(taken from Levitin, pages 361-370, section 10.2)

In this section, we consider the important problem of maximizing the flow of a ma- terial through a transportation network (pipeline system, communication system, electrical distribution system, and so on). We will assume that the transportation network in question can be represented by a connected weighted digraph with n vertices numbered from 1 to n and a set of edges E, with the following properties:

- It contains exactly one vertex with no entering edges; this vertex is called the source and assumed to be numbered 1.
- It contains exactly one vertex with no leaving edges; this vertex is called the sink and assumed to be numbered n.
- The weight uij of each directed edge (i, j) is a positive integer, called the *edge capacity*. (This number represents the upper bound on the amount of the material that can be sent from i to j through a link represented by this edge.)

A digraph satisfying these properties is called a *flow network* or simply *a network*. A small instance of a network is given in Figure 10.4 below:

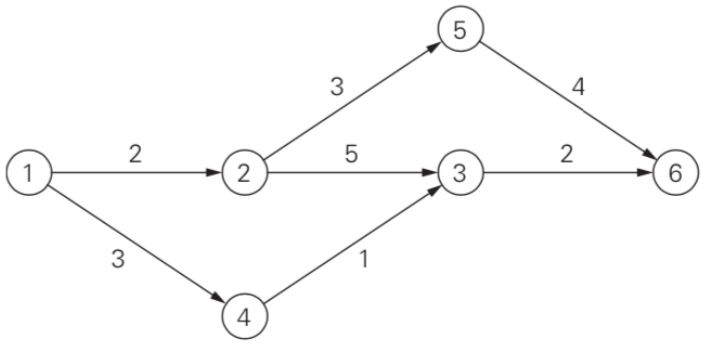

Figure 10.4

It is assumed that the source and the sink are the only source and destination of the material, respectively; all the other vertices can serve only as points where a flow can be redirected without consuming or adding any amount of the material. In other words, the total amount of the material entering an intermediate vertex must be equal to the total amount of the material leaving the vertex. This condition is called the *flow-conservation requirement.*

Since no amount of the material can change by going through intermediate vertices of the network, it stands to reason that the total amount of the material leaving the source must end up at the sink. This quantity, the total outflow from the source - or, equivalently, the total inflow into the sink - is called the *value of the flow*. We denote it by v. It is this quantity that we will want to maximize over all possible flows in a network. Thus, a (feasible) flow is an assignment of real numbers xij to edges (i, j) of a given network that satisfy flow-conservation constraints and the capacity constraints $0 \le x_{ij} \le u_{ij}$ for every edge $(i, j) \in E$.

We can always start with the zero flow (i.e., set $x_{ij} = 0$ for every edge (i, j) in the network). Then, on each iteration, we can try to find a path from source to sink along which some additional flow can be sent. Such a path is called *flow augmenting*. If a flow-augmenting path is found, we adjust the flow along the edges of this path to get a flow of an increased value and try

to find an augmenting path for the new flow. If no flow-augmenting path can be found, we conclude that the current flow is optimal. This general template for solving the maximum-flow problem is called the augmenting-path method, also known as the Ford-Fulkerson method after L. R. Ford, Jr., and D. R. Fulkerson, who discovered it.

It is natural to search for a flow-augmenting path from source to sink by following directed edges (i, j) for which the current flow xij is less than the edge capacity u_{ij}. Among several possibilities, let us assume that we identify the augmenting path 1→2→3→6 first. We can increase the flow along this path by a maximum of 2 units, which is the smallest unused capacity of its edges. The new flow is shown in Figure 10.5b.

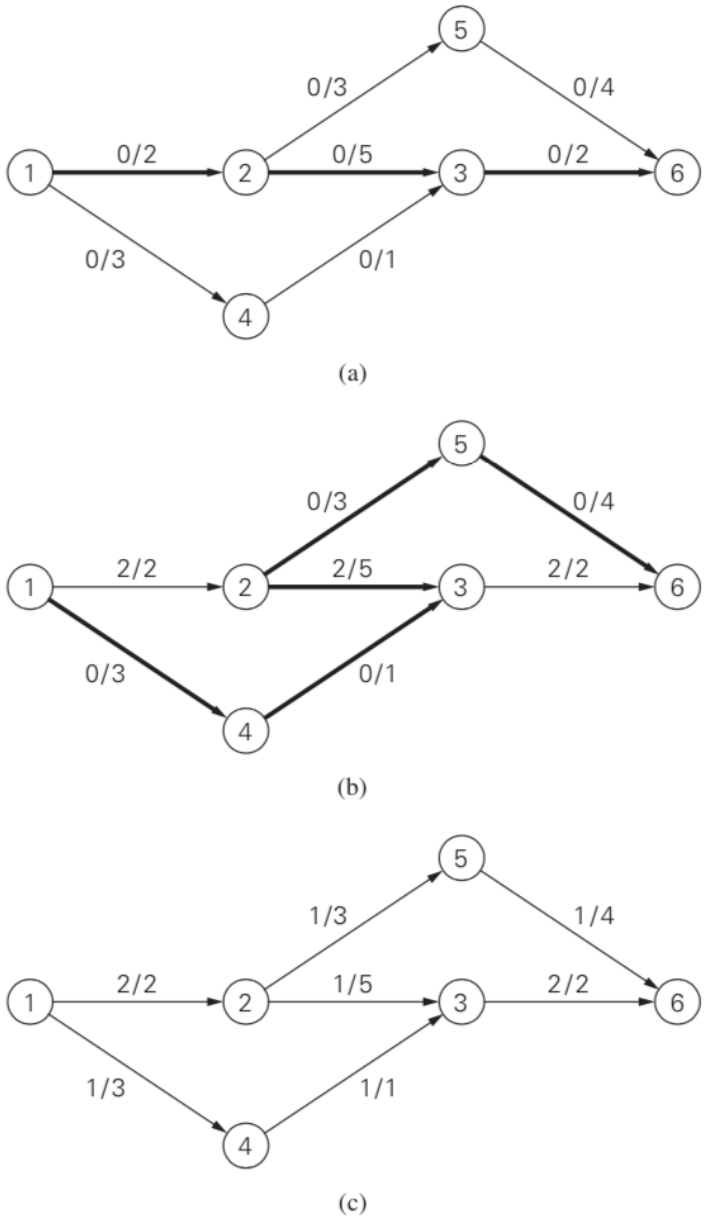

FIGURE 10.5 Illustration of the augmenting-path method. Flow-augmenting

This is as far as our simpleminded idea about flow-augmenting paths will be able to take us. Unfortunately, the flow shown in Figure 10.5b is not optimal: its value can still be increased along the path 1→4→3←2→5→6 by increasing the flow by 1 on edges (1, 4), (4, 3), (2, 5), and (5, 6) and decreasing it by 1 on edge (2, 3). The flow obtained as the result of this augmentation is shown in Figure 10.5c. It is indeed maximal. (Can you tell why?)

Thus, to find a flow-augmenting path for a flow x, we need to consider paths from source to sink in the underlying undirected graph in which any two consecutive vertices i, j are either

i. connected by a directed edge from i to j with some positive unused capacity $r_{ij} = u_{ij} - x_{ij}$ (so that we can increase the flow through that edge by up to r_{ij} units), or

ii. connected by a directed edge from j to i with some positive flow x_{ji} (so that we can decrease the flow through that edge by up to x_{ji} units).

Edges of the first kind are called *forward edges* because their tail is listed before their head in the vertex list 1 → . . . i → j . . . → n defining the path; edges of the second kind are called *backward edges* because their tail is listed after their head in the path list 1 → . . . i ← j . . . → n. To illustrate, for the path 1→4→3←2→5→6 of the last example, (1, 4), (4, 3), (2, 5), and (5, 6) are the forward edges, and (3, 2) is the backward edge.

Using Edmonds-Karp algorithm to solve the max flow for the network in Fig. 10.4, the steps are shown in Fig. 10.7:

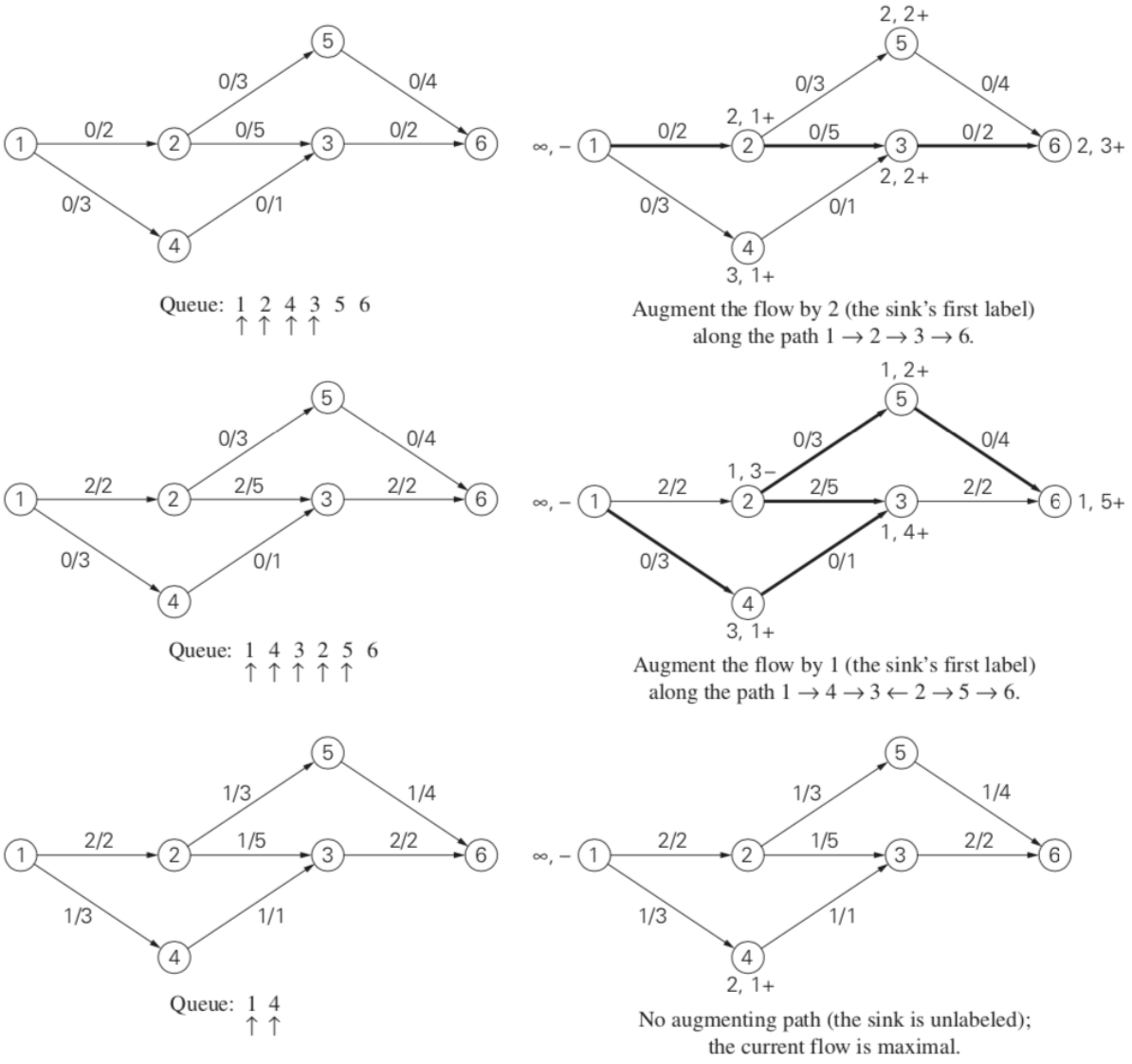

FIGURE 10.7 Illustration of the shortest-augmenting-path algorithm. The diagrams on the left show the current flow before the next iteration begins; the diagrams on the right show the results of the vertex labeling on that iteration, the augmenting path found (in bold), and the flow before its augmentation. Vertices deleted from the queue are indicated by the ↑ symbol.

Hence, the flow value increases at least by 1 on each iteration of the augmenting-path method. Since the value of a maximum flow is bounded above (e.g., by the sum of the capacities of the source edges), the augmenting-path method has to stop after a finite number

of iterations. Surprisingly, the final flow always turns out to be maximal, irrespective of a sequence of augmenting paths. This remarkable result stems from the proof of the Max-Flow Min-Cut Theorem.

THEOREM (Max-Flow Min-Cut Theorem) The value of a maximum flow in a network is equal to the capacity of its minimum cut.

A cut induced by partitioning vertices of a network into some subset X containing the source and X^-, the complement of X, containing the sink is the set of all the edges with a tail in X and a head in X^-. We denote a cut $C(X, X^-)$ or simply C. For example, for the network in Figure 10.4:

If X = {1} and henceX^- = {2,3,4,5,6}, $C(X,X^-)$ = {(1,2),(1,4)}

If X = {1,2,3,4,5} and hence X^-= {6}, $C(X,X^-)$ = {(3,6),(5,6)};

If X = {1,2,4} and hence X^- = {3,5,6}, $C(X,X^-)$={(2,3),(2,5),(4,3)}.

The name “cut” stems from the following property: if all the edges of a cut were deleted from the network, there would be no directed path from source to sink. Indeed, let $C(X, X^-)$ be a cut. Consider a directed path from source to sink. If v_i is the first vertex of that path which belongs to X^-(the set of such vertices is not empty, because it contains the sink), then v_i is not the source and its immediate predecessor v_{i-1} on that path belongs to X. Hence, the edge from v_{i-1} to v_i must be an element of the cut $C(X, X^-)$. This proves the property in question.

The capacity of a cut $C(X, X^-)$, denoted $c(X, X^-)$, is defined as the sum of capacities of the edges that compose the cut. For the three examples of cuts given above, the capacities are equal to 5, 6, and 9, respectively. Since the number of different cuts in a network is nonempty and finite (why?), there always exists a minimum cut, i.e., a cut with the smallest capacity.

The proof of the max-flow-min cut theorem implies that when the augmenting-path method terminates, it yields both a maximum flow and a minimum cut. If labeling of the kind utilized in the shortest-augmenting-path algorithm is used, a minimum cut is formed by the edges from the labeled to unlabeled ver- tices on the last iteration of the method. Finally, the proof implies that all such edges must be full (i.e., the flows must be equal to the edge capacities), and all the edges from unlabeled vertices to labeled, if any, must be empty (i.e., have zero flows on them). In particular, for the network in Figure 10.7, the algorithm finds the cut {(1, 2), (4, 3)} of minimum capacity 3, both edges of which are full as required.