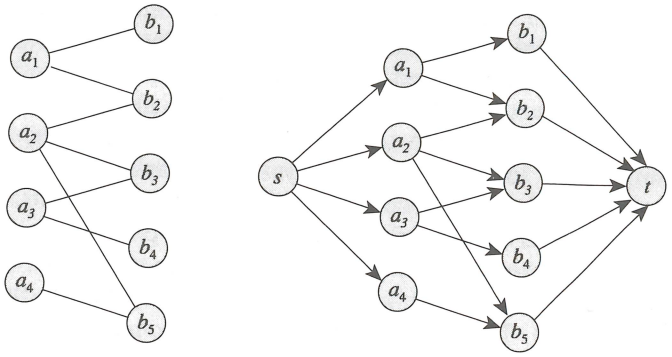

Figure 12.4 Transforming a bipartite (cardinality) matching problem to a maximum flow problem: (a) original network; (b) unit capacity maximum flow network.

constraints, we set the flow on the arcs (s, i_r) and (j_r, t) equal to 1 for all $r = 1, 2, \ldots, k$. Clearly, this choice gives us a flow of value k from node s to node t.

Similarly, given an integral flow of value k from node s to node t in the transformed network, we can specify a corresponding matching in the original network: By flow decomposition, the integral flow of cardinality k decomposes into k paths of the form $s - i_1 - j_1 - t, s - i_2 - j_2 - t, \ldots, s - i_k - j_k - t$. Since each of the arcs incident to nodes s and t have a unit capacity, no two nodes in N_1 or N_2 appear in more than one of these paths, and so the k arcs $\{(i_1, j_1), (i_2, j_2), \ldots, (i_k, j_k)\}$ define a matching.

We have thus established an equivalence between matchings in the original network and integral flows in the transformed network. Therefore, to solve the matching problem, we solve a maximum flow problem in the transformed network using the $O(\sqrt{n}m)$ time algorithm described in Section 8.2. Recall that this algorithm produces an integer optimal flow. The matching corresponding to the maximum flows is a maximum cardinality matching. We have therefore established the following result.

Theorem 12.1. *It is possible to solve the maximum cardinality bipartite matching problem in $O(\sqrt{n}m)$ time.*

12.4 BIPARTITE WEIGHTED MATCHING PROBLEM

In this section we study the bipartite weighted matching problem; namely, given a weighted bipartite network $G = (N_1 \cup N_2, A)$ with $|N_1| = |N_2|$ and arc weights c_{ij}, find a perfect matching of minimum weight. We allow the network G to be directed or undirected. If the network is directed, we require that for each arc $(i, j) \in A$, $i \in N_1$ and $j \in N_2$. If the network is undirected, we make it directed by designating all arcs as pointing from the nodes in N_1 to those in N_2. We shall, therefore, henceforth assume that G is a directed graph. In the operations research literature, the bipartite weighted matching problem is known as the *assignment problem*; for the sake of brevity and to conform with this convention, we adopt this terminology.

Recall that the assignment problem is a special case of the minimum cost flow problem and can be stated as the following linear program.

$$\text{Minimize} \quad \sum_{(i,j)\in A} c_{ij}x_{ij} \tag{12.1a}$$

subject to

$$\sum_{\{j:(i,j)\in A\}} x_{ij} = 1 \qquad \text{for all } i \in N_1, \tag{12.1b}$$

$$\sum_{\{j:(j,i)\in A\}} x_{ji} = 1 \qquad \text{for all } i \in N_2, \tag{12.1c}$$

$$x_{ij} \geq 0 \qquad \text{for all } (i, j) \in A. \tag{12.1d}$$

Since we can formulate the weighted bipartite matching problem as this special type of flow problem, it is not too surprising to learn that most algorithms for the assignment problem can be viewed as adaptations of algorithms for the minimum cost flow problem. However, the special structure of the assignment problem often permits us to simplify these algorithms and to obtain improved bounds on their running times.

One popular algorithm for the assignment problem is a specialization of the network simplex algorithm discussed in Chapter 11. Another popular algorithm is the successive shortest path algorithm and its many variants. In the following discussion, we briefly describe some of these successive shortest path-based algorithms. We also describe an adaptation of the cost scaling algorithm.

Successive Shortest Path Algorithm

This algorithm is a direct implementation of the successive shortest path algorithm for the minimum cost flow problem discussed in Section 9.7. Recall that the successive shortest path algorithm obtains shortest path distances from a supply node to all other nodes in a residual network, uses these distances to update node potentials and then augments flow from that supply node to a demand node. This algorithm, when applied to the assignment problem, would augment 1 unit flow in every iteration, which would amount to assigning one additional node in N_1. Consequently, if we let $S(n, m, C)$ denote the time needed to solve a shortest path problem with nonnegative arc lengths and let $n_1 = |N_1|$, the algorithm would terminate within n_1 iterations and would require $O(n_1 S(n, m, C))$ time.

Hungarian Algorithm

The Hungarian algorithm is a direct implementation of the primal–dual algorithm for the minimum cost flow problem that we discussed in Section 9.8. Recall that the primal–dual algorithm first transforms the minimum cost flow problem into a problem with a single supply node s^* and a single demand node t^*. At every iteration, the primal–dual algorithm computes shortest path distances from s^* to all other nodes, updates node potentials, and then solves a maximum flow problem that sends the maximum possible flow from node s^* to node t^* over arcs with zero reduced costs. When applied to the assignment problem, this algorithm terminates within n_1 iter-

ations since each iteration sends at least 1 unit of flow, and hence assigns at least one additional node in N_1. The time required to solve shortest path problems in all these iterations is $O(n_1 S(n, m, C))$. Next consider the total time required to establish maximum flows. The labeling algorithm, described in Section 6.5, for solving the maximum flow problem would require a total of $O(nm)$ time because it would perform n augmentations and each augmentation requires $O(m)$ time. The dominant portion of these computations is the time required to solve shortest path problems. Consequently, the overall running time of the algorithm is $O(n_1 S(n, m, C))$.

Relaxation Algorithm

The relaxation algorithm, which is closely related to the successive shortest path algorithm, is another popular approach for solving the assignment problem. This algorithm relaxes the constraint (12.1c), thus allowing any node in N_2 to be assigned to more than one node in N_1. The relaxed problem is easy to solve: We assign each node $i \in N_1$ to any node $j \in N_2$ with the minimum cost c_{ij} among all arcs in $A(i)$. As a result, some nodes in N_2 might be unassigned while some other nodes are overassigned (i.e., assigned to more than one node in N_1). The algorithm then gradually converts this solution to a feasible assignment while always maintaining the reduced cost optimality condition. At each iteration the algorithm selects an overassigned node k in N_2, obtains shortest path distances from node k to all other nodes in the residual network with reduced costs as arc lengths, updates node potentials, and augments a unit flow from node k to an unassigned node in N_2 along the shortest path. Since each iteration assigns one more node in N_2 and never converts any assigned node into an unassigned node, within n_1 such iterations, the algorithm obtains a feasible assignment. The relaxation algorithm maintains optimality conditions throughout. Therefore, the shortest path problems have nonnegative arc lengths, and the overall running time of the algorithm is $O(n_1 S(n, m, C))$.

Cost Scaling Algorithm

This algorithm is an adaptation of the cost scaling algorithm for the minimum cost flow problem discussed in Section 10.3. Recall that the cost scaling algorithm performs $O(\log(nC))$ scaling phases and the generic implementation requires $O(n^2 m)$ time for each scaling phase. The bottleneck operation in each scaling phase is performing nonsaturating pushes which require $O(n^2 m)$ time; all other operations, such as finding admissible arcs and performing saturating pushes, require $O(nm)$ time. When we apply the cost scaling algorithm to the assignment problem, each push is a saturating push since each arc capacity is 1. Consequently, the cost scaling algorithm solves the assignment problem in $O(nm \log(nC))$ time.

A modified version of the cost scaling algorithm has an improved running time of $O(\sqrt{n}m \log(nC))$, which is the best available time bound for assignment problems satisfying the similarity assumption. This improvement rests on decomposing the computations in each scaling phase into two subphases. In the first subphase, we apply the usual cost scaling algorithm with the difference that whenever we have relabeled a node more than $2\sqrt{n}$ times, we set this node aside and do not examine it further. When we have set aside all (remaining) active nodes, we initiate the second subphase. It is possible to show that the first subphase requires $O(\sqrt{n_1}m)$ time, and

when it ends, the network will contain at most $O(\sqrt{n_1})$ active nodes. The second subphase makes these active nodes inactive by identifying "approximate shortest paths" from nodes with excesses to nodes with deficits and augmenting unit flow along these paths. The algorithm uses Dial's algorithm (described in Section 4.6) to identify each such path in $O(m)$ time. Consequently, the second subphase also runs in $O(\sqrt{n_1}m \log(nC))$. We provide a reference for this algorithm in the reference notes.

We summarize the preceding discussion.

Theorem 12.2. *The successive shortest path algorithm, Hungarian algorithm, and the relaxation algorithm solve the assignment problem in $O(n_1 S(n, m, C))$ time. A straightforward implementation of the cost scaling algorithm solves the assignment problem in $O(nm \log(nC))$ time and a further improvement of this algorithm runs in $O(\sqrt{n_1}m \log(nC))$ time.* ◆

12.5 STABLE MARRIAGE PROBLEM

The stable marriage problem is a novel application of bipartite matchings. This problem can be stated as follows. A certain community consists of n men and n women. Each person ranks those of the opposite sex in accordance with his or her preferences for a spouse. For a given matching, a man–woman pair is said to be *unstable* if they are not married to each other but prefer each other to their current spouses. A perfect matching (marriage) of men and women is said to be *stable* if it contains no unstable pairs. The stable marriage problem is to identify a stable perfect matching. In this section we show that for *any* set of rankings, we can always find a stable matching. We establish this result constructively, specifying an algorithm that constructs a stable matching in $O(n^2)$ time.

The input to the stable marriage problem consists of two $n \times n$ matrices; the first matrix gives each man's ranking of women and the second matrix gives each woman's ranking of men. A higher rank denotes a more favored person. Without any loss of generality, we can assume that each rank is an integer between 1 and n. To implement the stable marriage algorithm efficiently, we use these two matrices to construct a vector of n elements for each person, called his or her *priority list*, that lists the persons of opposite sex in decreasing order of their rankings. Since all the ranks are between 1 and n, we can construct these priority lists in a total of $O(n^2)$ time using a bucket sort algorithm (see Exercise 12.30).

The algorithm for the stable marriage problem is an iterative greedy algorithm: Each man proposes to his most preferred woman, and each woman receiving more than one proposal rejects all except her most preferred man from among those who have proposed to her. The algorithm maintains a set, LIST, of unassigned men and for each man it maintains an index, called *current-woman*, which denotes the woman in his priority list that he will next offer a proposal. Initially, LIST = N_1, the set of all men, and the *current-woman* of each man is the first woman in his priority list.

The stable marriage algorithm proceeds as follows. At each iteration, the algorithm selects a man from LIST, say Bill, and he proposes to his *current-woman*, say Helen. If Helen is still unassigned, she accepts the proposal and Bill and Helen

Assignment problem. The assignment problem has been a popular, heavily studied research topic within the operations research community. The paper by Ahuja, Magnanti, and Orlin [1989] presented a detailed survey of assignment algorithms. Kuhn [1955] developed the first (primal–dual) algorithm for the assignment problem. Although researchers have developed several different algorithms for the assignment problem, many of these algorithms share common features. The successive shortest path algorithm for the minimum cost flow problem, discussed in Section 9.7, appears to lie at the heart of many (apparently different) assignment algorithms. This approach yields an $O(n\, S(n, m, C))$ time algorithm for solving the assignment problem, where $S(n, m, C)$ is the time needed for solving a shortest path problem with nonnegative arc lengths. Currently, $S(n, m, C) = O(\min\{m + n \log n, m \log\log C, m + n\sqrt{\log C}\})$. Therefore, $O(nm + n^2 \log n)$ is the best available strongly polynomial time bound for solving the assignment problem. Gabow and Tarjan [1989a] developed a cost scaling algorithm for the assignment problem that runs in $O(n^{1/2}m \log(nC))$ time. Bertsekas [1988] proposed an auction algorithm for the assignment problem. Incorporating scaling in the auction algorithm, Orlin and Ahuja [1992] also obtained an $O(n^{1/2}m \log(nC))$ time algorithm; this is the algorithm that we mentioned in Section 12.4. The reference notes for Chapter 11 provide references for simplex-based approaches for the assignment problem. Carpento, Martello, and Toth [1988] presented FORTRAN codes for several algorithms for the assignment problem. For recent computational studies of assignment algorithms, see Bertsekas [1988], Zaki [1990], and Kennington and Wang [1990].

Nonbipartite weighted matching problems. Edmonds [1965b] gave the first algorithm for the nonbipartite weighted matching problem. Gabow [1975] and Lawler [1976] developed $O(n^3)$ implementations of this algorithm. Currently, the fastest algorithms for this problem are (1) an $O(nm + n^2 \log n)$ algorithm due to Gabow [1990], and (2) an $O(m \log(nC)\sqrt{n\alpha(m, n) \log n})$ algorithm due to Gabow and Tarjan [1989b]. For information concerning the empirical behavior of nonbipartite weighted matching algorithms, see Grötschel and Holland [1985].

Stable marriage problem. Our discussion of this problem has presented the most basic results obtained by Gale and Shapley [1962]. The book by Gusfield and Irving [1989] on the stable marriage problem contains a wealth of information on this topic. The paper by Roth, Rothblum, and Vande Vate [1990] studied polyhedral aspects of the stable marriage problem and used linear programming theory to obtain simpler proofs of many fundamental results for the problem.

Paths and assignments. We presented two transformations to reduce shortest path problems to matching problems. The transformation of the shortest path problem in directed networks to an assignment problem is due to Hoffman and Markowitz [1963] and the transformation of the shortest path problem in undirected networks to the nonbipartite weighted matching problem is due to Edmonds [1967].

The applications of matchings that we gave in Section 12.2 are adapted from the following papers: