25 Matchings in Bipartite Graphs

Many real-world problems can be modeled as finding matchings in an undirected graph. For an undirected graph $G = (V, E)$, a ***matching*** is a subset of edges $M \subseteq E$ such that every vertex in V has at most one incident edge in M.

For example, consider the following scenario. You have one or more positions to fill and several candidates to interview. According to your schedule, you are able to interview candidates at certain time slots. You ask the candidates to indicate the subsets of time slots at which they are available. How can you schedule the interviews so that each time slot has at most one candidate scheduled, while maximizing the number of candidates that you can interview? You can model this scenario as a matching problem on a bipartite graph in which each vertex represents either a candidate or a time slot, with an edge between a candidate and a time slot if the candidate is available then. If an edge is included in the matching, that means you are scheduling a particular candidate for a particular time slot. Your goal is to find a ***maximum matching***: a matching of maximum cardinality. One of the authors of this book was faced with exactly this situation when hiring teaching assistants for a large class. He used the Hopcroft-Karp algorithm in Section 25.1 to schedule the interviews.

Another application of matching is the U.S. National Resident Matching Program, in which medical students are matched to hospitals where they will be stationed as medical residents. Each student ranks the hospitals by preference, and each hospital ranks the students. The goal is to assign students to hospitals so that there is never a student and a hospital that both have regrets because the student was not assigned to the hospital, yet each ranked the other higher than who or where they were assigned. This scenario is perhaps the best-known real-world example of the "stable-marriage problem," which Section 25.2 examines.

Yet another instance where matching comes into play occurs when workers must be assigned to tasks in order to maximize the overall effectiveness of the assignment. For each worker and each task, the worker has some quantified effectiveness for that task. Assuming that there are equal numbers of workers and tasks, the goal is to find a matching with the maximum total effectiveness. Such a situation is an example of an assignment problem, which Section 25.3 shows how to solve.

The algorithms in this chapter find matchings in ***bipartite*** graphs. As in Section 24.3, the input is an undirected graph $G = (V, E)$, where $V = L \cup R$, the vertex sets L and R are disjoint, and every edge in E is incident on one vertex in L and one vertex in R. A matching, therefore, matches vertices in L with vertices in R. In some applications, the sets L and R have equal cardinality, and in other applications they need not be the same size.

An undirected graph need not be bipartite for the concept of matching to apply. Matching in general undirected graphs has applications in areas such as scheduling and computational chemistry. It models problems in which you want to pair up entities, represented by vertices. Two vertices are adjacent if they represent compatible entities, and you need to find a large set of compatible pairs. Maximum-matching and maximum-weight matching problems on general graphs can be solved by polynomial-time algorithms whose running times are similar to those for bipartite matching, but the algorithms are significantly more complicated. Exercise 25.2-5 discusses the general version of the stable-marriage problem, known as the "stable-roommates problem." Although matching applies to general undirected graphs, this chapter deals only with bipartite graphs.

25.1 Maximum bipartite matching (revisited)

Section 24.3 demonstrated one way to find a maximum matching in a bipartite graph, by finding a maximum flow. This section provides a more efficient method, the Hopcroft-Karp algorithm, which runs in $O(\sqrt{V} E)$ time. Figure 25.1(a) shows a matching in an undirected bipartite graph. A vertex that has an incident edge in matching M is ***matched*** under M, and otherwise, it is ***unmatched***. A ***maximal matching*** is a matching M to which no other edges can be added, that is, for every edge $e \in E - M$, the edge set $M \cup \{e\}$ fails to be a matching. A maximum matching is always maximal, but the reverse does not always hold.

Many algorithms to find maximum matchings, the Hopcroft-Karp algorithm included, work by incrementally increasing the size of a matching. Given a matching M in an undirected graph $G = (V, E)$, an ***M-alternating path*** is a simple path whose edges alternate between being in M and being in $E - M$. Figure 25.1(b) depicts an ***M-augmenting path*** (sometimes called an augmenting path with respect to M): an M-alternating path whose first and last edges belong to $E - M$. Since an M-augmenting path contains one more edge in $E - M$ than in M, it must consist of an odd number of edges.

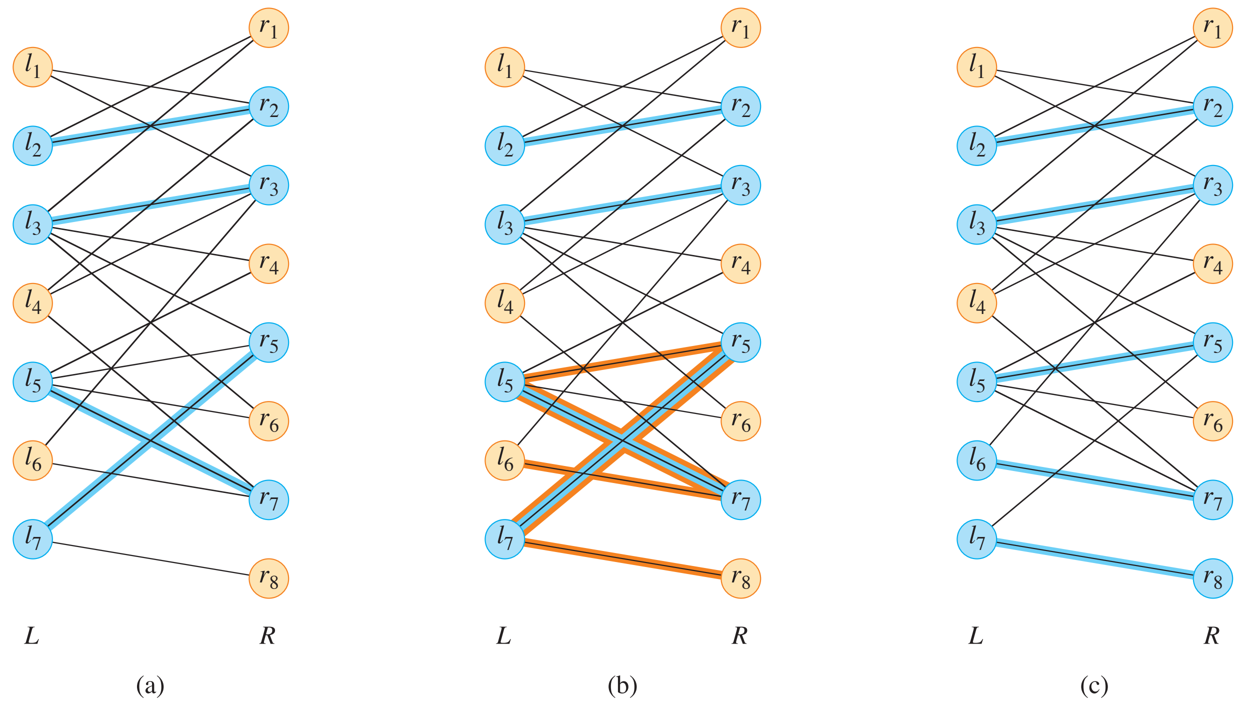


Figure 25.1 A bipartite graph, where $V = L \cup R$, $L = \{l_1, l_2, \ldots, l_7\}$, and $R = \{r_1, r_2, \ldots, r_8\}$. **(a)** A matching M with cardinality 4, highlighted in blue. Matched vertices are blue, and unmatched vertices are tan. **(b)** The five edges highlighted in orange form an M-augmenting path P going between vertices l_6 and r_8. **(c)** The set of edges $M' = M \oplus P$ highlighted in blue is a matching containing one more edge than M and adding l_6 and r_8 to the matched vertices. This matching is not a maximum matching (see Exercise 25.1-1).

Figure 25.1(c) demonstrates the following lemma, which shows that by removing from matching M the edges in an M-augmenting path that belong to M and adding to M the edges in the M-augmenting path that are not in M, the result is a new matching with one more edge than M. Since a matching is a set of edges, the lemma relies on the notion of the ***symmetric difference*** of two sets: $X \oplus Y = (X - Y) \cup (Y - X)$, that is, the elements that belong to X or Y, but not both. Alternatively, you can think of $X \oplus Y$ as $(X \cup Y) - (X \cap Y)$. The operator $\oplus$ is commutative and associative. Furthermore, $X \oplus X = \emptyset$ and $X \oplus \emptyset = \emptyset \oplus X = X$ for any set X, so that the empty set is the identity for $\oplus$.

Lemma 25.1
Let M be a matching in any undirected graph $G = (V, E)$, and let P be an M-augmenting path. Then the set of edges $M' = M \oplus P$ is also a matching in G with $|M'| = |M| + 1$.

Proof Let P contain q edges, so that $\lceil q/2 \rceil$ edges belong to $E - M$ and $\lfloor q/2 \rfloor$ edges belong to M, and let these q edges be $(v_1, v_2), (v_2, v_3), \ldots, (v_q, v_{q+1})$. Because P is an M-augmenting path, vertices v_1 and v_{q+1} are unmatched under M and all other vertices in P are matched. Edges $(v_1, v_2), (v_3, v_4), \ldots, (v_q, v_{q+1})$ belong to $E - M$, and edges $(v_2, v_3), (v_4, v_5), \ldots, (v_{q-1}, v_q)$ belong to M. The symmetric difference $M' = M \oplus P$ reverses these roles, so that edges $(v_1, v_2), (v_3, v_4), \ldots, (v_q, v_{q+1})$ belong to M' and $(v_2, v_3), (v_4, v_5), \ldots, (v_{q-1}, v_q)$ belong to $E - M'$. Each vertex $v_1, v_2, \ldots, v_q, v_{q+1}$ is matched under M', which gains one additional edge relative to M, and no other vertices or edges in G are affected by the change from M to M'. Hence, M' is a matching in G, and $|M'| = |M| + 1$. ∎

Since taking the symmetric difference of a matching M with an M-augmenting path increases the size of the matching by 1, the following corollary shows that taking the symmetric difference of M with k vertex-disjoint M-augmenting paths increases the size of the matching by k.

Corollary 25.2
Let M be a matching in any undirected graph $G = (V, E)$ and $P_1, P_2, \ldots, P_k$ be vertex-disjoint M-augmenting paths. Then the set of edges $M' = M \oplus (P_1 \cup P_2 \cup \cdots \cup P_k)$ is a matching in G with $|M'| = |M| + k$.

Proof Since the M-augmenting paths $P_1, P_2, \ldots, P_k$ are vertex-disjoint, we have that $P_1 \cup P_2 \cup \cdots \cup P_k = P_1 \oplus P_2 \oplus \cdots \oplus P_k$. Because the operator $\oplus$ is associative, we have

$$\begin{aligned} M \oplus (P_1 \cup P_2 \cup \cdots \cup P_k) &= M \oplus (P_1 \oplus P_2 \oplus \cdots \oplus P_k) \\ &= (\cdots((M \oplus P_1) \oplus P_2) \oplus \cdots \oplus P_{k-1}) \oplus P_k . \end{aligned}$$

A simple induction on i using Lemma 25.1 shows that $M \oplus (P_1 \cup P_2 \cup \cdots \cup P_{i-1})$ is a matching in G containing $|M| + i - 1$ edges and that path P_i is an augmenting path with respect to $M \oplus (P_1 \cup P_2 \cup \cdots \cup P_{i-1})$. Each of these augmenting paths increases the size of the matching by 1, and so $|M'| = |M| + k$. ∎

As the Hopcroft-Karp algorithm goes from matching to matching, it will be useful to consider the symmetric difference between two matchings.

Lemma 25.3
Let M and M^* be matchings in graph $G = (V, E)$, and consider the graph $G' = (V, E')$, where $E' = M \oplus M^*$. Then, G' is a disjoint union of simple paths, simple cycles, and/or isolated vertices. The edges in each such simple path or simple cycle

alternate between M and M^*. If $|M^*| > |M|$, then G' contains at least $|M^*| - |M|$ vertex-disjoint M-augmenting paths.

Proof Each vertex in G' has degree 0, 1, or 2, since at most two edges of E' can be incident on a vertex: at most one edge from M and at most one edge from M^*. Therefore, each connected component of G' is either a singleton vertex, an even-length simple cycle with edges alternately in M and M^*, or a simple path with edges alternately in M and M^*. Since

$$\begin{aligned} E' &= M \oplus M^* \\ &= (M \cup M^*) - (M \cap M^*) \end{aligned}$$

and $|M^*| > |M|$, the edge set E' must contain $|M^*| - |M|$ more edges from M^* than from M. Because each cycle in G' has an even number of edges drawn alternately from M and M^*, each cycle has an equal number of edges from M and M^*. Therefore, the simple paths in G' account for there being $|M^*| - |M|$ more edges from M^* than M. Each path containing a different number of edges from M and M^* either starts and ends with edges from M, containing one more edge from M than from M^*, or starts and ends with edges from M^*, containing one more edge from M^* than from M. Because E' contains $|M^*| - |M|$ more edges from M^* than from M, there are at least $|M^*| - |M|$ paths of the latter type, and each one is an M-augmenting path. Because each vertex has at most two incident edges from E', these paths must be vertex-disjoint. ■

If an algorithm finds a maximum matching by incrementally increasing the size of the matching, how does it determine when to stop? The following corollary gives the answer: when there are no augmenting paths.

Corollary 25.4
Matching M in graph $G = (V, E)$ is a maximum matching if and only if G contains no M-augmenting path.

Proof We prove the contrapositive of both directions of the lemma statement. The contrapositive of the forward direction is straightforward. If there is an M-augmenting path P in G, then by Lemma 25.1, the matching $M \oplus P$ contains one more edge than M, meaning that M could not be a maximum matching.

To show the contrapositive of the backward direction—if M is not a maximum matching, then G contains an M-augmenting path—let M^* be a maximum matching in Lemma 25.3, so that $|M^*| > |M|$. Then G contains at least $|M^*| - |M| > 0$ vertex-disjoint M-augmenting paths. ■

We already have learned enough to create a maximum-matching algorithm that runs in $O(VE)$ time. Start with the matching M empty. Then repeatedly run a variant of either breadth-first search or depth-first search from an unmatched vertex that takes alternating paths until you find another unmatched vertex. Use the resulting M-augmenting path to increase the size of M by 1.

The Hopcroft-Karp algorithm

The Hopcroft-Karp algorithm improves the running time to $O(\sqrt{V}E)$. The procedure HOPCROFT-KARP is given an undirected bipartite graph, and it uses Corollary 25.2 to repeatedly increase the size of the matching M it finds. Corollary 25.4 proves that the algorithm is correct, since it terminates once there are no M-augmenting paths. It remains to show that the algorithm does run in $O(\sqrt{V}E)$ time. We'll see that the **repeat** loop of lines 2–5 iterates $O(\sqrt{V})$ times and how to implement line 3 so that it runs in $O(E)$ time in each iteration.

```
HOPCROFT-KARP(G)
1  $M = \emptyset$
2  repeat
3      let $\mathcal{P} = \{P_1, P_2, \ldots, P_k\}$ be a maximal set of vertex-disjoint
           shortest $M$-augmenting paths
4      $M = M \oplus (P_1 \cup P_2 \cup \cdots \cup P_k)$
5  until $\mathcal{P} == \emptyset$
6  return $M$
```

Let's first see how to find a maximal set of vertex-disjoint shortest M-augmenting paths in $O(E)$ time. There are three phases. The first phase forms a directed version G_M of the undirected bipartite graph G. The second phase creates a directed acyclic graph H from G_M via a variant of breadth-first search. The third phase finds a maximal set of vertex-disjoint shortest M-augmenting paths by running a variant of depth-first search on the transpose H^{T} of H. (Recall that the transpose of a directed graph reverses the direction of each edge. Since H is acyclic, so is H^{T}.)

Given a matching M, you can think of an M-augmenting path P as starting at an unmatched vertex in L, traversing an odd number of edges, and ending at an unmatched vertex in R. The edges in P traversed from L to R must belong to $E - M$, and the edges in P traversed from R to L must belong to M. The first phase, therefore, creates the directed graph G_M by directing the edges accordingly: $G_M = (V, E_M)$, where

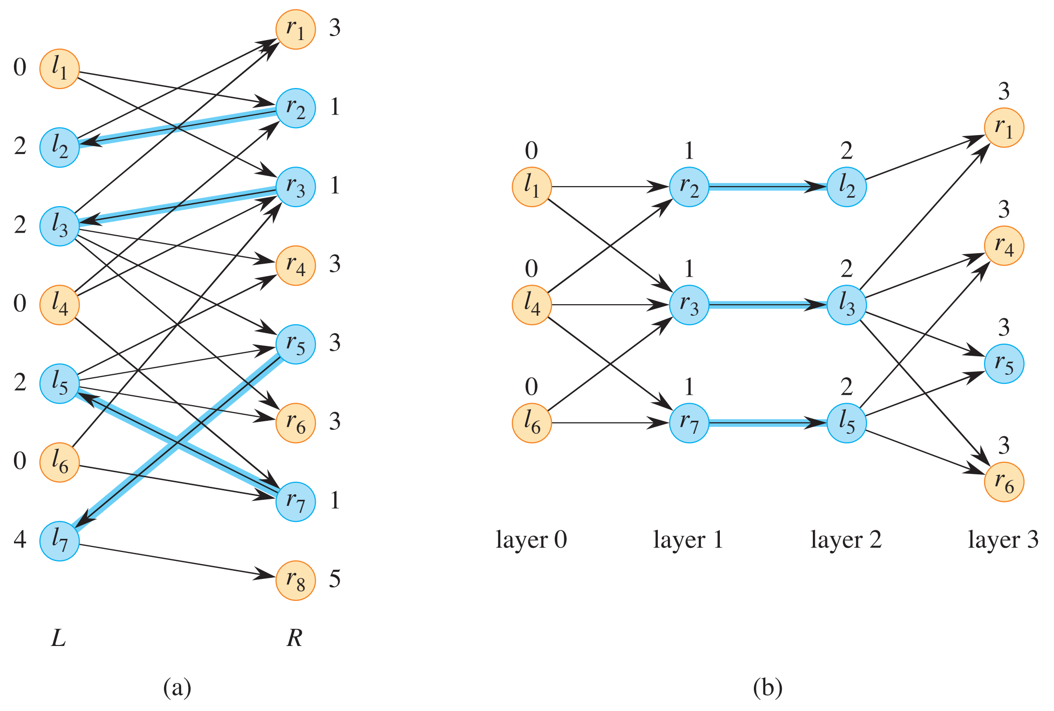


Figure 25.2 **(a)** The directed graph G_M created in the first phase for the undirected bipartite graph G and matching M in Figure 25.1(a). Breadth-first distances from any unmatched vertex in L appear next to each vertex. **(b)** The dag H created from G_M in the second phase. Because the smallest distance to an unmatched vertex in R is 3, vertices l_7 and r_8, with distances greater than 3, are not in H.

$$\begin{aligned} E_M = {} & \{(l,r) : l \in L, r \in R, \text{ and } (l,r) \in E - M\} && \text{(edges from } L \text{ to } R) \\ & \cup \{(r,l) : r \in R, l \in L, \text{ and } (l,r) \in M\} && \text{(edges from } R \text{ to } L) . \end{aligned}$$

Figure 25.2(a) shows the graph G_M for the graph G and matching M in Figure 25.1(a).

The dag $H = (V_H, E_H)$ created by the second phase has layers of vertices. Figure 25.2(b) shows the dag H corresponding to the directed graph G_M in part (a) of the figure. Each layer contains only vertices from L or only vertices from R, alternating from layer to layer. The layer that a vertex resides in is given by that vertex's minimum breadth-first distance in G_M from any unmatched vertex in L. Vertices in L appear in even-numbered layers, and vertices in R appear in odd-numbered layers. Let q denote the smallest distance in G_M of any unmatched vertex in R. Then, the last layer in H contains the vertices in R with distance q. Vertices whose distance exceeds q do not appear in V_H. (The graph H in Figure 25.2(b) omits vertices l_7 and r_8 because their distances from any unmatched vertex in L exceed $q = 3$.) The edges in E_H form a subset of E_M:

$$E_H = \{(l,r) \in E_M : r.d \leq q \text{ and } r.d = l.d + 1\} \cup \{(r,l) \in E_M : l.d \leq q\} ,$$

where the attribute d of a vertex gives the vertex's breadth-first distance in G_M from any unmatched vertex in L. Edges that do not go between two consecutive layers are omitted from E_H.

To determine the breadth-first distances of vertices, run breadth-first search on the graph G_M, but starting from all the unmatched vertices in L. (In the BFS procedure on page 556, replace the root vertex s by the set of unmatched vertices in L.) The predecessor attributes π computed by the BFS procedure are not needed here, since H is a dag and not necessarily a tree.

Every path in H from a vertex in layer 0 to an unmatched vertex in layer q corresponds to a shortest M-augmenting path in the original bipartite graph G. Just use the undirected versions of the directed edges in H. Moreover, every shortest M-augmenting path in G is present in H.

The third phase identifies a maximal set of vertex-disjoint shortest M-augmenting paths. As Figure 25.3 shows, it starts by creating the transpose H^{T} of H. Then, for each unmatched vertex r in layer q, it performs a depth-first search starting from r until it either reaches a vertex in layer 0 or has exhausted all possible paths without reaching a vertex in layer 0. Instead of maintaining discovery and finish times, the depth-first search just needs to keep track of the predecessor attributes π in the depth-first tree of each search. Upon reaching a vertex in layer 0, tracing back along the predecessors identifies an M-augmenting path. Each vertex is searched from only when it is first discovered in any search. If the search from a vertex r in layer q cannot find a path of undiscovered vertices to an undiscovered vertex in layer 0, then no M-augmenting path including r goes into the maximal set.

Figure 25.3 shows the result of the third phase. The first depth-first search starts from vertex r_1. It identifies the M-augmenting path $\langle (r_1, l_3), (l_3, r_3), (r_3, l_1) \rangle$, which is highlighted in orange, and discovers vertices r_1, l_3, r_3, and l_1. The next depth-first search starts from vertex r_4. This search first examines the edge (r_4, l_3), but because l_3 was already discovered, it backtracks and examines edge (r_4, l_5). From there, it continues and identifies the M-augmenting path $\langle (r_4, l_5), (l_5, r_7), (r_7, l_6) \rangle$, which is highlighted in yellow, and discovers vertices r_4, l_5, r_7, and l_6. The depth-first search from vertex r_6 gets stuck at vertices l_3 and l_5, which have already been discovered, and so this search fails to find a path of undiscovered vertices to a vertex in layer 0. There is no depth-first search from vertex r_5 because it is matched, and depth-first searches start from unmatched vertices. Therefore, the maximal set of vertex-disjoint shortest M-augmenting paths found contains just the two M-augmenting paths $\langle\langle r_1, l_3), (l_3, r_3), (r_3, l_1) \rangle$ and $\langle (r_4, l_5), (l_5, r_7), (r_7, l_6) \rangle$.

You might have noticed that in this example, this maximal set of two vertex-disjoint shortest M-augmenting paths is not a maximum set. The graph contains three vertex-disjoint shortest M-augmenting paths: $\langle (r_1, l_2), (l_2, r_2), (r_2, l_1) \rangle$, $\langle (r_4, l_3), (l_3, r_3), (r_3, l_4) \rangle$, and $\langle (r_6, l_5), (l_5, r_7), (r_7, l_6) \rangle$. No matter: the algorithm

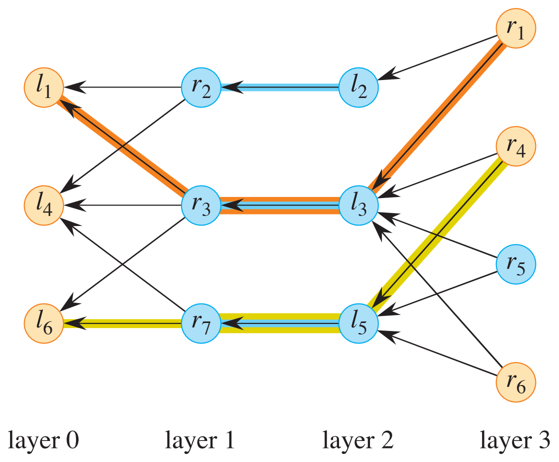


Figure 25.3 The transpose H^T of the dag H created in the third phase. The first depth-first search, starting from vertex r_1, identifies the M-augmenting path $\langle(r_1, l_3), (l_3, r_3), (r_3, l_1)\rangle$ highlighted in orange, and it discovers vertices r_1, l_3, r_3, l_1. The second depth-first search, starting from vertex r_4, identifies the M-augmenting path $\langle(r_4, l_5), (l_5, r_7), (r_7, l_6)\rangle$ highlighted in yellow, discovering vertices r_4, l_5, r_7, l_6.

requires the set of vertex-disjoint shortest M-augmenting paths found in line 3 of HOPCROFT-KARP to be only maximal, not necessarily maximum.

It remains to show that all three phases of line 3 take $O(E)$ time. We assume that in the original bipartite graph G, each vertex has at least one incident edge so that $|V| = O(E)$, which in turn implies that $|V| + |E| = O(E)$. The first phase creates the directed graph G_M by simply directing each edge of G, so that $|V_M| = |V|$ and $|E_M| = |E|$. The second phase performs a breadth-first search on G_M, taking $O(V_M + E_M) = O(E_M) = O(E)$ time. In fact, it can stop once the first distance in the queue within the breadth-first search exceeds the shortest distance q to an unmatched vertex in R. The dag H has $|V_H| \le |V_M|$ and $|E_H| \le |E_M|$, so that it takes $O(V_H + E_H) = O(E)$ time to construct. Finally, the third phase performs depth-first searches from the unmatched vertices in layer q. Once a vertex is discovered, it is not searched from again, and so the analysis of depth-first search from Section 20.3 applies here: $O(V_H + E_H) = O(E)$. Hence, all three phases take just $O(E)$ time.

Once the maximal set of vertex-disjoint shortest M-augmenting paths have been found in line 3, updating the matching in line 4 takes $O(E)$ time, as it is just a matter of going through the edges of the M-augmenting paths and adding edges to and removing edges from the matching M. Thus, each iteration of the **repeat** loop of lines 2–5 can run in $O(E)$ time.

It remains to show that the **repeat** loop iterates $O(\sqrt{V})$ times. We start with the following lemma, which shows that after each iteration of the **repeat** loop, the length of an augmenting path increases.

Lemma 25.5
Let $G = (V, E)$ be an undirected bipartite graph with matching M, and let q be the length of a shortest M-augmenting path. Let $\mathcal{P} = \{P_1, P_2, \ldots, P_k\}$ be a maximal set of vertex-disjoint M-augmenting paths of length q. Let $M' = M \oplus (P_1 \cup P_2 \cup \cdots \cup P_k)$, and suppose that P is a shortest M'-augmenting path. Then P has more than q edges.

Proof We consider separately the cases in which P is vertex-disjoint from the augmenting paths in $\mathcal{P}$ and in which it is not vertex-disjoint.

First, assume that P is vertex-disjoint from the augmenting paths in $\mathcal{P}$. Then, P contains edges that are in M but are not in any of $P_1, P_2, \ldots, P_k$, so that P is also an M-augmenting path. Since P is disjoint from $P_1, P_2, \ldots, P_k$ but is also an M-augmenting path, and since $\mathcal{P}$ is a maximal set of shortest M-augmenting paths, P must be longer than any of the augmenting paths in $\mathcal{P}$, each of which has length q. Therefore, P has more than q edges.

Now, assume that P visits at least one vertex from the M-augmenting paths in $\mathcal{P}$. By Corollary 25.2, M' is a matching in G with $|M'| = |M| + k$. Since P is an M'-augmenting path, by Lemma 25.1, $M' \oplus P$ is a matching with $|M' \oplus P| = |M'| + 1 = |M| + k + 1$. Now let $A = M \oplus M' \oplus P$. We claim that $A = (P_1 \cup P_2 \cup \cdots \cup P_k) \oplus P$:

$$\begin{aligned}
A &= M \oplus M' \oplus P \\
&= M \oplus (M \oplus (P_1 \cup P_2 \cup \cdots \cup P_k)) \oplus P \\
&= (M \oplus M) \oplus (P_1 \cup P_2 \cup \cdots \cup P_k) \oplus P && \text{(associativity of } \oplus) \\
&= \emptyset \oplus (P_1 \cup P_2 \cup \cdots \cup P_k) \oplus P && (X \oplus X = \emptyset \text{ for all } X) \\
&= (P_1 \cup P_2 \cup \cdots \cup P_k) \oplus P && (\emptyset \oplus X = X \text{ for all } X)\,.
\end{aligned}$$

Lemma 25.3 with $M^* = M' \oplus P$ gives that A contains at least $|M' \oplus P| - |M| = k + 1$ vertex-disjoint M-augmenting paths. Since each such M-augmenting path has at least q edges, we have $|A| \geq (k+1)q = kq + q$.

Now we claim that P shares at least one edge with some M-augmenting path in $\mathcal{P}$. Under the matching M', every vertex in each M-augmenting path in $\mathcal{P}$ is matched. (Only the first and last vertex in each M-augmenting path P_i is unmatched under M, and under $M \oplus P_i$, all vertices in P_i are matched. Because the M-augmenting paths in $\mathcal{P}$ are vertex-disjoint, no other path in $\mathcal{P}$ can affect whether the vertices in P_i are matched. That is, the vertices in P_i are matched under $(M \oplus P_i) \oplus P_j$ if and only if they are matched under $M \oplus P_i$, for any other path $P_j \in \mathcal{P}$.) Suppose that P shares a vertex v with some path $P_i \in \mathcal{P}$. Vertex v cannot be an endpoint of P, because the endpoints of P are unmatched under M'. Therefore, v has an incident edge in P that belongs to M'. Since any vertex has at most one incident edge in a matching, this edge must also belong to P_i, thus proving the claim.

Because $A = (P_1 \cup P_2 \cup \cdots \cup P_k) \oplus P$ and P shares at least one edge with some $P_i \in \mathcal{P}$, we have that $|A| < |P_1 \cup P_2 \cup \cdots \cup P_k| + |P|$. Thus, we have

$$\begin{aligned} kq + q &\le |A| \\ &< |P_1 \cup P_2 \cup \cdots \cup P_k| + |P| \\ &= kq + |P| \ , \end{aligned}$$

so that $q < |P|$. We conclude that P contains more than q edges. ■

The next lemma bounds the size of a maximum matching, based on the length of a shortest augmenting path.

Lemma 25.6
Let M be a matching in graph $G = (V, E)$, and let a shortest M-augmenting path in G contain q edges. Then the size of a maximum matching in G is at most $|M| + |V|/(q+1)$.

Proof Let M^* be a maximum matching in G. By Lemma 25.3, G contains at least $|M^*| - |M|$ vertex-disjoint M-augmenting paths. Each of these paths contains at least q edges, and hence at least $q+1$ vertices. Because these paths are vertex-disjoint, we have $(|M^*| - |M|)(q+1) \le |V|$, so that $|M^*| \le |M| + |V|/(q+1)$. ■

The final lemma bounds the number of iterations of the **repeat** loop of lines 2–5.

Lemma 25.7
When the HOPCROFT-KARP procedure runs on an undirected bipartite graph $G = (V, E)$, the **repeat** loop of lines 2–5 iterates $O(\sqrt{V})$ times.

Proof By Lemma 25.5, the length q of the shortest M-augmenting paths found in line 3 increases from iteration to iteration. After $\lceil \sqrt{|V|} \rceil$ iterations, therefore, we must have $q \ge \lceil \sqrt{|V|} \rceil$. Consider the situation after the first time line 4 executes with M-augmenting paths whose length is at least $\lceil \sqrt{|V|} \rceil$. Since the size of a matching increases by at least one edge per iteration, Lemma 25.6 implies that the number of additional iterations before achieving a maximum matching is at most

$$\frac{|V|}{\lceil \sqrt{|V|} \rceil + 1} < \frac{|V|}{\sqrt{|V|}} = \sqrt{|V|} \, .$$

25.1-6

In a ***d-regular*** graph, every vertex has degree d. If $G = (V, E)$ is bipartite with vertex partition $V = L \cup R$ and also d-regular, then $|L| = |R|$. Use Hall's theorem (see Exercise 25.1-5) to prove that every d-regular bipartite graph contains a perfect matching. Then use that result to prove that every d-regular bipartite graph contains d disjoint perfect matchings.

25.2 The stable-marriage problem

In Section 25.1, the goal was to find a maximum matching in an undirected bipartite graph. If you know that the graph $G = (V, E)$ with vertex partition $V = L \cup R$ is a ***complete bipartite graph***[1]—containing an edge from every vertex in L to every vertex in R—then you can find a maximum matching by a simple greedy algorithm.

When a graph can have several matchings, you might want to decide which matchings are most desirable. In Section 25.3, we'll add weights to the edges and find a matching of maximum weight. In this section, we will instead add some information to each vertex in a complete bipartite graph: a ranking of the vertices in the other side. That is, each vertex in L has an ordered list of all the vertices in R, and vice-versa. To keep things simple, let's assume that L and R each contain n vertices. The goal here is to match each vertex in L with a vertex in R in a "stable" way.

This problem derives its name, the ***stable-marriage problem***, from the notion of heterosexual marriage, viewing L as a set of women and R as a set of men.[2] Each woman ranks all the men in terms of desirability, and each man does the same with all the women. The goal is to pair up women and men (a matching) so that if a woman and a man are not matched to each other, then at least one of them prefers their assigned partner.

If a woman and a man are not matched to each other but each prefers the other over their assigned partner, they form a ***blocking pair***. A blocking pair has incentive to opt out of the assigned pairing and get together on their own. If that were to occur, then this pair would block the matching from being "stable." A ***stable***

[1] The definition of a complete bipartite graph differs from the definition of complete graph given on page 1167 because in a bipartite graph, there are no edges between vertices in L and no edges between vertices in R.

[2] Although marriage norms are changing, it's traditional to view the stable-marriage problem through the lens of heterosexual marriage.

matching, therefore, is a matching that has no blocking pair. If there is a blocking pair, then the matching is ***unstable***.

Let's look at an example with four women—Wanda, Emma, Lacey, and Karen—and four men—Oscar, Davis, Brent, and Hank—having the following preferences:

Wanda:	Brent, Hank, Oscar, Davis
Emma:	Davis, Hank, Oscar, Brent
Lacey:	Brent, Davis, Hank, Oscar
Karen:	Brent, Hank, Davis, Oscar

Oscar:	Wanda, Karen, Lacey, Emma
Davis:	Wanda, Lacey, Karen, Emma
Brent:	Lacey, Karen, Wanda, Emma
Hank:	Lacey, Wanda, Emma, Karen

A stable matching comprises the following pairs:

Lacey and Brent
Wanda and Hank
Karen and Davis
Emma and Oscar

You can verify that this matching has no blocking pair. For example, even though Karen prefers Brent and Hank to her partner Davis, Brent prefers his partner Lacey to Karen, and Hank prefers his partner Wanda to Karen, so that neither Karen and Brent nor Karen and Hank form a blocking pair. In fact, this stable matching is unique. Suppose instead that the last two pairs were

Emma and Davis
Karen and Oscar

Then Karen and Davis would be a blocking pair, because they were not paired together, Karen prefers Davis to Oscar, and Davis prefers Karen to Emma. Therefore, this matching is not stable.

Stable matchings need not be unique. For example, suppose that there are three women—Monica, Phoebe, and Rachel—and three men—Chandler, Joey, and Ross—with these preferences:

Monica:	Chandler, Joey, Ross
Phoebe:	Joey, Ross, Chandler
Rachel:	Ross, Chandler, Joey

Chandler:	Phoebe, Rachel, Monica
Joey:	Rachel, Monica, Phoebe
Ross:	Monica, Phoebe, Rachel

In this case, there are three stable matchings:

Matching 1	Matching 2	Matching 3
Monica and Chandler	Phoebe and Chandler	Rachel and Chandler
Phoebe and Joey	Rachel and Joey	Monica and Joey
Rachel and Ross	Monica and Ross	Phoebe and Ross

In matching 1, all women get their first choice and all men get their last choice. Matching 2 is the opposite, with all men getting their first choice and all women getting their last choice. When all the women or all the men get their first choice, there plainly cannot be a blocking pair. In matching 3, everyone gets their second choice. You can verify that there are no blocking pairs.

You might wonder whether it is always possible to come up with a stable matching no matter what rankings each participant provides. The answer is yes. (Exercise 25.2-3 asks you to show that even in the scenario of the National Resident Matching Program, where each hospital takes on multiple students, it is always possible to devise a stable assignment.) A simple algorithm known as the Gale-Shapley algorithm always finds a stable matching. The algorithm has two variants, which mirror each other: "woman-oriented" and "man-oriented." Let's examine the woman-oriented version. Each participant is either "free" or "engaged." Everyone starts out free. Engagements occur when a free woman proposes to a man. When a man is first proposed to, he goes from free to engaged, and he always stays engaged, though not necessarily to the same woman. If an engaged man receives a proposal from a woman whom he prefers to the woman he's currently engaged to, that engagement is broken, the woman to whom he had been engaged becomes free, and the man and the woman whom he prefers become engaged. Each woman proposes to the men in her preference list, in order, until the last time she becomes engaged. When a woman is engaged, she temporarily stops proposing, but if she becomes free again, she continues down her list. Once everyone is engaged, the algorithm terminates. The procedure GALE-SHAPLEY on the next page makes this process more concrete. The procedure allows for some choice: any free woman may be selected in line 2. We'll see that the procedure produces a stable matching regardless of the order in which line 2 chooses free women. For the man-oriented version, just reverse the roles of men and women in the procedure.

Let's see how the GALE-SHAPLEY procedure executes on the example with Wanda, Emma, Lacey, Karen, Oscar, Davis, Brent, and Hank. After everyone is initialized to free, here is one possible version of what can occur in successive iterations of the **while** loop of lines 2–9:

1. Wanda proposes to Brent. Brent is free, so that Wanda and Brent become engaged and no longer free.

```
GALE-SHAPLEY(men, women, rankings)
 1  assign each woman and man as free
 2  while some woman $w$ is free
 3      let $m$ be the first man on $w$'s ranked list to whom she has not proposed
 4      if $m$ is free
 5          $w$ and $m$ become engaged to each other (and not free)
 6      elseif $m$ ranks $w$ higher than the woman $w'$ he is currently engaged to
 7          $m$ breaks the engagement to $w'$, who becomes free
 8          $w$ and $m$ become engaged to each other (and not free)
 9      else $m$ rejects $w$, with $w$ remaining free
10  return the stable matching consisting of the engaged pairs
```

2. Emma proposes to Davis. Davis is free, so that Emma and Davis become engaged and no longer free.
3. Lacey proposes to Brent. Brent is engaged to Wanda, but he prefers Lacey. Brent breaks the engagement to Wanda, who becomes free. Lacey and Brent become engaged, with Lacey no longer free.
4. Karen proposes to Brent. Brent is engaged to Lacey, whom he prefers to Karen. Brent rejects Karen, who remains free.
5. Karen proposes to Hank. Hank is free, so that Karen and Hank become engaged and no longer free.
6. Wanda proposes to Hank. Hank is engaged to Karen, but he prefers Wanda. Hank breaks the engagement with Karen, who becomes free. Wanda and Hank become engaged, with Wanda no longer free.
7. Karen proposes to Davis. Davis is engaged to Emma, but he prefers Karen. Davis breaks the engagement to Emma, who becomes free. Karen and Davis become engaged, with Karen no longer free.
8. Emma proposes to Hank. Hank is engaged to Wanda, whom he prefers to Emma. Hank rejects Emma, who remains free.
9. Emma proposes to Oscar. Oscar is free, so that Emma and Oscar become engaged and no longer free.

At this point, everyone is engaged and nobody is free, so the **while** loop terminates. The procedure returns the stable matching we saw earlier.

The following theorem shows that not only does GALE-SHAPLEY terminate, but that it always returns a stable matching, thereby proving that a stable matching always exists.

Theorem 25.9
The procedure GALE-SHAPLEY always terminates and returns a stable matching.

Proof Let's first show that the **while** loop of lines 2–9 always terminates, so that the procedure terminates. The proof is by contradiction. If the loop fails to terminate, it is because some woman remains free. In order for a woman to remain free, she must have proposed to all the men and been rejected by each one. In order for a man to reject a woman, he must be already engaged. Therefore, all the men are engaged. Once engaged, a man stays engaged (though not necessarily to the same woman). There are an equal number n of women and men, however, which means that every woman is engaged, leading to the contradiction that no women are free. We must also show that the **while** loop makes a bounded number of iterations. Since each of the n women goes through her ranking of the n men in order, possibly not reaching the end of her list, the total number of iterations is at most n^2. Therefore, the **while** loop always terminates, and the procedure returns a matching.

We need to show that there are no blocking pairs. We first observe that once a man m is engaged to a woman w, all subsequent actions for m occur in lines 6–8. Therefore, once a man is engaged, he stays engaged, and any time he breaks an engagement to a woman w, it's for a woman whom he prefers to w. Suppose that a woman w is matched with a man m, but she prefers man m'. We'll show that w and m' is not a blocking pair, because m' does not prefer w to his partner. Because w ranks m' higher than m, she must have proposed to m' before proposing to m, and m' either rejected her proposal or accepted it and later broke the engagement. If m' rejected the proposal from w, it is because he was already engaged to some woman he prefers to w. If m' accepted and later broke the engagement, he was at some point engaged to w but later accepted a proposal from a woman he prefers to w. In either case, he ultimately ends up with a partner whom he prefers to w. We conclude that even though w might prefer m' to her partner m, it is not also the case that m' prefers w to his partner. Therefore, the procedure returns a matching containing no blocking pairs. ■

Exercise 25.2-1 asks you to provide the proof of the following corollary.

Corollary 25.10
Given preference rankings for n women and n men, the Gale-Shapley algorithm can be implemented to run in $O(n^2)$ time. ■

Because line 2 can choose any free woman, you might wonder whether different choices can produce different stable matchings. The answer is no: as the following theorem shows, every execution of the GALE-SHAPLEY produces exactly the same result. Moreover, the stable matching returned is optimal for the women.

Theorem 25.11
Regardless of how women are chosen in line 2 of GALE-SHAPLEY, the procedure always returns the same stable matching, and in this stable matching, each woman has the best partner possible in any stable matching.

Proof The proof that each woman has the best partner possible in any stable matching is by contradiction. Suppose that the GALE-SHAPLEY procedure returns a stable matching M, but that there is a different stable matching M' in which some woman w prefers her partner m' to the partner m she has in M. Because w ranks m' higher than m, she must have proposed to m' before proposing to m. Then there is a woman w' whom m' prefers to w, and m' was already engaged to w' when w proposed or m' accepted the proposal from w and later broke the engagement in favor of w'. Either way, there is a moment when m' decided against w in favor of w'. Now suppose, without loss of generality, that this moment was the first time that any man rejected a partner who belongs to some stable matching.

We claim that w' cannot have a partner m'' in a stable matching whom she prefers to m'. If there were such a man m'', then in order for w' to propose to m', she would have proposed to m'' and been rejected at some point before proposing to m'. If m' accepted the proposal from w and later broke it to accept w', then since this was the first rejection in a stable matching, we get the contradiction that m'' could not have rejected w' beforehand. If m' was already engaged to w' when w proposed, then again, m'' could not have rejected w' beforehand, thus proving the claim.

Since w' does not prefer anyone to m' in a stable matching and w' is not matched with m' in M' (because m' is matched with w in M'), w' prefers m' to her partner in M'. Since w' prefers m' over her partner in M' and m' prefers w' over his partner w in M', the pair w' and m' is a blocking pair in M'. Because M' has a blocking pair, it cannot be a stable matching, thereby contradicting the assumption that there exists some stable matching in which each woman has the best partner possible other than the matching M returned by GALE-SHAPLEY.

We put no condition on the execution of the procedure, which means that all possible orders in which line 2 selects women result in the same stable matching being returned. ■

Corollary 25.12
There can be stable matchings that the GALE-SHAPLEY procedure does not return.

Proof Theorem 25.11 says that for a given set of rankings, GALE-SHAPLEY returns just one matching, no matter how it chooses women in line 2. The earlier example of three women and three men with three different stable matchings shows that there can be multiple stable matchings for a given set of rankings. A call of GALE-SHAPLEY is capable of returning only one of these stable matchings. ■

Although the GALE-SHAPLEY procedure gives the best possible outcome for the women, the following corollary shows that it also produces the worst possible outcome for the men.

Corollary 25.13
In the stable matching returned by the procedure GALE-SHAPLEY, each man has the worst partner possible in any stable matching.

Proof Let M be the matching returned by a call to GALE-SHAPLEY. Suppose that there is another stable matching M' and a man m who prefers his partner w in M to his partner w' in M'. Let the partner of w in M' be m'. By Theorem 25.11, m is the best partner that w can have in any stable matching, which means that w prefers m to m'. Since m prefers w to w', the pair w and m is a blocking pair in M', contradicting the assumption that M' is a stable matching. ■

Exercises

25.2-1
Describe how to implement the Gale-Shapley algorithm so that it runs in $O(n^2)$ time.

25.2-2
Is it possible to have an unstable matching with just two women and two men? If so, provide and justify an example. If not, argue why not.

25.2-3
The National Resident Matching Program differs from the scenario for the stable-marriage problem set out in this section in two ways. First, a hospital may be matched with more than one student, so that hospital h takes $r_h \geq 1$ students. Second, the number of students might not equal the number of hospitals. Describe how to modify the Gale-Shapley algorithm to fit the requirements of the National Resident Matching Program.

25.2-4
Prove the following property, which is known as ***weak Pareto optimality***:

> Let M be the stable matching produced by the GALE-SHAPLEY procedure, with women proposing to men. Then, for a given instance of the stable-marriage problem there is no matching—stable or unstable—such that every

woman has a partner whom she prefers to her partner in the stable matching M.

25.2-5

The ***stable-roommates problem*** is similar to the stable-marriage problem, except that the graph is a complete graph, not bipartite, with an even number of vertices. Each vertex represents a person, and each person ranks all the other people. The definitions of blocking pairs and stable matching extend in the natural way: a blocking pair comprises two people who both prefer each other to their current partner, and a matching is stable if there are no blocking pairs. For example, consider four people—Wendy, Xenia, Yolanda, and Zelda—with the following preference lists:

Wendy:	Xenia, Yolanda, Zelda
Xenia:	Wendy, Zelda, Yolanda
Yolanda:	Wendy, Zelda, Xenia
Zelda:	Xenia, Yolanda, Wendy

You can verify that the following matching is stable:

Wendy and Xenia
Yolanda and Zelda

Unlike the stable-marriage problem, the stable-roommates problem can have inputs for which no stable matching exists. Find such an input and explain why no stable matching exists.

25.3 The Hungarian algorithm for the assignment problem

Let us once again add some information to a complete bipartite graph $G = (V, E)$, where $V = L \cup R$. This time, instead of having the vertices of each side rank the vertices on the other side, we assign a weight to each edge. Again, let's assume that the vertex sets L and R each contain n vertices, so that the graph contains n^2 edges. For $l \in L$ and $r \in R$, denote the weight of edge (l, r) by $w(l, r)$, which represents the utility gained by matching vertex l with vertex r.

The goal is to find a perfect matching M^* (see Exercises 25.1-5 and 25.1-6) whose edges have the maximum total weight over all perfect matchings. That is, letting $w(M) = \sum_{(l,r)\in M} w(l, r)$ denote the total weight of the edges in matching M, we want to find a perfect matching M^* such that

$$w(M^*) = \max\{w(M) : M \text{ is a perfect matching}\} .$$

We call finding such a maximum-weight perfect matching the ***assignment problem***. A solution to the assignment problem is a perfect matching that maximizes the total utility. Like the stable-marriage problem, the assignment problem finds a matching that is "good," but with a different definition of good: maximizing total value rather than achieving stability.

Although you could enumerate all $n!$ perfect matchings to solve the assignment problem, an algorithm known as the ***Hungarian algorithm*** solves it much faster. This section will prove an $O(n^4)$ time bound, and Problem 25-2 asks you to refine the algorithm to reduce the running time to $O(n^3)$. Instead of working with the complete bipartite graph G, the Hungarian algorithm works with a subgraph of G called the "equality subgraph." The equality subgraph, which is defined below, changes over time and has the beneficial property that any perfect matching in the equality subgraph is also an optimal solution to the assignment problem.

The equality subgraph depends on assigning an attribute h to each vertex. We call h the ***label*** of a vertex, and we say that h is a ***feasible vertex labeling*** of G if

$$l.h + r.h \geq w(l, r) \text{ for all } l \in L \text{ and } r \in R \; .$$

A feasible vertex labeling always exists, such as the ***default vertex labeling*** given by

$$l.h = \max\{w(l, r) : r \in R\} \quad \text{for all } l \in L \; , \tag{25.1}$$

$$r.h = 0 \quad \text{for all } r \in R \; . \tag{25.2}$$

Given a feasible vertex labeling h, the ***equality subgraph*** $G_h = (V, E_h)$ of G consists of the same vertices as G and the subset of edges

$$E_h = \{(l, r) \in E : l.h + r.h = w(l, r)\} \; .$$

The following theorem ties together a perfect matching in an equality subgraph and an optimal solution to the assignment problem.

Theorem 25.14
Let $G = (V, E)$, where $V = L \cup R$, be a complete bipartite graph where each edge $(l, r) \in E$ has weight $w(l, r)$. Let h be a feasible vertex labeling of G and G_h be the equality subgraph of G. If G_h contains a perfect matching M^*, then M^* is an optimal solution to the assignment problem on G.

Proof If G_h contains a perfect matching M^*, then because G_h and G have the same sets of vertices, M^* is also a perfect matching in G. Because each edge of M^* belongs to G_h and each vertex has exactly one incident edge from any perfect matching, we have

$$\begin{aligned} w(M^*) &= \sum_{(l,r)\in M^*} w(l,r) \\ &= \sum_{(l,r)\in M^*} (l.h + r.h) \quad \text{(because all edges in } M^* \text{ belong to } G_h) \\ &= \sum_{l\in L} l.h + \sum_{r\in R} r.h \quad \text{(because } M^* \text{ is a perfect matching)}\ . \end{aligned}$$

Letting M be any perfect matching in G, we have

$$\begin{aligned} w(M) &= \sum_{(l,r)\in M} w(l,r) \\ &\le \sum_{(l,r)\in M} (l.h + r.h) \quad \text{(because } h \text{ is a feasible vertex labeling)} \\ &= \sum_{l\in L} l.h + \sum_{r\in R} r.h \quad \text{(because } M \text{ is a perfect matching)}\ . \end{aligned}$$

Thus, we have

$$w(M) \le \sum_{l\in L} l.h + \sum_{r\in R} r.h = w(M^*)\ , \tag{25.3}$$

so that M^* is a maximum-weight perfect matching in G. ■

The goal now becomes finding a perfect matching in an equality subgraph. Which equality subgraph? It does not matter! We have free rein to not only choose an equality subgraph, but to change which equality subgraph we choose as we go along. We just need to find *some* perfect matching in *some* equality subgraph.

To understand the equality subgraph better, consider again the proof of Theorem 25.14 and, in the second half, let M be any matching. The proof is still valid, in particular, inequality (25.3): the weight of any matching is always at most the sum of the vertex labels. If we choose any set of vertex labels that define an equality subgraph, then a maximum-cardinality matching in this equality subgraph has total value at most the sum of the vertex labels. If the set of vertex labels is the "right" one, then it will have total value equal to $w(M^*)$, and a maximum-cardinality matching in the equality subgraph is also a maximum-weight perfect matching. The Hungarian algorithm repeatedly modifies the matching and the vertex labels in order to achieve this goal.

The Hungarian algorithm starts with any feasible vertex labeling h and any matching M in the equality subgraph G_h. It repeatedly finds an M-augmenting path P in G_h and, using Lemma 25.1, updates the matching to be $M \oplus P$, thereby incrementing the size of the matching. As long as there is some equality subgraph that contains an M-augmenting path, the size of the matching can increase, until a perfect matching is achieved.

Four questions arise:

1. What initial feasible vertex labeling should the algorithm start with? Answer: the default vertex labeling given by equations (25.1) and (25.2).
2. What initial matching in G_h should the algorithm start with? Short answer: any matching, even an empty matching, but a greedy maximal matching works well.
3. If an M-augmenting path exists in G_h, how to find it? Short answer: use a variant of breadth-first search similar to the second phase of the procedure used in the Hopcroft-Karp algorithm to find a maximal set of shortest M-augmenting paths.
4. What if the search for an M-augmenting path fails? Short answer: update the feasible vertex labeling to bring in at least one new edge.

We'll elaborate on the short answers using the example that starts in Figure 25.4. Here, $L = \{l_1, l_2, \ldots, l_7\}$ and $R = \{r_1, r_2, \ldots, r_7\}$. The edge weights appear in the matrix shown in part (a), where the weight $w(l_i, r_j)$ appears in row i and column j. The feasible vertex labels, given by the default vertex labeling, appear to the left of and above the matrix. Matrix entries in red indicate edges (l_i, r_j) for which $l_i.h + r_j.h = w(l_i, r_j)$, that is, edges in the equality subgraph G_h appearing in part (b) of the figure.

Greedy maximal bipartite matching

There are several ways to implement a greedy method to find a maximal bipartite matching. The procedure GREEDY-BIPARTITE-MATCHING shows one. Edges in Figure 25.4(b) highlighted in blue indicate the initial greedy maximal matching in G_h. Exercise 25.3-2 asks you to show that the GREEDY-BIPARTITE-MATCHING procedure returns a matching that is at least half the size of a maximum matching.

```
GREEDY-BIPARTITE-MATCHING($G$)
1  $M = \emptyset$
2  for each vertex $l \in L$
3      if $l$ has an unmatched neighbor in $R$
4          choose any such unmatched neighbor $r \in R$
5          $M = M \cup \{(l, r)\}$
6  return $M$
```

$$\begin{array}{cc|ccccccc|}
 & & r_1 & r_2 & r_3 & r_4 & r_5 & r_6 & r_7 \\
 & h & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
\hline
l_1 & 10 & 4 & 10 & 10 & 10 & 2 & 9 & 3 \\
l_2 & 12 & 6 & 8 & 5 & 12 & 9 & 7 & 2 \\
l_3 & 15 & 11 & 9 & 6 & 7 & 9 & 5 & 15 \\
l_4 & 9 & 3 & 9 & 6 & 7 & 5 & 6 & 3 \\
l_5 & 6 & 2 & 6 & 5 & 3 & 2 & 4 & 2 \\
l_6 & 11 & 10 & 8 & 11 & 4 & 11 & 2 & 11 \\
l_7 & 8 & 3 & 4 & 5 & 4 & 3 & 6 & 8 \\
\hline
\end{array}$$

(a)

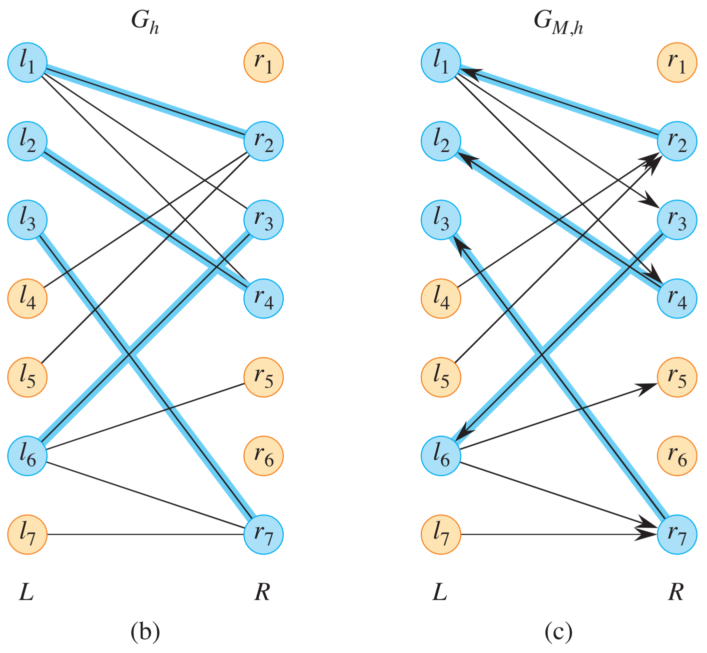


Figure 25.4 The start of the Hungarian algorithm. **(a)** The matrix of edge weights for a bipartite graph with $L = \{l_1, l_2, \ldots, l_7\}$ and $R = \{r_1, r_2, \ldots, r_7\}$. The value in row i and column j indicates $w(l_i, r_j)$. Feasible vertex labels appear above and next to the matrix. Red entries correspond to edges in the equality subgraph. **(b)** The equality subgraph G_h. Edges highlighted in blue belong to the initial greedy maximal matching M. Blue vertices are matched, and tan vertices are unmatched. **(c)** The directed equality subgraph $G_{M,h}$ created from G_h by directing edges in M from R to L and all other edges from L to R.

Finding an M-augmenting path in G_h

To find an M-augmenting path in the equality subgraph G_h with a matching M, the Hungarian algorithm first creates the ***directed equality subgraph*** $G_{M,h}$ from G_h, just as the Hopcroft-Karp algorithm creates G_M from G. As in the Hopcroft-Karp algorithm, you can think of an M-augmenting path as starting from an unmatched vertex in L, ending at an unmatched vertex in R, taking unmatched edges from L to R, and taking matched edges from R to L. Thus, $G_{M,h} = (V, E_{M,h})$, where

$$\begin{aligned} E_{M,h} = {} & \{(l,r) : l \in L, r \in R, \text{ and } (l,r) \in E_h - M\} && \text{(edges from } L \text{ to } R) \\ & \cup \{(r,l) : r \in R, l \in L, \text{ and } (l,r) \in M\} && \text{(edges from } R \text{ to } L)\,. \end{aligned}$$

Because an M-augmenting path in the directed equality subgraph $G_{M.h}$ is also an M-augmenting path in the equality subgraph G_h, it suffices to find M-augmenting paths in $G_{M.h}$. Figure 25.4(c) shows the directed equality subgraph $G_{M,h}$ corresponding to the equality subgraph G_h and matching M from part (b) of the figure.

With the directed equality subgraph $G_{M,h}$ in hand, the Hungarian algorithm searches for an M-augmenting path from any unmatched vertex in L to any unmatched vertex in R. Any exhaustive graph-search method suffices. Here, we'll use breadth-first search, starting from all the unmatched vertices in L (just as the Hopcroft-Karp algorithm does when creating the dag H), but stopping upon first discovering some unmatched vertex in R. Figure 25.5 shows the idea. To start from all the unmatched vertices in L, initialize the first-in, first-out queue with all the unmatched vertices in L, rather than just one source vertex. Unlike the dag H in the Hopcroft-Karp algorithm, here each vertex needs just one predecessor, so that the breadth-first search creates a ***breadth-first forest*** $F = (V_F, E_F)$. Each unmatched vertex in L is a root in F.

In Figure 25.5(g), the breadth-first search has found the M-augmenting path $\langle (l_4, r_2), (r_2, l_1), (l_1, r_3), (r_3, l_6), (l_6, r_5) \rangle$. Figure 25.6(a) shows the new matching created by taking the symmetric difference of the matching M in Figure 25.5(a) with this M-augmenting path.

When the search for an M-augmenting path fails

Having updated the matching M from an M-augmenting path, the Hungarian algorithm updates the directed equality subgraph $G_{M,h}$ according to the new matching and then starts a new breadth-first search from all the unmatched vertices in L. Figure 25.6 shows the start of this process, picking up from Figure 25.5.

In Figure 25.6(d), the queue contains vertices l_4 and l_3. Neither of these vertices has an edge that leaves it, however, so that once these vertices are removed from the queue, the queue becomes empty. The search terminates at this point, before discovering an unmatched vertex in R to yield an M-augmenting path. Whenever this situation occurs, the most recently discovered vertices must belong to L. Why? Whenever an unmatched vertex in R is discovered, the search has found an M-augmenting path, and when a matched vertex in R is discovered, it has an unvisited neighbor in L, which the search can then discover.

Recall that we have the freedom to work with any equality subgraph. We can change the directed equality subgraph "on the fly," as long we do not counteract the work already done. The Hungarian algorithm updates the feasible vertex labeling h to fulfill the following criteria:

1. No edge in the breadth-first forest F leaves the directed equality subgraph.
2. No edge in the matching M leaves the directed equality subgraph.
3. At least one edge (l, r), where $l \in L \cap V_F$ and $r \in R - V_F$ goes into E_h, and hence into $E_{M,h}$. Therefore, at least one vertex in R will be newly discovered.

Thus, at least one new edge enters the directed equality subgraph, and any edge that leaves the directed equality subgraph belongs to neither the matching M nor

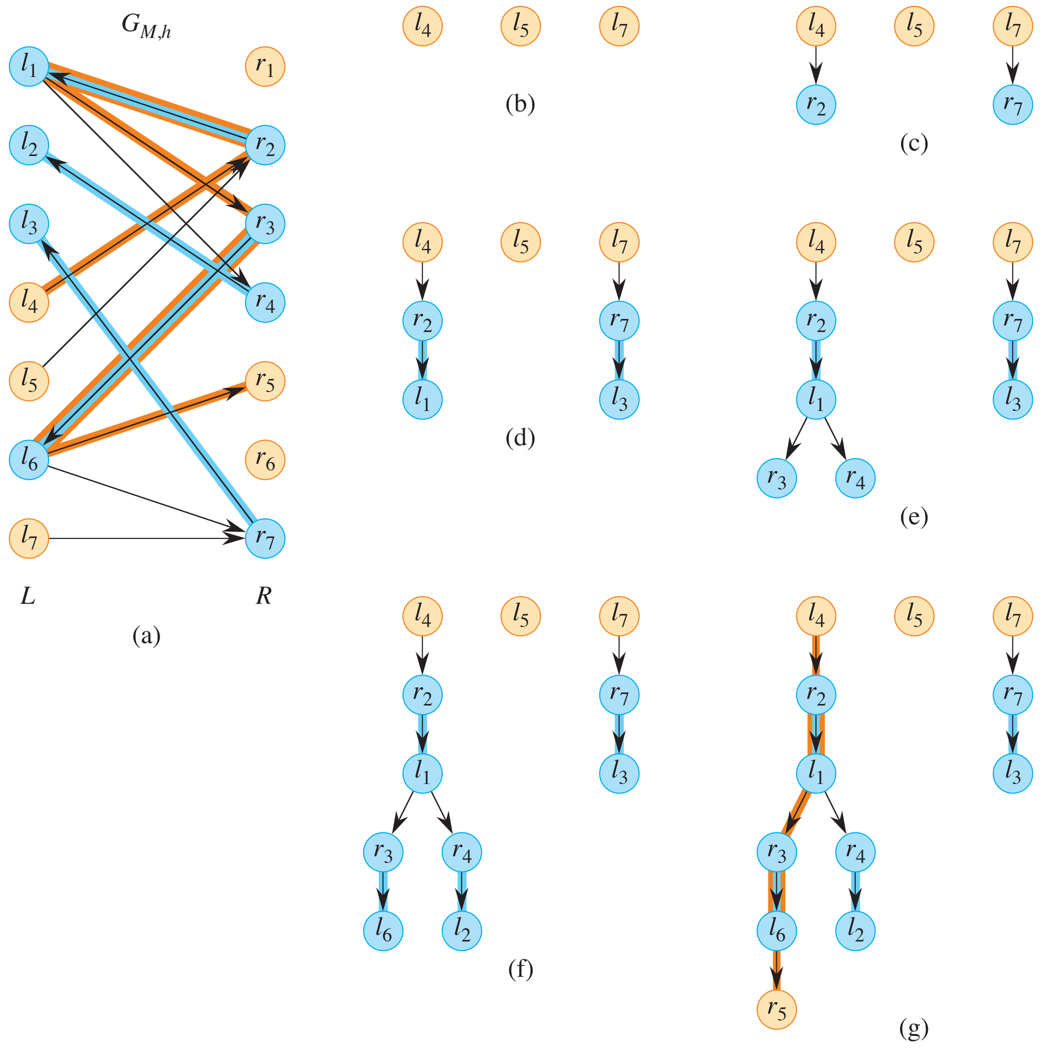

$G_{M,h}$
l_1 l_2 l_3 l_4 l_5 l_6 l_7
r_1 r_2 r_3 r_4 r_5 r_6 r_7
L
R
(a)
(b)
(c)
(d)
(e)
(f)
(g)

Figure 25.5 Finding an M-augmenting path in $G_{M,h}$ by breadth-first search. **(a)** The directed equality subgraph $G_{M,h}$ from Figure 25.4(c). **(b)–(g)** Successive versions of the breadth-first forest F, shown as the vertices at each distance from the roots—the unmatched vertices in L—are discovered. In parts (b)–(f), the layer of vertices closest to the bottom of the figure are those in the first-in, first-out queue. For example, in part (b), the queue contains the roots $\langle l_4, l_5, l_7 \rangle$, and in part (e), the queue contains $\langle r_3, r_4 \rangle$, at distance 3 from the roots. In part (g), the unmatched vertex r_5 is discovered, so the breadth-first search terminates. The path $\langle (l_4, r_2), (r_2, l_1), (l_1, r_3), (r_3, l_6), (l_6, r_5) \rangle$, highlighted in orange in parts (a) and (g), is an M-augmenting path. Taking its symmetric difference with the matching M yields a new matching with one more edge than M.

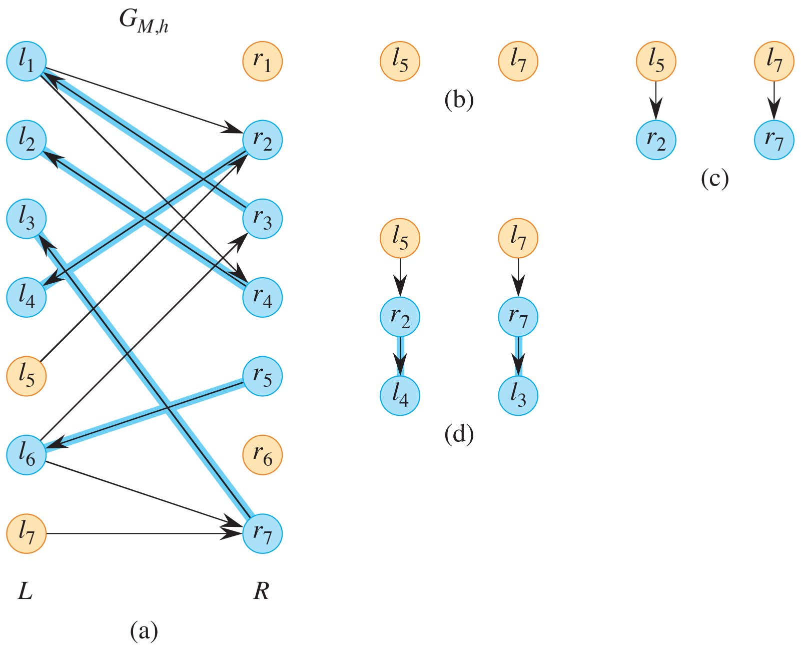


Figure 25.6 **(a)** The new matching M and the new directed equality subgraph $G_{M,h}$ after updating the matching in Figure 25.5(a) with the M-augmenting path in Figure 25.5(g). **(b)–(d)** Successive versions of the breadth-first forest F in a new breadth-first search with roots l_5 and l_7. After the vertices l_4 and l_3 in part (d) have been removed from the queue, the queue becomes empty before the search can discover an unmatched vertex in R.

the breadth-first forest F. Newly discovered vertices in R are enqueued, but their distances are not necessarily 1 greater than the distances of the most recently discovered vertices in L.

To update the feasible vertex labeling, the Hungarian algorithm first computes the value

$$\delta = \min\{l.h + r.h - w(l,r) : l \in F_L \text{ and } r \in R - F_R\} , \tag{25.4}$$

where $F_L = L \cap V_F$ and $F_R = R \cap V_F$ denote the vertices in the breadth-first forest F that belong to L and R, respectively. That is, δ is the smallest difference by which an edge incident on a vertex in F_L missed being in the current equality subgraph G_h. The Hungarian algorithm then creates a new feasible vertex labeling, say h', by subtracting δ from $l.h$ for all vertices $l \in F_L$ and adding δ to $r.h$ for all vertices $r \in F_R$:

$$v.h' = \begin{cases} v.h - \delta & \text{if } v \in F_L , \\ v.h + \delta & \text{if } v \in F_R , \\ v.h & \text{otherwise } (v \in V - V_F) . \end{cases} \tag{25.5}$$

The following lemma shows that these changes achieve the three criteria above.

Lemma 25.15
Let h be a feasible vertex labeling for the complete bipartite graph G with equality subgraph G_h, and let M be a matching for G_h and F be a breadth-first forest being constructed for the directed equality subgraph $G_{M,h}$. Then, the labeling h' in equation (25.5) is a feasible vertex labeling for G with the following properties:

1. If (u, v) is an edge in the breadth-first forest F for $G_{M,h}$, then $(u, v) \in E_{M,h'}$.
2. If (l, r) belongs to the matching M for G_h, then $(r, l) \in E_{M,h'}$.
3. There exist vertices $l \in F_L$ and $r \in R - F_R$ such that $(l, r) \notin E_{M,h}$ but $(l, r) \in E_{M,h'}$.

Proof We first show that h' is a feasible vertex labeling for G. Because h is a feasible vertex labeling, we have $l.h + r.h \geq w(l, r)$ for all $l \in L$ and $r \in R$. In order for h' to not be a feasible vertex labeling, we would need $l.h' + r.h' < l.h + r.h$ for some $l \in L$ and $r \in R$. The only way this could occur would be for some $l \in F_L$ and $r \in R - F_R$. In this instance, the amount of the decrease equals δ, so that $l.h' + r.h' = l.h - \delta + r.h$. By equation (25.4), we have that $l.h - \delta + r.h \geq w(l, r)$ for any $l \in F_L$ and $r \in R - F_R$, so that $l.h' + r.h' \geq w(l, r)$. For all other edges, we have $l.h' + r.h' \geq l.h + r.h \geq w(l, r)$. Thus, h' is a feasible vertex labeling.

Now we show that each of the three desired properties holds:

1. If $l \in F_L$ and $r \in F_R$, then we have $l.h' + r.h' = l.h + r.h$ because δ is added to the label of l and subtracted from the label of r. Therefore, if an edge belongs to F for the directed graph $G_{M,h}$, it also belongs to $G_{M,h'}$.
2. We claim that at the time the Hungarian algorithm computes the new feasible vertex labeling h', for every edge $(l, r) \in M$, we have $l \in F_L$ if and only if $r \in F_R$. To see why, consider a matched vertex r and let $(l, r) \in M$. First suppose that $r \in F_R$, so that the search discovered r and enqueued it. When r was removed from the queue, l was discovered, so $l \in F_L$. Now suppose that $r \notin F_R$, so r is undiscovered. We will show that $l \notin F_L$. The only edge in $G_{M,h}$ that enters l is (r, l), and since r is undiscovered, the search has not taken this edge; if $l \in F_L$, it is not because of the edge (r, l). The only other way that a vertex in L can be in F_L is if it is a root of the search, but only unmatched vertices in L are roots and l is matched. Thus, $l \notin F_L$, and the claim is proved.

   We already saw that $l \in F_L$ and $r \in F_R$ implies $l.h' + r.h' = l.h + r.h$. For the opposite case, when $l \in L - F_L$ and $R \in R - F_R$, we have that $l.h' = l.h$ and $r.h' = r.h$, so that again $l.h' + r.h' = l.h + r.h$. Thus, if edge (l, r) is in the matching M for the equality graph G_h, then $(r, l) \in E_{M,h'}$.

3. Let (l, r) be an edge not in E_h such that $l \in F_L$, $r \in R - F_R$, and $\delta = l.h + r.h - w(l, r)$. By the definition of δ, there is at least one such edge. Then, we have

$$\begin{aligned} l.h' + r.h' &= l.h - \delta + r.h \\ &= l.h - (l.h + r.h - w(l, r)) + r.h \\ &= w(l, r) , \end{aligned}$$

and thus $(l, r) \in E_{h'}$. Since (l, r) is not in E_h, it is not in the matching M, so that in $E_{M,h'}$ it must be directed from L to R. Thus, $(l, r) \in E_{M,h'}$. ■

It is possible for an edge to belong to $E_{M,h}$ but not to $E_{M,h'}$. By Lemma 25.15, any such edge belongs neither to the matching M nor to the breadth-first forest F at the time that the new feasible vertex labeling h' is computed. (See Exercise 25.3-3.)

Going back to Figure 25.6(d), the queue became empty before an M-augmenting path was found. Figure 25.7 shows the next steps taken by the algorithm. The value of $\delta = 1$ is achieved by the edge (l_5, r_3) because in Figure 25.4(a), $l_5.h + r_3.h - w(l_5, r_3) = 6 + 0 - 5 = 1$. In Figure 25.7(a), the values of $l_3.h$, $l_4.h$, $l_5.h$, and $l_7.h$ have decreased by 1 and the values of $r_2.h$ and $r_7.h$ have increased by 1 because these vertices are in F. As a result, the edges (l_1, r_2) and (l_6, r_7) leave $G_{M,h}$ and the edge (l_5, r_3) enters. Figure 25.7(b) shows the new directed equality subgraph $G_{M,h}$. With edge (l_5, r_3) now in $G_{M,h}$, Figure 25.7(c) shows that this edge is added to the breadth-first forest F, and r_3 is added to the queue. Parts (c)–(f) show the breadth-first forest continuing to be built until in part (f), the queue once again becomes empty after vertex l_2, which has no edges leaving, is removed. Again, the algorithm must update the feasible vertex labeling and the directed equality subgraph. Now the value of $\delta = 1$ is achieved by three edges: (l_1, r_6), (l_5, r_6), and (l_7, r_6).

As Figure 25.8 shows in parts (a) and (b), these edges enter $G_{M,h}$, and edge (l_6, r_3) leaves. Part (c) shows that edge (l_1, r_6) is added to the breadth-first forest. (Either of edges (l_5, r_6) or (l_7, r_6) could have been added instead.) Because r_6 is unmatched, the search has found the M-augmenting path $\langle (l_5, r_3), (r_3, l_1), (l_1, r_6) \rangle$, highlighted in orange.

Figure 25.9(a) shows $G_{M,h}$ after the matching M has been updated by taking its symmetric difference with the M-augmenting path. The Hungarian algorithm starts its last breadth-first search, with vertex l_7 as the only root. The search proceeds as shown in parts (b)–(h) of the figure, until the queue becomes empty after removing l_4. This time, we find that $\delta = 2$, achieved by the five edges (l_2, r_5), (l_3, r_1), (l_4, r_5), (l_5, r_1), and (l_5, r_5), each of which enters $G_{M,h}$. Figure 25.10(a) shows the results of decreasing the feasible vertex label of each vertex in F_L by 2 and increasing the feasible vertex label of each vertex in F_R

$$\begin{array}{cc|ccccccc|}
 & & r_1 & r_2 & r_3 & r_4 & r_5 & r_6 & r_7 \\
 & h & 0 & 1 & 0 & 0 & 0 & 0 & 1 \\
\hline
l_1 & 10 & 4 & 10 & 10 & 10 & 2 & 9 & 3 \\
l_2 & 12 & 6 & 8 & 5 & 12 & 9 & 7 & 2 \\
l_3 & 14 & 11 & 9 & 6 & 7 & 9 & 5 & 15 \\
l_4 & 8 & 3 & 9 & 6 & 7 & 5 & 6 & 3 \\
l_5 & 5 & 2 & 6 & 5 & 3 & 2 & 4 & 2 \\
l_6 & 11 & 10 & 8 & 11 & 4 & 11 & 2 & 11 \\
l_7 & 7 & 3 & 4 & 5 & 4 & 3 & 6 & 8 \\
\hline
\end{array}$$

(a)

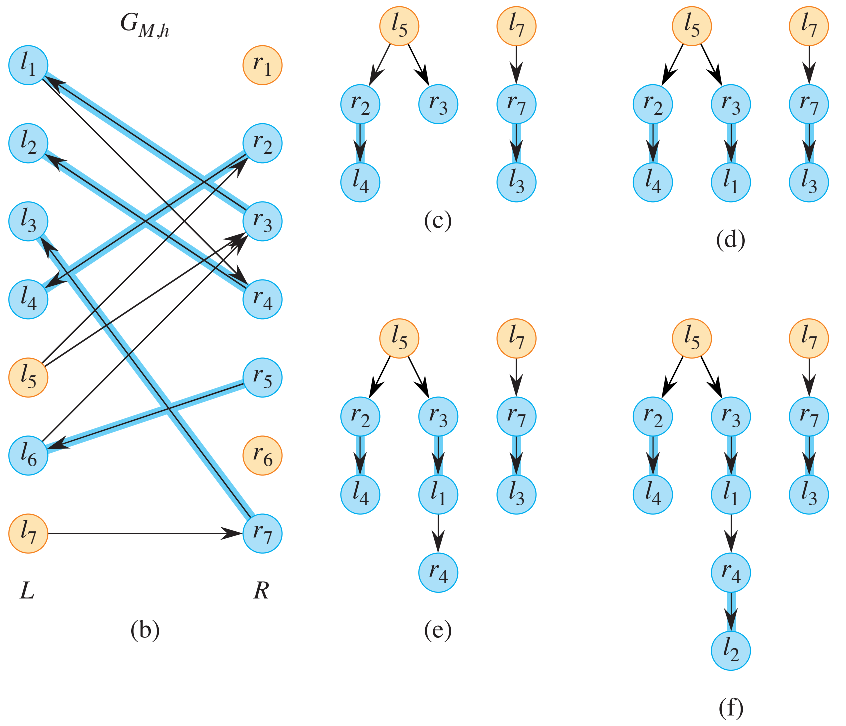

$G_{M,h}$
l_1 l_2 l_3 l_4 l_5 l_6 l_7
r_1 r_2 r_3 r_4 r_5 r_6 r_7
L
R
(b)
(c)
(d)
(e)
(f)

Figure 25.7 Updating the feasible vertex labeling and the directed equality subgraph $G_{M,h}$ when the queue becomes empty before finding an M-augmenting path. **(a)** With $\delta = 1$, the values of $l_3.h$, $l_4.h$, $l_5.h$, and $l_7.h$ decreased by 1 and $r_2.h$ and $r_7.h$ increased by 1. Edges (l_1, r_2) and (l_6, r_7) leave $G_{M,h}$, and edge (l_5, r_3) enters. These changes are highlighted in yellow. **(b)** The resulting directed equality subgraph $G_{M,h}$. **(c)–(f)** With edge (l_5, r_3) added to the breadth-first forest and r_3 added to the queue, the breadth-first search continues until the queue once again becomes empty in part (f).

by 2, and Figure 25.10(b) shows the resulting directed equality subgraph $G_{M,h}$. Part (c) shows that edge (l_3, r_1) is added to the breadth-first forest. Since r_1 is an unmatched vertex, the search terminates, having found the M-augmenting path $\langle (l_7, r_7), (r_7, l_3), (l_3, r_1) \rangle$, highlighted in orange. If r_1 had been matched, vertex r_5 would also have been added to the breadth-first forest, with any of l_2, l_4, or l_5 as its parent.

After updating the matching M, the algorithm arrives at the perfect matching shown for the equality subgraph G_h in Figure 25.11. By Theorem 25.14, the edges in M form an optimal solution to the original assignment problem given in the matrix. Here, the weights of edges (l_1, r_6), (l_2, r_4), (l_3, r_1), (l_4, r_2), (l_5, r_3), (l_6, r_5), and (l_7, r_7) sum to 65, which is the maximum weight of any matching.

The weight of the maximum-weight matching equals the sum of all the feasible vertex labels. These problems—maximizing the weight of a matching and mini-

		r_1	r_2	r_3	r_4	r_5	r_6	r_7
	h	0	2	1	1	0	0	2
l_1	9	4	10	10	10	2	9	3
l_2	11	6	8	5	12	9	7	2
l_3	13	11	9	6	7	9	5	15
l_4	7	3	9	6	7	5	6	3
l_5	4	2	6	5	3	2	4	2
l_6	11	10	8	11	4	11	2	11
l_7	6	3	4	5	4	3	6	8

(a)

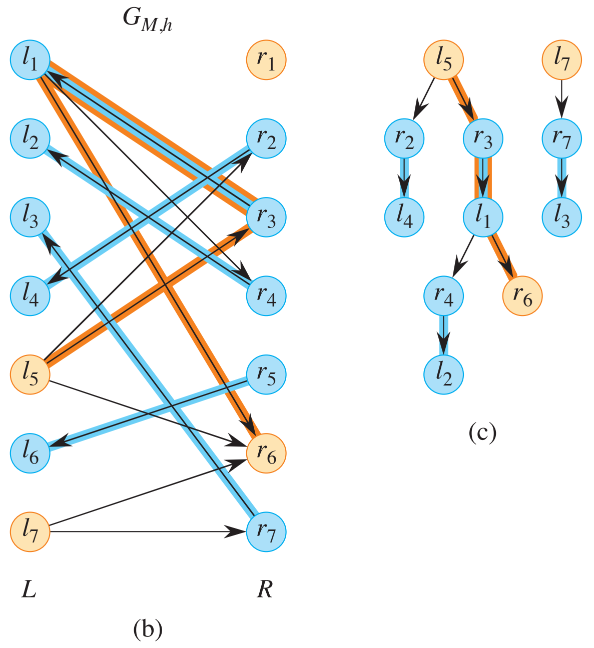


Figure 25.8 Another update to the feasible vertex labeling and directed equality subgraph $G_{M,h}$ because the queue became empty before finding an M-augmenting path. **(a)** With $\delta = 1$, the values of $l_1.h$, $l_2.h$, $l_3.h$, $l_4.h$, $l_5.h$, and $l_7.h$ decrease by 1, and $r_2.h$, $r_3.h$, $r_4.h$, and $r_7.h$ increase by 1. Edge (l_6, r_3) leaves $G_{M,h}$, and edges (l_1, r_6), (l_5, r_6) and (l_7, r_6) enter. **(b)** The resulting directed equality subgraph $G_{M,h}$. **(c)** With edge (l_1, r_6) added to the breadth-first forest and r_6 unmatched, the search terminates, having found the M-augmenting path $\langle (l_5, r_3), (r_3, l_1), (l_1, r_6) \rangle$, highlighted in orange in parts (b) and (c).

mizing the sum of the feasible vertex labels—are "duals" of each other, in a similar vein to how the value of a maximum flow equals the capacity of a minimum cut. Section 29.3 explores duality in more depth.

The Hungarian algorithm

The procedure HUNGARIAN on page 737 and its subroutine FIND-AUGMENTING-PATH on page 738 follow the steps we have just seen. The third property in Lemma 25.15 ensures that in line 23 of FIND-AUGMENTING-PATH the queue Q is nonempty. The pseudocode uses the attribute π to indicate predecessor vertices in the breadth-first forest. Instead of coloring vertices, as in the BFS procedure on page 556, the search puts the discovered vertices into the sets F_L and F_R. Because the Hungarian algorithm does not need breadth-first distances, the pseudocode omits the d attribute computed by the BFS procedure.

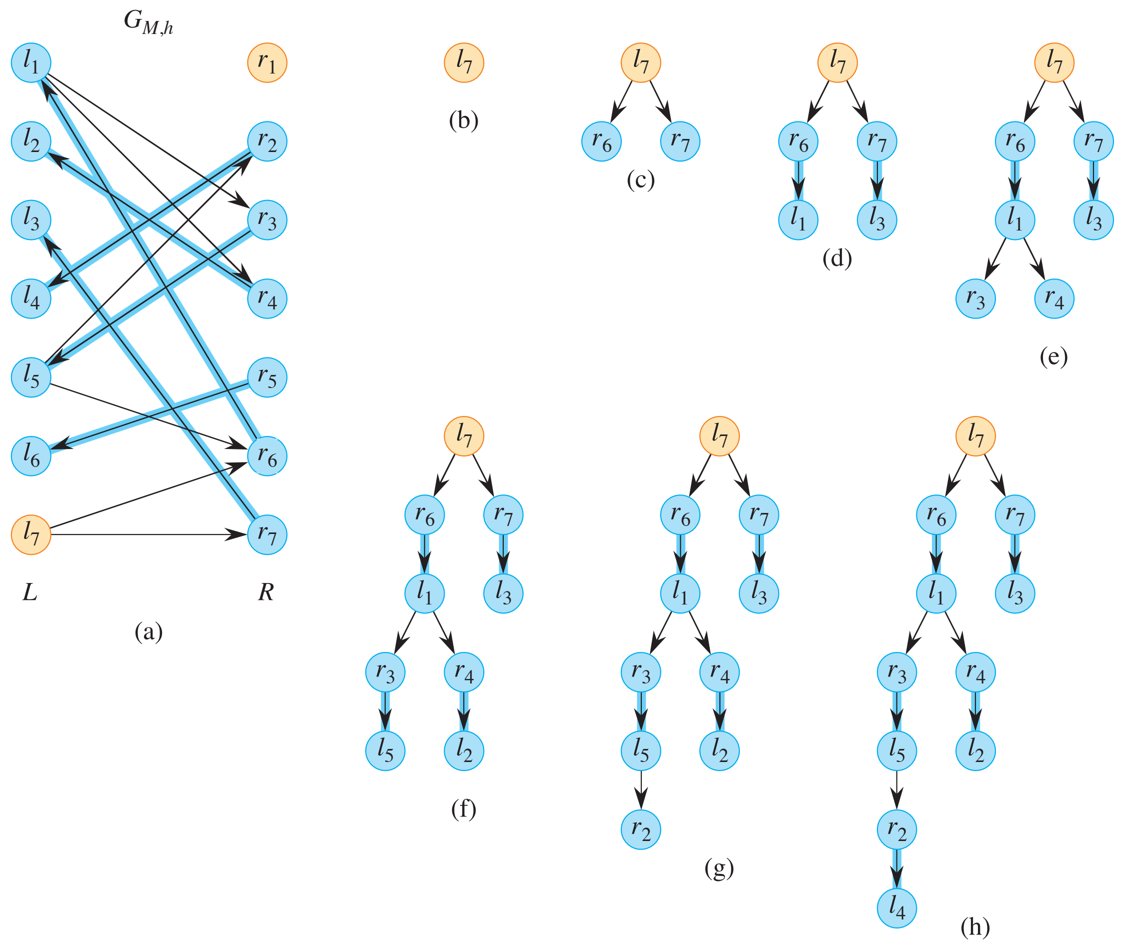


Figure 25.9 **(a)** The new matching M and the new directed equality subgraph $G_{M,h}$ after updating the matching in Figure 25.8 with the M-augmenting path in Figure 25.8 parts (b) and (c). **(b)–(h)** Successive versions of the breadth-first forest F in a new breadth-first search with root l_7. After the vertex l_4 in part (h) has been removed from the queue, the queue becomes empty before the search discovers an unmatched vertex in R.

Now, let's see why the Hungarian algorithm runs in $O(n^4)$ time, where $|V| = n/2$ and $|E| = n^2$ in the original graph G. (Below we outline how to reduce the running time to $O(n^3)$.) You can go through the pseudocode of HUNGARIAN to verify that lines 1–6 and 11 take $O(n^2)$ time. The **while** loop of lines 7–10 iterates at most n times, since each iteration increases the size of the matching M by 1. Each test in line 7 can take constant time by just checking whether $|M| < n$, each update of M in line 9 takes $O(n)$ time, and the updates in line 10 take $O(n^2)$ time.

To achieve the $O(n^4)$ time bound, it remains to show that each call of FIND-AUGMENTING-PATH runs in $O(n^3)$ time. Let's call each execution of lines 10–22

		r_1	r_2	r_3	r_4	r_5	r_6	r_7
	h	0	4	3	3	0	2	4
l_1	7	4	10	10	10	2	9	3
l_2	9	6	8	5	12	9	7	2
l_3	11	11	9	6	7	9	5	15
l_4	5	3	9	6	7	5	6	3
l_5	2	2	6	5	3	2	4	2
l_6	11	10	8	11	4	11	2	11
l_7	4	3	4	5	4	3	6	8

(a)

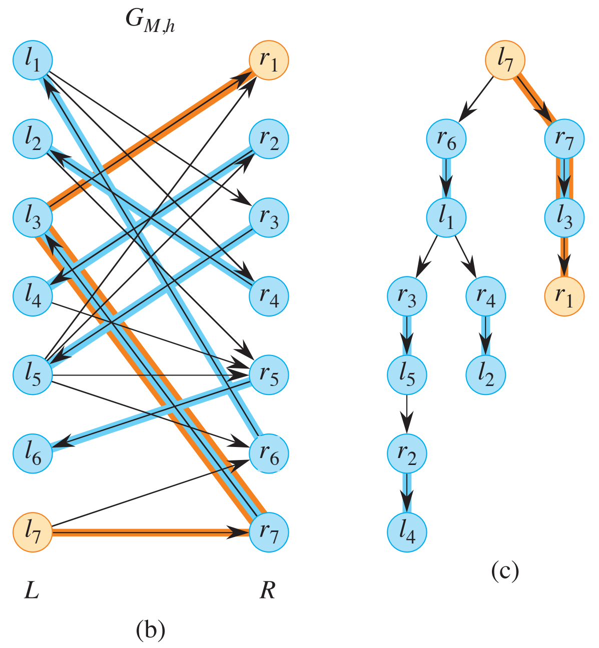


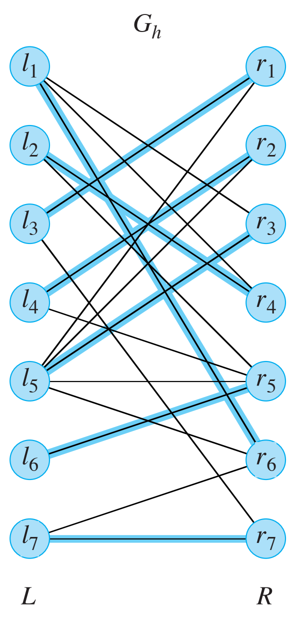
Figure 25.10 Updating the feasible vertex labeling and directed equality subgraph $G_{M,h}$. **(a)** Here, $\delta = 2$, so the values of $l_1.h$, $l_2.h$, $l_3.h$, $l_4.h$, $l_5.h$, and $l_7.h$ decreased by 2, and the values of $r_2.h$, $r_3.h$, $r_4.h$, $r_6.h$, and $r_7.h$ increased by 2. Edges (l_2, r_5), (l_3, r_1), (l_4, r_5), (l_5, r_1), and (l_5, r_5) enter $G_{M,h}$. **(b)** The resulting directed graph $G_{M,h}$. **(c)** With edge (l_3, r_1) added to the breadth-first forest and r_1 unmatched, the search terminates, having found the M-augmenting path $\langle (l_7, r_7), (r_7, l_3), (l_3, r_1) \rangle$, highlighted in orange in parts (b) and (c).

a ***growth step***. Ignoring the growth steps, you can verify that FIND-AUGMENTING-PATH is a breadth-first search. With the sets F_L and F_R represented appropriately, the breadth-first search takes $O(V + E) = O(n^2)$ time. Within a call of FIND-AUGMENTING-PATH, at most n growth steps can occur, since each growth step is guaranteed to discover at least one vertex in R. Since there are at most n^2 edges in $G_{M,h}$, the **for** loop of lines 16–22 iterates at most n^2 times per call of FIND-AUGMENTING-PATH. The bottleneck is lines 10 and 15, which take $O(n^2)$ time, so that FIND-AUGMENTING-PATH takes $O(n^3)$ time.

Exercise 25.3-5 asks you to show that reconstructing the directed equality subgraph $G_{M,h}$ in line 15 is actually unnecessary, so that its cost can be eliminated. Reducing the cost of computing δ in line 10 to $O(n)$ takes a little more effort and is the subject of Problem 25-2. With these changes, each call of FIND-AUGMENTING-PATH takes $O(n^2)$ time, so that the Hungarian algorithm runs in $O(n^3)$ time.

$$\begin{array}{cc|ccccccc|}
 & & r_1 & r_2 & r_3 & r_4 & r_5 & r_6 & r_7 \\
 & h & 0 & 4 & 3 & 3 & 0 & 2 & 4 \\
\hline
l_1 & 7 & 4 & 10 & 10 & 10 & 2 & \mathbf{9} & 3 \\
l_2 & 9 & 6 & 8 & 5 & \mathbf{12} & 9 & 7 & 2 \\
l_3 & 11 & \mathbf{11} & 9 & 6 & 7 & 9 & 5 & 15 \\
l_4 & 5 & 3 & \mathbf{9} & 6 & 7 & 5 & 6 & 3 \\
l_5 & 2 & 2 & 6 & \mathbf{5} & 3 & 2 & 4 & 2 \\
l_6 & 11 & 10 & 8 & 11 & 4 & \mathbf{11} & 2 & 11 \\
l_7 & 4 & 3 & 4 & 5 & 4 & 3 & 6 & \mathbf{8} \\
\hline
\end{array}$$

(a)



(b)

Figure 25.11 The final matching, shown for the equality subgraph G_h with blue edges and blue entries in the matrix. The weights of the edges in the matching sum to 65, which is the maximum for any matching in the original complete bipartite graph G, as well as the sum of all the final feasible vertex labels.

```
HUNGARIAN($G$)
1  for each vertex $l \in L$
2      $l.h = \max\{w(l,r) : r \in R\}$    // from equation (25.1)
3  for each vertex $r \in R$
4      $r.h = 0$                            // from equation (25.2)
5  let $M$ be any matching in $G_h$ (such as the matching returned by
       GREEDY-BIPARTITE-MATCHING)
6  from $G$, $M$, and $h$, form the equality subgraph $G_h$
       and the directed equality subgraph $G_{M,h}$
7  while $M$ is not a perfect matching in $G_h$
8      $P$ = FIND-AUGMENTING-PATH($G_{M,h}$)
9      $M = M \oplus P$
10     update the equality subgraph $G_h$
           and the directed equality subgraph $G_{M,h}$
11 return $M$
```

```
FIND-AUGMENTING-PATH($G_{M,h}$)
 1  $Q = \emptyset$
 2  $F_L = \emptyset$
 3  $F_R = \emptyset$
 4  for each unmatched vertex $l \in L$
 5      $l.\pi$ = NIL
 6      ENQUEUE($Q, l$)
 7      $F_L = F_L \cup \{l\}$        // forest $F$ starts with unmatched vertices in $L$
 8  repeat
 9      if $Q$ is empty           // ran out of vertices to search from?
10          $\delta = \min\{l.h + r.h - w(l, r) : l \in F_L$ and $r \in R - F_R\}$
11          for each vertex $l \in F_L$
12              $l.h = l.h - \delta$      // relabel according to equation (25.5)
13          for each vertex $r \in F_R$
14              $r.h = r.h + \delta$      // relabel according to equation (25.5)
15          from $G$, $M$, and $h$, form a new directed equality graph $G_{M,h}$
16          for each new edge $(l, r)$ in $G_{M,h}$      // continue search with new edges
17              if $r \notin F_R$
18                  $r.\pi = l$                   // discover $r$, add it to $F$
19                  if $r$ is unmatched
20                      an $M$-augmenting path has been found
                            (exit the repeat loop)
21                  else ENQUEUE($Q, r$)         // can search from $r$ later
22                      $F_R = F_R \cup \{r\}$
23      $u$ = DEQUEUE($Q$)                        // search from $u$
24      for each neighbor $v$ of $u$ in $G_{M,h}$
25          if $v \in L$
26              $v.\pi = u$
27              $F_L = F_L \cup \{v\}$               // discover $v$, add it to $F$
28              ENQUEUE($Q, v$)                 // can search from $v$ later
29          elseif $v \notin F_R$                     // $v \in R$, do same as lines 18–22
30              $v.\pi = u$
31              if $v$ is unmatched
32                  an $M$-augmenting path has been found
                        (exit the repeat loop)
33              else ENQUEUE($Q, v$)
34                  $F_R = F_R \cup \{v\}$
35  until an $M$-augmenting path has been found
36  using the predecessor attributes $\pi$, construct an $M$-augmenting path $P$
        by tracing back from the unmatched vertex in $R$
37  return $P$
```