \emptyset$ because $n(G')$ is even. If S' is nonempty but does not contain all of K_d, then $G' - S'$ has only one component, and $1 \leq |S'|$. Finally, when $K_d \subseteq S'$, we let $S = S' - V(K_d)$. We have $G' - S' = G - S$, so $o(G' - S') = o(G - S) \leq |S| + d = |S'|$. We have verified that G' satisfies Tutte's Condition. ∎



Corollary 3.3.7 guarantees that there is a short PROOF that a maximum matching indeed has maximum size by exhibiting a vertex set S whose deletion leaves the appropriate number of odd components.

Most applications of Tutte's Theorem involve showing that some other condition implies Tutte's Condition and hence guarantees a 1-factor. Some were proved by other means long before Tutte's Theorem was available.

3.3.8. Corollary. (Petersen [1891]) Every 3-regular graph with no cut-edge has a 1-factor.

Proof: Let G be a 3-regular graph with no cut-edge. We prove that G satisfies Tutte's Condition. Given $S \subseteq V(G)$, we count the edges between S and the odd components of $G - S$. Since G is 3-regular, each vertex of S is incident to at most three such edges. If each odd component H of $G - S$ is incident to at least three such edges, then $3o(G - S) \leq 3|S|$ and hence $o(G - S) \leq |S|$, as desired.

Let m be the number of edges from S to H. The sum of the vertex degrees in H is $3n(H) - m$. Since H is a graph, the sum of its vertex degrees must be even. Since $n(H)$ is odd, we conclude that m must also be odd. Since G has no cut-edge, m cannot equal 1. We conclude that there are at least three edges from S to H, as desired. ∎

Proof by contradiction would also be natural here. Assuming $o(G - S) > |S|$ also leads to $o(G - S) \leq |S|$, so we rewrite the proof directly. Corollary 3.3.8 is best possible; the Petersen graph satisfies the hypothesis but does not have *two* edge-disjoint 1-factors (Petersen [1898]).