2

THE SHORTEST DIRECTED PATH PROBLEM

A weighted directed graph is called a *directed network* or simply a *net.* The real numbers assigned to the arcs of the net may be positive, negative, or zero, and may be considered as the 'lengths' of the arcs. For example, if the streets of a city are represented by a net, where the nodes denote the intersections of the one-way and/or two-way streets and the arcs denote the streets, then the lengths associated with the arcs represent the true lengths of the streets from one intersection to the next. Likewise, an electrical system may be modeled by a net whose nodes denote the various states of the system and whose arcs represent all permissible transitions among the states. The length associated with an arc (i, j) is then the energy required in transforming the state i to the state j with positive length indicating energy absorption and negative length indicating energy release.

The *length* of a directed edge train is the sum of lengths of all the arcs in the directed edge train. In general, there are many directed edge trains from a specified node s to another t in a given net. A directed edge train with minimum length from s to t is called a *shortest directed edge train* from s to t in the net. The *length* of a directed circuit is the sum of lengths of all the arcs in the directed circuit. If the length of a directed circuit is negative, we say that the directed circuit is *negative* and the net contains a *negative directed circuit.* The problem of finding a shortest directed edge train in a net is not always well defined if the net contains a negative directed circuit. To see this, let C be such a negative directed circuit and let i be a node on C. If i is also on a directed edge train from node s to node t, then by proceeding from s to i, traversing the directed circuit C a sufficient number of times, and finally following the directed edge train from i to t, we obtain a directed edge train with arbitrarily small length, which may approach $-\infty$.

Thus, in such a situation the problem is not well defined. For our purposes, we shall restrict ourselves to the class of nets that do not contain any negative directed circuits. Then the problem of finding a shortest directed edge train from s to t is equivalent to that of finding a shortest directed path from s to t. This follows from the fact that a shortest directed path from s to t is as short as any shortest directed edge train from s to t provided that the net does not contain any negative directed circuits. Under this assumption, it is sufficient to consider the shortest directed path problem in a net that does not contain any negative directed circuits.

The shortest directed path problem is important in that it often occurs as a subproblem of other optimization problems. For instance, a minimal cost flow problem to be discussed in Chapter 5 can be formulated as a shortest directed path problem in a net in which the lengths of the arcs denote the costs in establishing the flows in the arcs. Or, in setting up a distribution system, it is frequently required to choose a minimal cost route among many alternate shipping routes. Cartaino and Dreyfus (1957) have shown that the problem of determining the least time for an airplane to climb to a given altitude can be formulated, in its discrete form, as a shortest directed path problem. Furthermore, the shortest directed path problem can usually be modified to solve other optimum path problems such as the longest directed path problem.

The purpose of this chapter is to describe a number of combinatorial algorithms for the shortest directed path problem in a net that does not contain any negative directed circuits.

2.1 SHORTEST DIRECTED PATHS

Given a net $G(V, E)$ with node set V and arc set E, suppose that each arc $(i, j) \in E$ is associated with a real number $l(i, j)$ called the *length* of the arc (i, j). The function l from the arc set E to the reals is the *length function*. Sometimes, it is convenient to consider arcs not in E to be arcs of infinite length in G. Therefore, a net is a triplet and can be compactly written as $G(V, E, l)$. To simplify the notation, we adopt the following convention for any nonempty subgraph $g(V_\alpha, E_\alpha)$ of G:

$$l(g) = \sum_{(i,j)\in E_\alpha} l(i, j) \tag{2.1}$$

Thus, if P_{st} is a directed path from node s to node t in G, then $l(P_{st})$ is the *length* of the directed path P_{st}. This length is different from that defined in Definition 1.49. To distinguish, sometimes the length in Definition 1.49 is referred to as the *arc length*. This should not create any confusion as the context will reveal. This leads to the notion of distance.

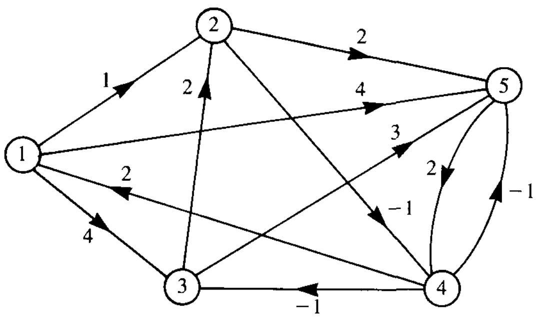

Fig. 2.1. A net $G(V, E, l)$ used to illustrate the length function.

DEFINITION 2.1

Distance. The length of a shortest directed path from a node s to a node t in a net is called the *distance* from s to t in the net.

Fig. 2.1 is a net $G(V, E, l)$ with node set V and arc set E given as

$$V = \{1, 2, 3, 4, 5\} \tag{2.2a}$$

$$\begin{aligned} E = \{&(1, 2), (1, 3), (1, 5), (2, 4), (2, 5), (3, 2), (3, 5),\\ &(4, 1), (4, 3), (4, 5), (5, 4)\} \end{aligned} \tag{2.2b}$$

The length function l from E to the reals is defined as

$$\begin{aligned} &l(1, 2) = 1, && l(1, 3) = 4, && l(1, 5) = 4,\\ &l(2, 4) = -1, && l(2, 5) = 2, && l(3, 2) = 2,\\ &l(3, 5) = 3, && l(4, 1) = 2, && l(4, 3) = -1,\\ &l(4, 5) = -1, && l(5, 4) = 2 \end{aligned} \tag{2.3}$$

There are seven directed paths $P_{15}^{(k)}$ $(k = 1, 2, \ldots, 7)$ from node 1 to node 5, and they are found to be

$$P_{15}^{(1)} = (1, 2)(2, 5) \tag{2.4a}$$

$$P_{15}^{(2)} = (1, 2)(2, 4)(4, 5) \tag{2.4b}$$

$$P_{15}^{(3)} = (1, 2)(2, 4)(4, 3)(3, 5) \tag{2.4c}$$

$$P_{15}^{(4)} = (1, 5) \tag{2.4d}$$

$$P_{15}^{(5)} = (1, 3)(3, 2)(2, 5) \tag{2.4e}$$

$$P_{15}^{(6)} = (1, 3)(3, 2)(2, 4)(4, 5) \tag{2.4f}$$

$$P_{15}^{(7)} = (1, 3)(3, 5) \tag{2.4g}$$

the lengths of which are computed as

$$l(P_{15}^{(1)}) = l(1, 2) + l(2, 5) = 1 + 2 = 3 \tag{2.5a}$$

$$l(P_{15}^{(2)}) = l(1, 2) + l(2, 4) + l(4, 5) = 1 - 1 - 1 = -1 \tag{2.5b}$$

$$l(P_{15}^{(3)}) = l(1, 2) + l(2, 4) + l(4, 3) + l(3, 5) = 1 - 1 - 1 + 3 = 2 \tag{2.5c}$$

$$l(P_{15}^{(4)}) = l(1, 5) = 4 \tag{2.5d}$$

$$l(P_{15}^{(5)}) = l(1, 3) + l(3, 2) + l(2, 5) = 4 + 2 + 2 = 8 \tag{2.5e}$$

$$\begin{aligned} l(P_{15}^{(6)}) &= l(1, 3) + l(3, 2) + l(2, 4) + l(4, 5) \\ &= 4 + 2 - 1 - 1 = 4 \end{aligned} \tag{2.5f}$$

$$l(P_{15}^{(7)}) = l(1, 3) + l(3, 5) = 4 + 3 = 7 \tag{2.5g}$$

Thus, the directed path $P_{15}^{(2)}$ of length -1 is the shortest path from node 1 to node 5 in G. There is one directed path $P_{15}^{(3)}$ of length 2, one $P_{15}^{(1)}$ of length 3, two $P_{15}^{(4)}$ and $P_{15}^{(6)}$ of length 4, one $P_{15}^{(7)}$ of length 7, and one $P_{15}^{(5)}$ of length 8.

Consider the directed subpath $P_{12}^{(6)} = (1, 3)(3, 2)$ of the directed path $P_{15}^{(6)}$. The length of the subpath $P_{12}^{(6)}$ is 6 whereas the length of $P_{15}^{(6)}$ is 4. Thus, the length of a directed path need not always be longer than that of its subpath. However, if all the arcs are of positive length, then the length of a directed path is always longer than that of any of its subpaths.

Let P_{st} be a shortest directed path from s to t in $G(V, E, l)$. Then any of its subpaths must itself be a shortest directed path in G. To see this, suppose that P_{ij} is a subpath of P_{st} from node i to node j. If there is a shorter directed path $\hat{P}_{ij}$ from i to j in G, then by proceeding from s to i on P_{st}, following the shorter directed path $\hat{P}_{ij}$, and finally proceeding from j to t on P_{st}, we produce a directed path from s to t which is shorter than P_{st}, contradicting the assumption that P_{st} is the shortest directed path from s to t. Thus, any subpath of a shortest directed path must itself be a shortest directed path. Since any path in a directed graph of n nodes contains at most $n - 1$ arcs, any shortest directed path in an n-node net contains at most $n - 1$ arcs. Note that in most applications, we can replace an undirected edge between nodes i and j by two arcs of the same length, one from i to j and the other from j to i, denoted by (i, j) and (j, i). Thus, any undirected graph or mixed graph can be converted into a directed graph. In this sense, an undirected edge with negative length in an undirected or mixed graph is equivalent to a negative directed circuit in its associated directed graph. As in other methods for solving the shortest directed path problem, the algorithms described in this chapter also yield, on one application, the shortest directed paths from a given node to all other nodes of the net that can be reached from the given node. Therefore, we shall discuss the problem of finding the shortest directed paths from one node, called the *source node,* to all other nodes in a net. A more general problem is that of finding the shortest directed paths between all pairs of nodes. These and other related problems will be treated in the following.

2.2 SHORTEST DIRECTED PATH ALGORITHMS

In this section, we describe several simple and efficient combinatorial algorithms for finding the shortest directed paths from a given node to all other nodes of a net that does not contain any negative directed circuits. The first algorithm is valid for the special situation where the arc lengths of a net are all nonnegative. The remainder of the algorithms are general and can be used in any nets with positive, zero or negative arc lengths as long as they do not contain any negative directed circuits. The special situation is considered first because it occurs most frequently in practice and the algorithm is most efficient in relation to the more complicated general algorithms. It is sufficiently important to be considered separately.

2.2.1 Dijkstra Algorithm

The most efficient algorithm for finding the shortest directed paths from a given node s to all other nodes in a net $G(V, E, l)$ with nonnegative arc lengths

$$l(i, j) \geqq 0 \quad \text{for all } (i, j) \in E \tag{2.6}$$

was first given by Dijkstra (1959). The algorithm is based on assigning labels to the nodes. The labels are continuously updated by an iterative procedure. At each stage, the number assigned to each node represents an upper bound on the directed path length from s to that node. At the end of the procedure, the node label becomes the exact length of a shortest directed path from s to that node in question.

To proceed, let each node i of G be assigned a nonnegative real number $l(i)$ called the *label* of i. Initially, all the labels are temporary. At each iteration, one of the temporary labels is made permanent, say $l(j)$. To distinguish a temporary label from a permanent one, we use $l^*(j)$ to indicate that node j has received the permanent label $l^*(j)$. The details of the algorithm are described as follows: Arcs not in E are considered as arcs of infinite length.

Step 1. Set $l^*(s) = 0$ and $l(i) = l(s, i)$ for $i \neq s$ and $i \in V$, where $l(s, i) = \infty$ for $i \neq s$ and $(s, i) \notin E$.

Step 2. Among all temporary labels $l(i)$, pick

$$l(k) = \min_i l(i) \tag{2.7}$$

Change $l(k)$ to $l^*(k)$. Stop if there is no temporary label left.

Step 3. Update all temporary labels of the nodes i with $(k, i) \in E$ for all i by

$$l(i) = \min\,[l(i),\, l^*(k) + l(k, i)] \tag{2.8}$$

Return to Step 2.

To show that the algorithm indeed produces the shortest directed paths from the source node s to all other nodes, assume that all the shortest directed paths P_{si} from s to i $(i = 1, 2, \dots, n-1)$ in the n-node net $G(V, E, l)$ have been ordered in accordance with their lengths. Without loss of generality, we assume that all arc lengths are positive and write

$$l(P_{s1}) \leqq l(P_{s2}) \leqq \dots \leqq l(P_{sk}) \leqq \dots \leqq l(P_{s(n-1)}) \tag{2.9}$$

since we can always rename the nodes, if necessary. If P_{s1} contains more than one arc, then it contains a subpath which is shorter than P_{s1}, contradicting our assumption that P_{s1} is the shortest among all the shortest directed paths P_{si} $(i = 1, 2, \dots, n-1)$. Thus, P_{s1} contains only one arc. We claim that P_{sk} contains at most k arcs. Assume to the contrary that P_{sk} contains more than k arcs. Then there are at least k nodes j on P_{sk} other than s and k. Since all the arc lengths are positive, each of the subpaths from s to j in P_{sk} must be shorter than P_{sk}, and we would have k shortest directed paths shorter than P_{sk}, a contradiction. Thus, P_{sk} contains at most k arcs. We next demonstrate that the above algorithm will calculate $l(P_{s1})$ first, $l(P_{s2})$ second, . . . , until $l(P_{s(n-1)})$ is found.

Clearly, at the end of the first iteration in Step 2 we generate $l(P_{s1}) = l^*(1)$. To find P_{s2} we need consider only one-arc or two-arc directed paths, the minimum length of which must be P_{s2}. If P_{s2} is a two-arc directed path with the last arc being $(j, 2)$, $j \neq 1$, the subpath (s, j) of P_{s2} is shorter than P_{s2}, a contradiction. Thus, P_{s2} can be either a one-arc directed path or a two-arc directed path with the last arc being $(1, 2)$. In other words, $l(P_{s2})$ is generated by means of the operation (2.8) from the permanent labels $l^*(s)$ and $l^*(1)$, yielding $l^*(2) = l(P_{s2})$ at the end of the second iteration in Step 2.

To complete the proof by induction, we assume that at the end of the kth iteration $l(P_{sk})$ can be generated by means of (2.8) from the permanent labels $l^*(s)$, $l^*(1)$, $l^*(2), \dots, l^*(k-1)$, $1 < k < n-1$. We show that $l(P_{s(k+1)})$ can be generated by means of (2.8) from the permanent labels $l^*(s)$, $l^*(1), \dots, l^*(k)$. If not all the internal nodes of $P_{s(k+1)}$ are permanently labeled, let j, $j \leqq k$, be the first such node with temporary label in $P_{s(k+1)}$ from s to $k+1$. Since all the arcs are of positive length, the subpath from s to j in $P_{s(k+1)}$ is shorter than $P_{s(k+1)}$, implying that $P_{s(k+1)}$ is not the $(k+1)$th shortest directed path in (2.9), a contradiction. Thus, all the internal nodes of $P_{s(k+1)}$ are permanently labeled. This means that to find $P_{s(k+1)}$ we need only search for those directed paths composed of a sequence of arcs whose endpoints are all permanently labeled followed by one arc reaching a temporary labeled node, the minimum length of which is $P_{s(k+1)}$. If $(q, k+1)$ is the last arc in $P_{s(k+1)}$, then

$$l(P_{s(k+1)}) = l^*(q) + l(q, k+1) \tag{2.10}$$

where $q = s, 1, 2, \ldots, k-1$ or k, and the operation (2.8) will yield a temporary label equal to $l(P_{s(k+1)})$. A node with this temporary label is identified at the end of the $(k+1)$th iteration in Step 2, and its label is changed to $l^*(k+1)$.

In the case where a net contains arcs with zero length, we can first modify the net as follows: For each arc (i, j) of zero length, remove the arc (i, j) and then identify the nodes i and j with the combined node being designated as i. The shortest directed path from s to j in the original net can be obtained from that from s to i in the modified net by the insertion of the zero length arc (i, j). In fact, the algorithm remains valid even with the inclusion of zero length arcs.

Observe that in ordering the shortest directed paths P_{sk} $(k = 1, 2, \ldots, n-1)$ in accordance with (2.9), we use one arc to reach node 1, one or two arcs to reach node 2, and at most k arcs to reach node k. By the very nature of our search process, the union of all these shortest directed paths P_{sk} $(k = 1, 2, \ldots, n-1)$ is a tree of the net.

The algorithm consists of operations of comparisons and additions. We shall count these numbers. In Step 2, the first time there are $n-2$ comparisons; the second time, $n-3$ comparisons; so that there are a total

$$(n-2) + (n-3) + (n-4) + \ldots + 2 + 1 = \tfrac{1}{2}(n-1)(n-2) \tag{2.11}$$

comparisons. Likewise, in Step 3 there are $n-2$ additions and the same number of comparisons the first time, and $n-3$ additions and the same number of comparisons the second time, and so forth. Thus, there are $\frac{1}{2}(n-1)(n-2)$ additions and comparisons in Step 3. As a result, the algorithm requires $(n-1)(n-2)$ comparisons and $\frac{1}{2}(n-1)(n-2)$ additions for an n-node net, and therefore its computational complexity is $O(n^2)$.

After we have determined the lengths of the shortest directed paths from node s to all other nodes j $(j = 1, 2, \ldots, n-1)$ in the net, we have not yet identified the shortest directed paths themselves that will give the desired lengths. One way to track down these paths is as follows: Suppose that we wish to identify a shortest directed path from s to j. If node j is not permanently labeled, then there is no directed path from s to j. If j is permanently labeled, we look for all arcs $(i, j) \in E$ to determine a node i such that

$$l^*(i) + l(i, j) = l^*(j) \tag{2.12}$$

We then repeat the process for node i. In this way, we can track back from node j to node s, and identify a shortest directed path from s to j.

Another approach is to modify the algorithm by assigning two numbers to each node j as a label of the form $[i, l(j)]$, as follows:

Step 1. Set $l^*(s) = 0$ and $l(i) = l(s, i)$ for $i \neq s$ and $i \in V$, where $l(s, i) = \infty$ for $i \neq s$ and $(s, i) \notin E$, and node i is labeled as $[s, l(i)]$

Step 2. Among all temporary labels $l(i)$, pick

$$l(k) = \min_i l(i) \tag{2.13}$$

Change $l(k)$ to $l^*(k)$. Stop if there is no temporary label left.

Step 3. Update all temporary labels of the nodes i with $(k, i) \in E$ for all i by

$$l(i) = \min\,[l(i), l^*(k) + l(k, i)] \tag{2.14}$$

Change the label on node i to $[k, l(i)]$ if $l(i)$ is decreased. Otherwise, keep the same label. Return to Step 2.

We illustrate the above results by the following examples.

Example 2.1

Consider the 7-node net $G(V, E)$ of Fig. 2.2, in which each undirected edge between nodes i and j denotes a pair of arcs of the same length, one from i to j and the other from j to i. We shall apply the algorithm to generate the shortest paths from node s to all other nodes 1, 2, 3, 4, 5, 6, as follows. For simplicity, only updated temporary labels are listed. The others remain the same.

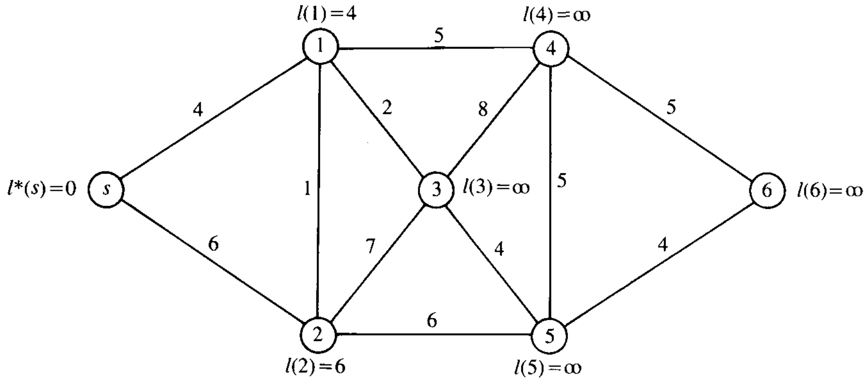

Fig. 2.2. A 7-node net $G(V, E)$ in which each undirected edge between nodes i and j denotes a pair of arcs of the same length, one from i to j and the other from j to i.

Step 1. Node s receives the permanent label $l^*(s) = 0$, and all other nodes receive the temporary labels indicated in Fig. 2.2.

First iteration

Step 2. Pick

$$\begin{aligned} l(1) &= \min [l(1), l(2), l(3), l(4), l(5), l(6)] \\ &= \min [4, 6, \infty, \infty, \infty, \infty] = 4 \end{aligned} \tag{2.15}$$

Thus, $k = 1$. Change $l(1)$ to $l^*(1) = 4$ for node 1.

Step 3.

$$l(i) = \min [l(i), l^*(1) + l(1, i)], \quad i = 2, 3, 4, 5, 6 \tag{2.16}$$

This gives

$$l(2) = \min [l(2), l^*(1) + l(1, 2)] = \min [6, 4 + 1] = 5 \tag{2.17a}$$

$$l(3) = \min [l(3), l^*(1) + l(1, 3)] = \min [\infty, 4 + 2] = 6 \tag{2.17b}$$

$$l(4) = \min [l(4), l^*(1) + l(1, 4)] = \min [\infty, 4 + 5] = 9 \tag{2.17c}$$

The labels of the resulting net are shown in Fig. 2.3.

Second iteration

Step 2. Pick

$$\begin{aligned} l(2) &= \min [l(2), l(3), l(4), l(5), l(6)] \\ &= \min [5, 6, 9, \infty, \infty] = 5 \end{aligned} \tag{2.18}$$

Thus, $k = 2$. Change $l(2)$ to $l^*(2) = 5$ for node 2.

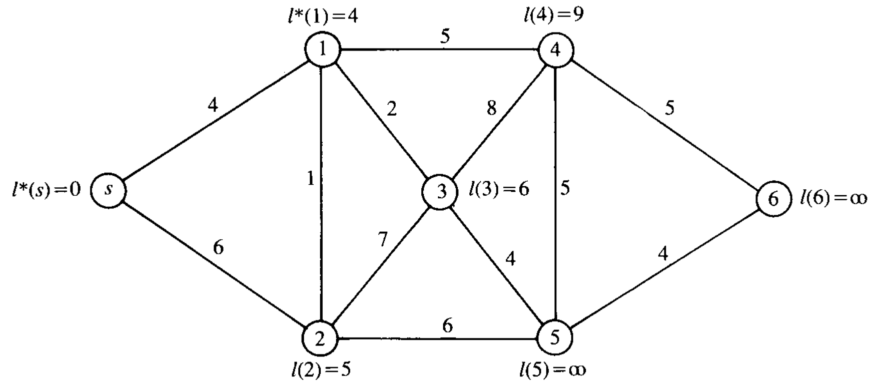

Fig. 2.3. The labeled net of Fig. 2.2 at the end of the first iteration of the Dijkstra algorithm.

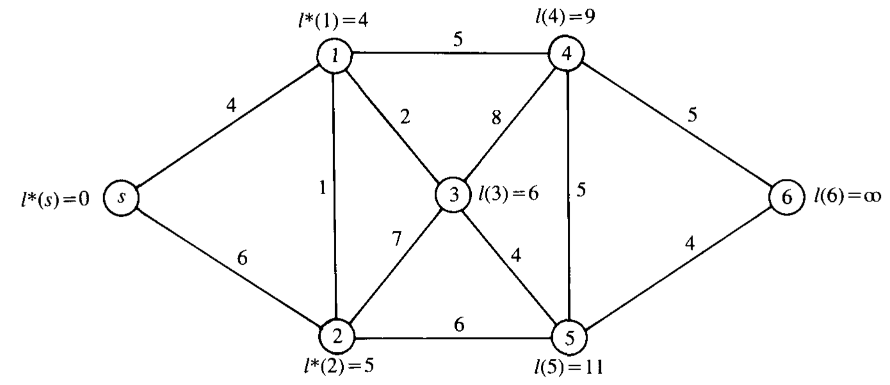

Fig. 2.4. The labeled net of Fig. 2.2 at the end of the second iteration of the Dijkstra algorithm.

Step 3.

$$l(i) = \min\,[l(i),\, l^*(2) + l(2,\, i)], \qquad i = 3, 4, 5, 6 \qquad (2.19)$$

This gives

$$l(3) = \min\,[l(3),\, l^*(2) + l(2,\, 3)] = \min\,[6, 5 + 7] = 6 \qquad (2.20a)$$

$$l(5) = \min\,[l(5),\, l^*(2) + l(2,\, 5)] = \min\,[\infty, 5 + 6] = 11 \qquad (2.20b)$$

The labels of the resulting net are shown in Fig. 2.4.

Third iteration

Step 2. Pick

$$\begin{aligned} l(3) &= \min\,[l(3),\, l(4),\, l(5),\, l(6)] \\ &= \min\,[6, 9, 11, \infty] = 6 \end{aligned} \qquad (2.21)$$

Thus, $k = 3$. Change $l(3)$ to $l^*(3) = 6$ for node 3.

Step 3.

$$l(i) = \min\,[l(i),\, l^*(3) + l(3,\, i)], \qquad i = 4, 5, 6 \qquad (2.22)$$

This gives

$$l(4) = \min\,[l(4),\, l^*(3) + l(3,\, 4)] = \min\,[9, 6 + 8] = 9 \qquad (2.23a)$$

$$l(5) = \min\,[l(5),\, l^*(3) + l(3,\, 5)] = \min\,[11, 6 + 4] = 10 \qquad (2.23b)$$

Fourth iteration

Step 2. Pick

$$l(4) = \min [l(4), l(5), l(6)]$$
$$= \min [9, 10, \infty] = 9 \tag{2.24}$$

Thus, $k = 4$. Change $l(4)$ to $l^*(4) = 9$ for node 4.

Step 3.

$$l(i) = \min [l(i), l^*(4) + l(4, i)], \qquad i = 5, 6 \tag{2.25}$$

This gives

$$l(5) = \min [l(5), l^*(4) + l(4, 5)] = \min [10, 9 + 5] = 10 \tag{2.26a}$$
$$l(6) = \min [l(6), l^*(4) + l(4, 6)] = \min [\infty, 9 + 5] = 14 \tag{2.26b}$$

Fifth iteration

Step 2. Pick

$$l(5) = \min [l(5), l(6)] = \min [10, 14] = 10 \tag{2.27}$$

Thus, $k = 5$. Change $l(5)$ to $l^*(5) = 10$ for node 5.

Step 3.

$$l(6) = \min [l(6), l^*(5) + l(5, 6)]$$
$$= \min [14, 10 + 4] = 14 \tag{2.28}$$

Sixth iteration

Step 2. Pick $l(6) = \min l(6) = 14$. Thus, $k = 6$. Change $l(6)$ to $l^*(6) = 14$. Stop.

The final labels of the nodes are shown in Fig. 2.5. Suppose that we wish to track down a shortest directed path P_{s6} from s to 6. We first look at all arcs $(i, 6) \in E$ to determine a node i such that

$$l^*(i) + l(i, 6) = l^*(6) = 14 \tag{2.29}$$

For $i = 4$ and 5 we have

$$l^*(4) + l(4, 6) = 9 + 5 = 14 = l^*(6) \tag{2.30a}$$
$$l^*(5) + l(5, 6) = 10 + 4 = 14 = l^*(6) \tag{2.30b}$$

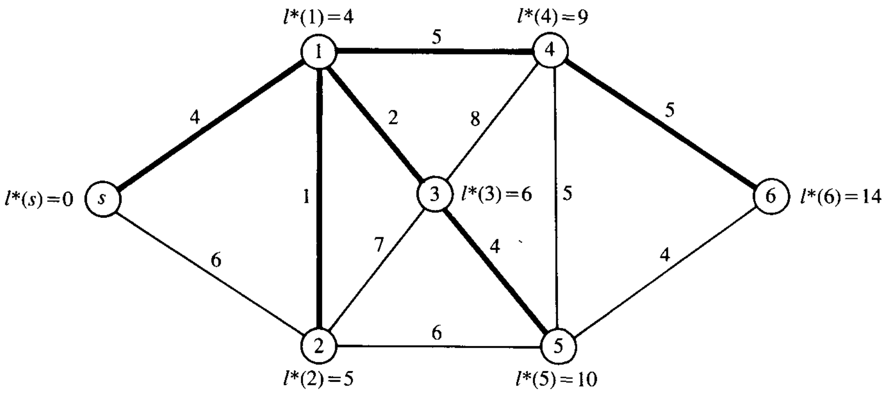

Fig. 2.5. The final labeled net of Fig. 2.2 after the application of the Dijkstra algorithm.

Thus, we can choose either $i = 4$ or $i = 5$. Let $i = 4$. We next consider all arcs $(i, 4) \in E$ to determine a node i such that

$$l^*(i) + l(i, 4) = l^*(4) = 9, \qquad i = 1, 3, 5 \tag{2.31}$$

identifying $i = 1$. We repeat this for node 1 with

$$l^*(i) + l(i, 1) = l^*(1) = 4, \qquad i = s, 2, 3 \tag{2.32}$$

giving $i = s$. Thus, a desired shortest directed path P_{s6} consists of the node sequence s, 1, 4, 6 or

$$P_{s6} = (s, 1)(1, 4)(4, 6) \tag{2.33}$$

Had we picked $i = 5$ in (2.30b), we would have generated the shortest directed path

$$P_{s6} = (s, 1)(1, 3)(3, 5)(5, 6) \tag{2.34}$$

Repeating this for all other nodes identifies all other shortest directed paths. The union of these paths forms a tree rooted at node s as shown in Fig. 2.6.

Alternatively, if we follow the modified procedure by assigning two numbers to a node, the final labeled net is shown in Fig. 2.7. To identify, for example, P_{s6} we first check the label of node 6, which is [4, 14*], showing that arc (4, 6) is in P_{s6}. We next check the label of node 4 which is [1, 9*], indicating that arc (1, 4) is in P_{s6}. Finally, the label of node 1 is [s, 4*], and we include (s, 1) in P_{s6}. Likewise, we can identify other directed paths, the union of which is a rooted tree shown in Fig. 2.6.

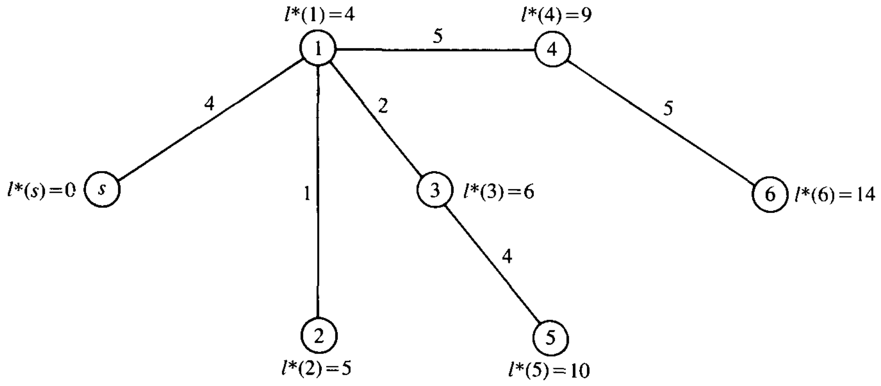

Fig. 2.6. A tree rooted at node s of the net of Fig. 2.2.

Example 2.2

Consider the net $G(V, E)$ of Fig. 2.8. We apply the modified algorithm to generate all the shortest directed paths P_{sk} ($k = 1, 2, \ldots, 13$) from s to k. Initially, node s is permanently labeled as $l^*(s) = 0$, node 1 is labeled by $[s, 1]$, node 4 by $[s, 2]$, and all other nodes are labeled by $[s, \infty]$, where $l(1) = 1$, $l(4) = 2$ and $l(j) = \infty$ for $j \neq s$, 1 and 4.

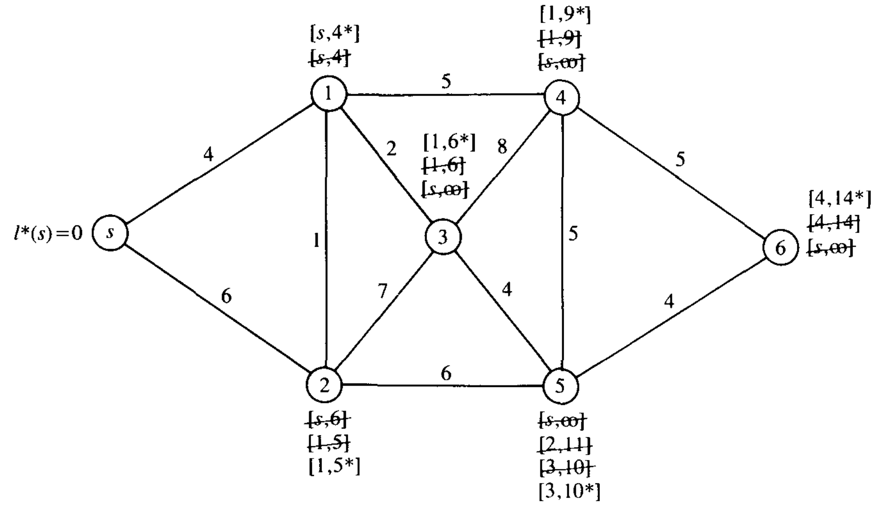

Fig. 2.7. The final labeled net of Fig. 2.2 after the application of the modified Dijkstra algorithm by assigning two numbers to a node.

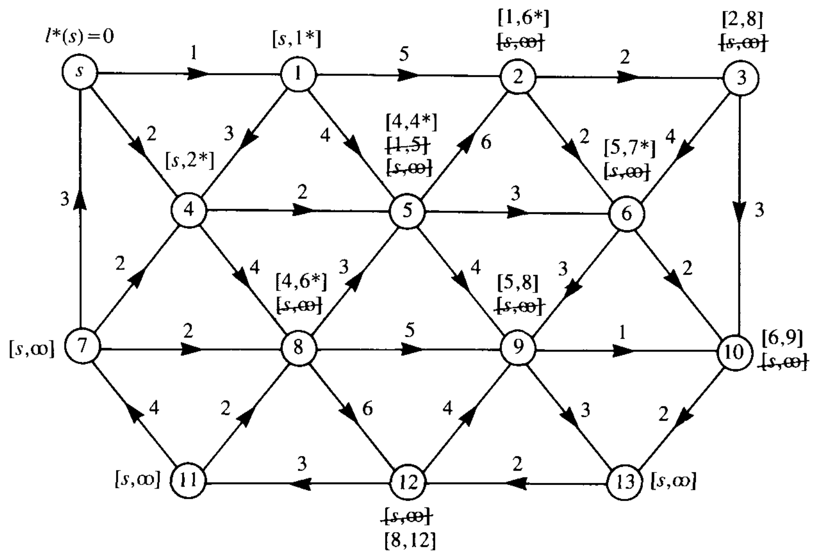

Fig. 2.8. A net $G(V, E)$ used to illustrate the modified Dijkstra algorithm with the labels showing the first six iterations.

First iteration

Step 2. Pick

$$l(1) = \min_i l(i) = 1 \tag{2.35}$$

Change $l(1)$ in $[s, l(1)]$ to $[s, l^*(1)]$ or $[s, 1^*]$.

Step 3.

$$l(i) = \min [l(i), l^*(1) + l(1, i)], \qquad i = 2, 4, 5 \tag{2.36}$$

This gives $l(2) = 6$, $l(4) = 2$ and $l(5) = 5$. The labels of nodes 2, 4, and 5 are changed to $[1, 6]$, $[s, 2]$, and $[1, 5]$, respectively.

Second iteration

Step 2. Pick

$$l(4) = \min_i l(i) = 2 \tag{2.37}$$

Change $l(4)$ in $[s, l(4)]$ to $[s, l^*(4)]$ or $[s, 2^*]$.

Step 3.

$$l(i) = \min\,[l(i), l^*(4) + l(4, i)], \qquad i = 5, 8 \tag{2.38}$$

This gives $l(5) = 4$ and $l(8) = 6$. The labels of nodes 5 and 8 are changed to $[4, 4]$ and $[4, 6]$, respectively.

These steps will now be repeated for the nodes in the following order: 5, 2, 8, 6, 3, 9, 10, 13, 12, 11, and 7. The details of the operations of the algorithm are described in Table 2.1, where the underlined entry in the ith row signifies that the label of the node corresponding to the entry is changed from a temporary label to a permanent one at the ith iteration. The first six iterations are indicated in Fig. 2.8, and the last seven are given in Fig. 2.9. The tree formed by the 13 shortest directed paths P_{sj} $(j = 1, 2, \dots, 13)$ from s to j in the net is shown in Fig. 2.10.

2.2.2 Ford–Moore–Bellman Algorithm

The Dijkstra algorithm described in the preceding section is valid only for those nets whose arc lengths are nonnegative. In this section, we consider the general situation where the arc lengths may be positive, zero or negative. As mentioned before, in order for the shortest directed path problem to be meaningfully defined, we assume that the nets do not contain any negative directed circuits. The algorithm was originally proposed by Ford (1956), Moore (1957) and Bellman (1958), and is based on the notion that a shortest directed path from node s to node j containing at most $k + 1$ arcs can be obtained from those from s to j containing at most k arcs. Thus, at the end of the kth iteration the labels of the nodes represent the lengths of those shortest directed paths from s containing $k + 1$ or fewer arcs.

Let $l_j^{(k)}$ denote the length of a shortest directed path $P_{sj}^{(k)}$ from s to j in a given n-node net $G(V, E, l)$ using at most k arcs. To initialize the algorithm, set $l_s = l_s^{(k)} = 0$ and

$$l_j^{(1)} = l(s, j), \qquad j = 1, 2, \dots, n - 1 \tag{2.39}$$

It is convenient to define $l(i, j) = \infty$ for $(i, j) \notin E$, $i \neq j$. When $l_j^{(k)}$ $(j = 1, 2, \dots, n - 1)$ are all known, $l_j^{(k+1)}$ can be calculated by the following recurrence relation: For all $(i, j) \in E$

$$l_j^{(k+1)} = \min\,\{l_j^{(k)}, \min_i\,[l_i^{(k)} + l(i, j)]\}, \qquad i \neq j, s; \quad j = 1, 2, \dots, n - 1 \tag{2.40}$$

Observe that a shortest directed path $P_{sj}^{(k+1)}$ from s to j using at most $k + 1$ arcs may consist of $k + 1$ arcs or less. If it uses exactly $k + 1$ arcs, let

TABLE 2.1

Nodes												
1	2	3	4	5	6	7	8	9	10	11	12	13
$\underline{s, 1^*}$	s, ∞	s, ∞	$s, 2$	s, ∞	s, ∞	s, ∞	s, ∞	s, ∞	s, ∞	s, ∞	s, ∞	s, ∞
$s, 1^*$	$1, 6$	s, ∞	$\underline{s, 2^*}$	$1, 5$	s, ∞	s, ∞	s, ∞	s, ∞	s, ∞	s, ∞	s, ∞	s, ∞
$s, 1^*$	$1, 6$	s, ∞	$s, 2^*$	$\underline{4, 4^*}$	s, ∞	s, ∞	$4, 6$	s, ∞	s, ∞	s, ∞	s, ∞	s, ∞
$s, 1^*$	$\underline{1, 6^*}$	s, ∞	$s, 2^*$	$4, 4^*$	$5, 7$	s, ∞	$4, 6$	$5, 8$	s, ∞	s, ∞	s, ∞	s, ∞
$s, 1^*$	$1, 6^*$	$2, 8$	$s, 2^*$	$4, 4^*$	$5, 7$	s, ∞	$\underline{4, 6^*}$	$5, 8$	s, ∞	s, ∞	s, ∞	s, ∞
$s, 1^*$	$1, 6^*$	$2, 8$	$s, 2^*$	$4, 4^*$	$\underline{5, 7^*}$	s, ∞	$4, 6^*$	$5, 8$	s, ∞	s, ∞	$8, 12$	s, ∞
$s, 1^*$	$1, 6^*$	$\underline{2, 8^*}$	$s, 2^*$	$4, 4^*$	$5, 7^*$	s, ∞	$4, 6^*$	$5, 8$	$6, 9$	s, ∞	$8, 12$	s, ∞
$s, 1^*$	$1, 6^*$	$2, 8^*$	$s, 2^*$	$4, 4^*$	$5, 7^*$	s, ∞	$4, 6^*$	$\underline{5, 8^*}$	$6, 9$	s, ∞	$8, 12$	s, ∞
$s, 1^*$	$1, 6^*$	$2, 8^*$	$s, 2^*$	$4, 4^*$	$5, 7^*$	s, ∞	$4, 6^*$	$5, 8^*$	$\underline{6, 9^*}$	s, ∞	$8, 12$	$9, 11$
$s, 1^*$	$1, 6^*$	$2, 8^*$	$s, 2^*$	$4, 4^*$	$5, 7^*$	s, ∞	$4, 6^*$	$5, 8^*$	$6, 9^*$	s, ∞	$8, 12$	$\underline{9, 11^*}$
$s, 1^*$	$1, 6^*$	$2, 8^*$	$s, 2^*$	$4, 4^*$	$5, 7^*$	s, ∞	$4, 6^*$	$5, 8^*$	$6, 9^*$	s, ∞	$\underline{8, 12^*}$	$9, 11^*$
$s, 1^*$	$1, 6^*$	$2, 8^*$	$s, 2^*$	$4, 4^*$	$5, 7^*$	s, ∞	$4, 6^*$	$5, 8^*$	$6, 9^*$	$\underline{12, 15^*}$	$8, 12^*$	$9, 11^*$
$s, 1^*$	$1, 6^*$	$2, 8^*$	$s, 2^*$	$4, 4^*$	$5, 7^*$	$\underline{11, 19^*}$	$4, 6^*$	$5, 8^*$	$6, 9^*$	$12, 15^*$	$8, 12^*$	$9, 11^*$

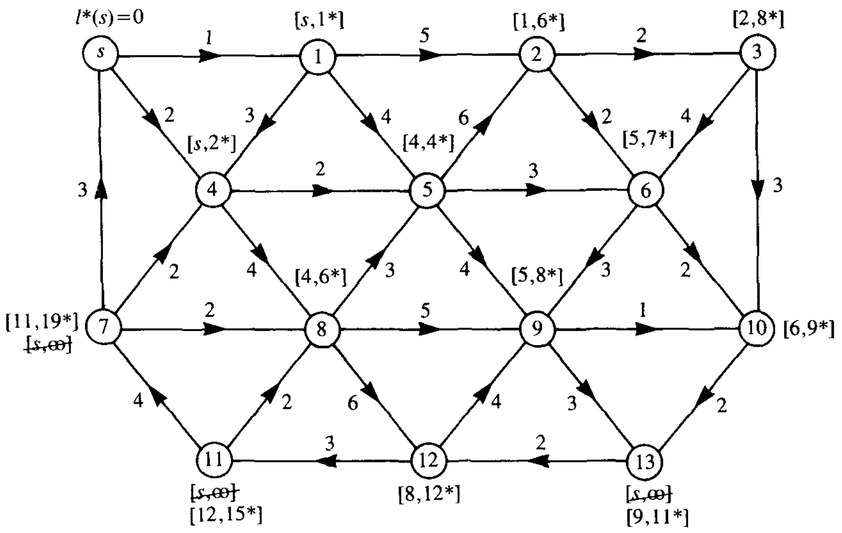

Fig. 2.9. The net used to indicate the labels of the last seven iterations after applying the modified Dijkstra algorithm to the net of Fig. 2.8.

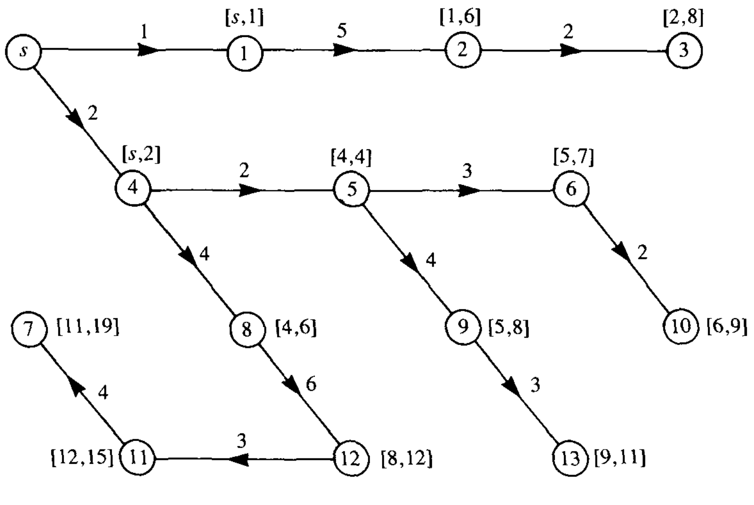

Fig. 2.10. The tree formed by the 13 shortest directed paths P_{sj} from s to j in the net of Fig. 2.8.

(i, j) be the last arc in $P_{sj}^{(k+1)}$. Then, $P_{sj}^{(k+1)}$ can be viewed as a k-arc shortest directed path $P_{si}^{(k)}$ from s to i followed by the arc (i, j). Thus, by considering all is we obtain the length of $P_{sj}^{(k+1)}$ as

$$l_j^{(k+1)} = l(P_{sj}^{(k+1)}) = \min_i \, [l_i^{(k)} + l(i, j)] \tag{2.41}$$

On the other hand, if $P_{sj}^{(k+1)}$ contains k arcs or less, then

$$l_j^{(k+1)} = l_j^{(k)} \tag{2.42}$$

Thus, by picking the smaller of (2.41) and (2.42) we arrive at equation (2.40). It is easy to see that the algorithm terminates when

$$l_j^{(k+1)} = l_j^{(k)}, \qquad j = 1, 2, \ldots, n-1 \tag{2.43}$$

for $k \leqq n-1$. If $l_j^{(k+1)} \neq l_j^{(k)}$ for some j when $k = n-1$, then the net must contain a negative directed circuit. Thus, the existence of a negative directed circuit will be detected by the failure of the algorithm to terminate at the $(n-1)$th iteration.

Observe that the algorithm consists of comparisons and additions. We shall count the numbers. After initialization, for each j with a fixed k, (2.40) requires at most $n-2$ additions and the same number of comparisons. Since there are $n-1$ nodes to be examined, for each iteration the algorithm needs to perform at most $(n-1)(n-2)$ additions and the same number of comparisons. Finally, there are at most $n-1$ iterations, and therefore the computational complexity of the algorithm is $O(n^3)$. This is to be compared with the computational complexity required for the Dijkstra algorithm, which is $O(n^2)$. Thus, the Dijkstra algorithm is most efficient in relation to the Ford–Moore–Bellman algorithm. However, the Dijkstra algorithm is valid only for nets with nonnegative arc lengths, whereas the Ford–Moore–Bellman algorithm is more general and is applicable to any nets with positive, negative or zero arc lengths. In the latter case, no negative directed circuit is allowed.

The above counting of additions and comparisons required for the algorithm considers the worst case where the directed graph is nearly complete. In many practical situations, the nets are sparse with missing arcs. Also, as the above example demonstrated, the algorithm may terminate before $k = n-1$ is reached. Thus, if b is the number of arcs of a net, and q is the number of iterations required for termination or the maximum number of arcs contained in any shortest directed path, then for each iteration the algorithm requires at most b additions and the same number of comparisons. Therefore, the computational complexity of the algorithm can also be written as $O(qb)$. For problems with $b \ll n^2$ and $q \ll n$, the required computation $O(qb)$ can be less than the $O(n^2)$ requirement of the Dijkstra algorithm.

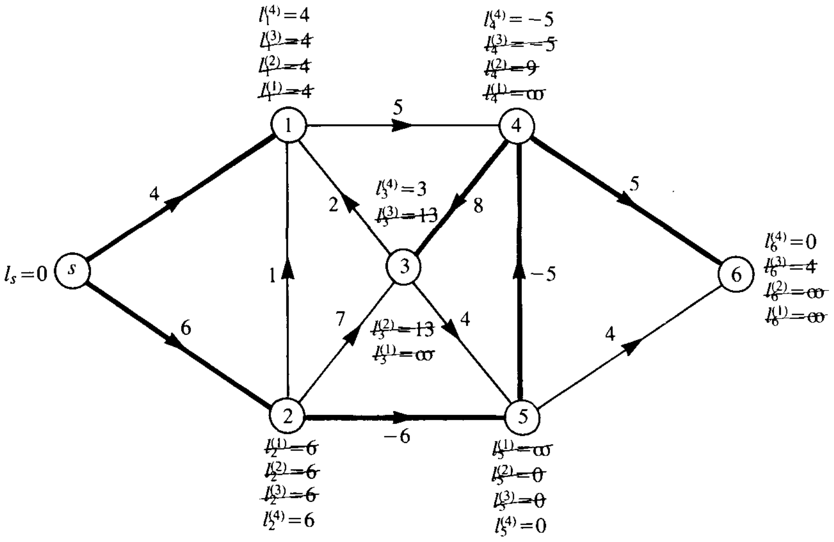

Fig. 2.11. A 7-node net $G(V, E)$ containing negative arc lengths used to illustrate the Ford–Moore–Bellman algorithm.

Example 2.3

Consider the 7-node net $G(V, E)$ of Fig. 2.11 where some of the arcs are of negative length. We apply the algorithm to generate the shortest directed paths P_{sj} $(j = 1, 2, \dots, 6)$ from s to j, as follows: Initialization

$$l_1^{(1)} = 4, \qquad l_2^{(1)} = 6, \qquad l_i^{(1)} = \infty \quad \text{for } i = 3, 4, 5, 6 \tag{2.44}$$

First iteration $(k = 1)$

$$l_1^{(2)} = \min\{l_1^{(1)}, \min[l_2^{(1)} + l(2, 1), l_3^{(1)} + l(3, 1)]\}$$

$$= \min\{4, \min[6 + 1, \infty + 2]\} = 4 \tag{2.45a}$$

$$l_2^{(2)} = \min\{l_2^{(1)}\} = l_2^{(1)} = 6 \tag{2.45b}$$

$$l_3^{(2)} = \min\{l_3^{(1)}, \min[l_2^{(1)} + l(2, 3), l_4^{(1)} + l(4, 3)]\}$$

$$= \min\{\infty, \min[6 + 7, \infty + 8]\} = 13 \tag{2.45c}$$

$$l_4^{(2)} = \min\{l_4^{(1)}, \min[l_1^{(1)} + l(1, 4), l_5^{(1)} + l(5, 4)]\}$$

$$= \min\{\infty, \min[4 + 5, \infty - 5]\} = 9 \tag{2.45d}$$

$$l_5^{(2)} = \min\{l_5^{(1)}, \min[l_2^{(1)} + l(2, 5), l_3^{(1)} + l(3, 5)]\}$$

$$= \min\{\infty, \min[6 - 6, \infty + 4]\} = 0 \tag{2.45e}$$

$$l_6^{(2)} = \min\{l_6^{(1)}, \min[l_4^{(1)} + l(4, 6), l_5^{(1)} + l(5, 6)]\}$$

$$= \min\{\infty, \min[\infty + 5, \infty + 4]\} = \infty \tag{2.45f}$$

Second iteration $(k = 2)$

$$l_1^{(3)} = \min\{l_1^{(2)}, \min[l_2^{(2)} + l(2,1), l_3^{(2)} + l(3,1)]\}$$

$$= \min\{4, \min[6+1, 13+2]\} = 4 \tag{2.46a}$$

$$l_2^{(3)} = \min\{l_2^{(2)}\} = l_2^{(2)} = 6 \tag{2.46b}$$

$$l_3^{(3)} = \min\{l_3^{(2)}, \min[l_2^{(2)} + l(2,3), l_4^{(2)} + l(4,3)]\}$$

$$= \min\{13, \min[6+7, 9+8]\} = 13 \tag{2.46c}$$

$$l_4^{(3)} = \min\{l_4^{(2)}, \min[l_1^{(2)} + l(1,4), l_5^{(2)} + l(5,4)]\}$$

$$= \min\{9, \min[4+5, 0-5]\} = -5 \tag{2.46d}$$

$$l_5^{(3)} = \min\{l_5^{(2)}, \min[l_2^{(2)} + l(2,5), l_3^{(2)} + l(3,5)]\}$$

$$= \min\{0, \min[6-6, 13+4]\} = 0 \tag{2.46e}$$

$$l_6^{(3)} = \min\{l_6^{(2)}, \min[l_4^{(2)} + l(4,6), l_5^{(2)} + l(5,6)]\}$$

$$= \min\{\infty, \min[9+5, 0+4]\} = 4 \tag{2.46f}$$

Third iteration $(k = 3)$

$$l_1^{(4)} = \min\{l_1^{(3)}, \min[l_2^{(3)} + l(2,1), l_3^{(3)} + l(3,1)]\} = 4 \tag{2.47a}$$

$$l_2^{(4)} = \min\{l_2^{(3)}\} = l_2^{(3)} = 6 \tag{2.47b}$$

$$l_3^{(4)} = \min\{l_3^{(3)}, \min[l_2^{(3)} + l(2,3), l_4^{(3)} + l(4,3)]\}$$

$$= \min\{13, \min[6+7, -5+8]\} = 3 \tag{2.47c}$$

$$l_4^{(4)} = \min\{l_4^{(3)}, \min[l_1^{(3)} + l(1,4), l_5^{(3)} + l(5,4)]\}$$

$$= \min\{-5, \min[4+5, 0-5]\} = -5 \tag{2.47d}$$

$$l_5^{(4)} = \min\{l_5^{(3)}, \min[l_2^{(3)} + l(2,5), l_3^{(3)} + l(3,5)]\}$$

$$= \min\{0, \min[6-6, 13+4]\} = 0 \tag{2.47e}$$

$$l_6^{(4)} = \min\{l_6^{(3)}, \min[l_4^{(3)} + l(4,6), l_5^{(3)} + l(5,6)]\}$$

$$= \min\{4, \min[-5+5, 0+4]\} = 0 \tag{2.47f}$$

Fourth iteration $(k = 4)$

$$l_j^{(5)} = \min\{l_j^{(4)}, \min_i [l_i^{(4)} + l(i,j)]\} = l_j^{(4)}, \qquad j = 1, 2, \ldots, 6 \tag{2.48}$$

The algorithm terminates at the end of the fourth iteration with $k = 4 \leqq n - 1 = 6$. The length of each shortest directed path is as indicated in Fig. 2.11 without the slashes. To identify a shortest directed path P_{sj} from s to j, we follow (2.12). For example, to find P_{s6} we first look for all arcs $(i, 6) \in E$ so that

$$l_6^{(4)} - l_i^{(4)} = l(i, 6), \qquad i = 4, 5 \tag{2.49}$$

or

$$l_6^{(4)} - l_4^{(4)} = 0 - (-5) = 5 = l(4, 6) \tag{2.50a}$$

$$l_6^{(4)} - l_5^{(4)} = 0 - 0 = 0 \neq l(5, 6) = 4 \tag{2.50b}$$

Thus, the arc $(4, 6)$ is in P_{s6}. We next consider the node 4 with

$$l_4^{(4)} - l_i^{(4)} = l(i, 4), \qquad i = 1, 5 \tag{2.51}$$

or

$$l_4^{(4)} - l_1^{(4)} = -5 - 4 = -9 \neq l(1, 4) = 5 \tag{2.52a}$$

$$l_4^{(4)} - l_5^{(4)} = -5 - 0 = -5 = l(5, 4) \tag{2.52b}$$

This shows that the arc $(5, 4)$ is in P_{s6}. We then consider node 5 with

$$l_5^{(4)} - l_i^{(4)} = l(i, 5), \qquad i = 2, 3 \tag{2.53}$$

or

$$l_5^{(4)} - l_2^{(4)} = 0 - 6 = -6 = l(2, 5) \tag{2.54a}$$

$$l_5^{(4)} - l_3^{(4)} = 0 - 3 = -3 \neq l(3, 5) = 4 \tag{2.54b}$$

Thus, arc $(2, 5)$ is in P_{s6}. Finally, we consider node 2 with

$$l_2^{(4)} - l_s = 6 - 0 = 6 = l(s, 2) \tag{2.55}$$

and identifies the arc $(s, 2)$. A desired shortest directed path P_{s6} from s to 6 is found to be

$$P_{s6} = (s, 2)(2, 5)(5, 4)(4, 6) \tag{2.56}$$

Note that if more than one node satisfies the equation (2.49), (2.51) or (2.53), then either node is permitted. The resulting solution is not unique, and more than one shortest directed path exists. For the net of Fig. 2.11, only one shortest directed path exists from s to any particular node. The tree formed by the shortest directed paths P_{sj} ($j = 1, 2, \ldots, 6$) from s to j with root s is shown in Fig. 2.12.

Example 2.4

The net $G(V, E, l)$ of Fig. 2.13 is a weighted mixed graph. We use the algorithm to generate all the shortest directed paths P_{sj} ($j = 1, 2, \ldots, 13$)

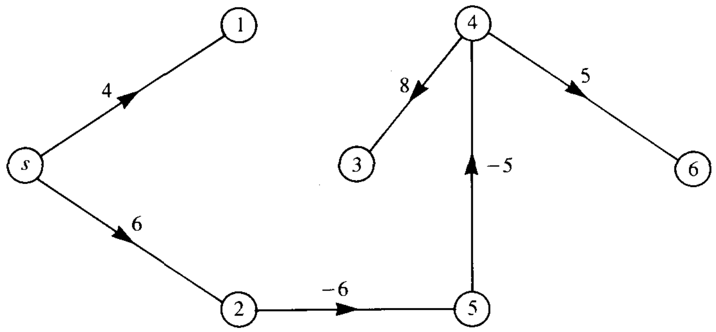

Fig. 2.12. The tree formed by the six shortest directed paths P_{sj} from s to j with root s in the net of Fig. 2.11.

from s to j, as follows:

Initialization ($k = 0$)

$$l_4^{(1)} = 2, \qquad l_i^{(1)} = \infty, \qquad i \neq 4 \tag{2.57}$$

First iteration ($k = 1$)

$$\begin{gathered} l_1^{(2)} = 5, \qquad l_4^{(2)} = 2, \qquad l_5^{(2)} = 0, \qquad l_7^{(2)} = 4 \\ l_8^{(2)} = 6, \qquad l_i^{(2)} = \infty, \qquad i = 2, 3, 6, 9, 10, 11, 12, 13 \end{gathered} \tag{2.58}$$

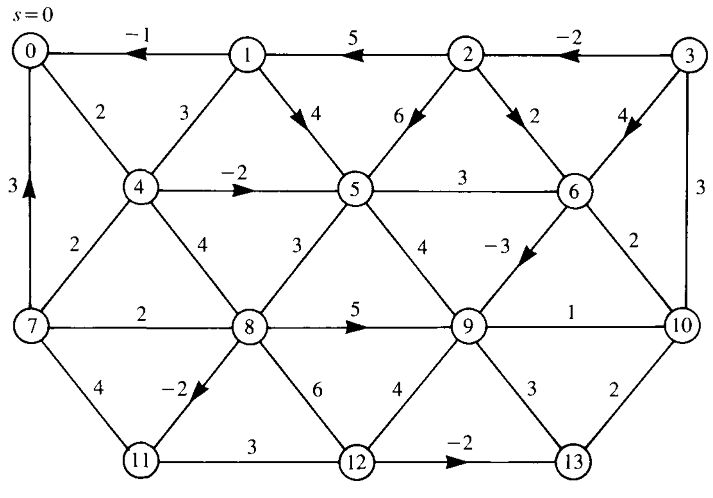

Fig. 2.13. A weighted mixed graph used to illustrate the Ford–Moore–Bellman algorithm.

TABLE 2.2

$k+1$ \ j	1	2	3	4	5	6	7	8	9	10	11	12	13
$l_j^{(1)}$	∞	∞	∞	2	∞	∞	∞	∞	∞	∞	∞	∞	∞
$l_j^{(2)}$	5	∞	∞	2	0	∞	4	6	∞	∞	∞	∞	∞
$l_j^{(3)}$	5	∞	∞	2	0	3	4	3	4	∞	4	12	∞
$l_j^{(4)}$	5	∞	∞	2	0	3	4	3	0	5	1	7	7
$l_j^{(5)}$	5	∞	8	2	0	3	4	3	0	1	1	4	3
$l_j^{(6)}$	5	6	4	2	0	3	4	3	0	1	1	4	2
$l_j^{(7)}$	5	2	4	2	0	3	4	3	0	1	1	4	2
$l_j^{(8)}$	5	2	4	2	0	3	4	3	0	1	1	4	2

Second iteration $(k = 2)$

$$l_1^{(3)} = 5, \quad l_4^{(3)} = 2, \quad l_5^{(3)} = 0, \quad l_6^{(3)} = 3$$
$$l_7^{(3)} = 4, \quad l_8^{(3)} = 3, \quad l_9^{(3)} = 4, \quad l_{11}^{(3)} = 4$$
$$l_{12}^{(3)} = 12, \quad l_i^{(3)} = \infty, \quad i = 2, 3, 10, 13 \tag{2.59}$$

The results of other iterations together with the first two are shown in Table 2.2 where the columns represent nodes j and the rows represent $l_j^{(k+1)}$.

Observe from Table 2.2 that for $k = 7$ we have

$$l_j^{(8)} = l_j^{(7)}, \qquad j = 1, 2, \ldots, 13 \tag{2.60}$$

Thus, the algorithm terminates at the end of the seventh iteration, and the entries in the last row or next to the last row give the lengths of the shortest directed paths P_{sj} $(j = 1, 2, \ldots, 13)$ from s to j. To find the shortest directed paths P_{sj}, we need to identify the node i of the last arc (i, j) in P_{sj}. The results are presented in Table 2.3, from which a shortest

TABLE 2.3

	1	2	3	4	5	6	7	8	9	10	11	12	13
Length from s to j	5	2	4	2	0	3	4	3	0	1	1	4	2
Last node i	4	3	10	s	4	5	4	5	6	9	8	9 or 11	12

directed path P_{sj} can be tracked back to s from j. For example, to find P_{s13} the table shows that its next to last node 13 is node 12. For node 12, its next to last node 12 is node 9 or 11. Continuing this process yields one of the following node sequences:

$$13, 12, 9, 6, 5, 4, s \quad \text{or} \quad 13, 12, 11, 8, 5, 4, s \tag{2.61a}$$

which correspond to the shortest directed path

$$P_{s13} = (s, 4)(4, 5)(5, 6)(6, 9)(9, 12)(12, 13) \tag{2.61b}$$

or

$$P_{s13} = (s, 4)(4, 5)(5, 8)(8, 11)(11, 12)(12, 13) \tag{2.61c}$$

The final labels of the nodes and all the shortest directed paths P_{sj} are indicated in Fig. 2.14, where the tree formed by P_{sj} is denoted by the heavy lines. Each label consists of two real numbers except node s, the first of which represents the last node i to node j and the second the length of P_{sj}.

2.2.3 Yen Algorithm

The Yen algorithm is the same as the Ford–Moore–Bellman algorithm except that it has a different interpretation of the symbol $l_j^{(k)}$, and requires roughly half the computations.

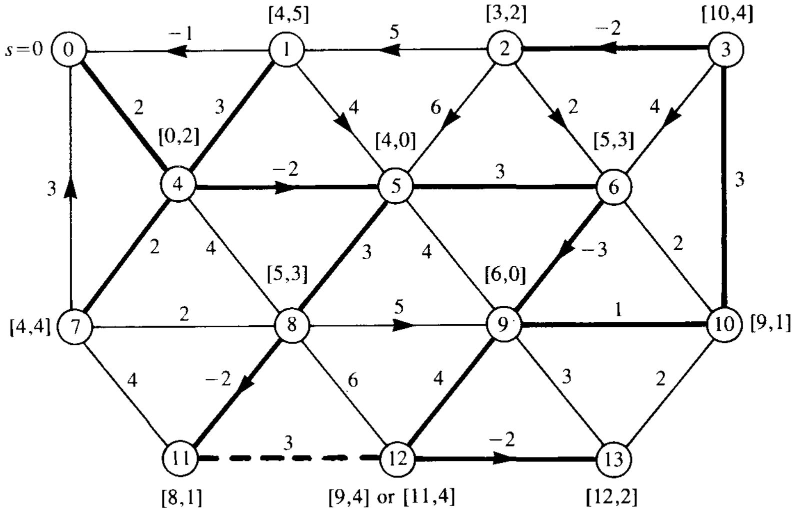

Fig. 2.14. The final labels of the nodes and all the shortest directed paths P_{sj} from s to j in the net of Fig. 2.13.

In a given net $G(V, E, l)$, let the source node s be designated as node 0, and the other $n-1$ nodes be assigned the integers $1, 2, \ldots, n-1$. Consider a k-arc directed path

$$P_{si_w} = (0, i_1)(i_1, i_2)\ldots(i_{p-1}, i_p)(i_p, i_{p+1})\ldots(i_{q-1}, i_q)\ldots(i_{w-1}, i_w) \quad (2.62)$$

from node 0 to node i_w in G. This path can also be expressed as a sequence of nodes from 0 to i_w along the directed path as

$$\{0, i_1, i_2, \ldots, i_{p-1}, i_p, i_{p+1}, \ldots, i_{q-1}, i_q, \ldots, i_w\} \quad (2.63)$$

A subsequence of this sequence is called a *block* if it is maximal and is either monotonically increasing or decreasing. For example, the subsequence $\{0, i_1, i_2, \ldots, i_p\}$ is a block if $i_{x-1} < i_x$ $(x = 2, 3, \ldots, p)$ and $i_p > i_{p+1}$. Likewise, the subsequence $\{i_p, i_{p+1}, \ldots, i_q\}$ is a block if $i_{x-1} > i_x$ $(x = p+1, p+2, \ldots, q)$ and $i_q < i_{q+1}$. Thus, the sequence (2.62) can be decomposed into, say, m overlapping blocks. We say that the directed path P_{si_w} consists of m blocks. The sequence

$$\{0, 2, 3, 7, 4, 1, 5, 6, 9\} \quad (2.64)$$

for example, is uniquely decomposable into three blocks: $\{0, 2, 3, 7\}$, $\{7, 4, 1\}$, and $\{1, 5, 6, 9\}$. With the notion of block, we can now attach a new interpretation to the symbol $l_j^{(k)}$.

Recall that in the preceding section the symbol $l_j^{(k)}$ denotes the length of a shortest directed path from s to j in G using at most k arcs. We now use the same symbol $l_j^{(k)}$ to mean the length of a shortest directed path from node 0 to node j in G with at most k blocks. Since a block may contain any number of nodes, a directed path $P_{sj}^{(k)}$ from 0 to j with at most k blocks may contain as many as $n-1$ arcs. In fact, $l_j^{(1)}$ may represent the length of a shortest directed path from 0 to j containing $n-1$ arcs. In this case, the nodes of the directed path increase from 0 to j. On the other hand, a directed path in an n-node net can at most contain $n-1$ blocks.

With these preliminaries, we now proceed to describe a modified Ford–Moore–Bellman algorithm, first proposed by Yen (1970). Define

$$l_j^{(0)} = l(0, j), \qquad j = 1, 2, \ldots, n-1 \quad (2.65)$$

and use $\{P_{sj}^{(k)}\}$ to denote the set of all directed paths $P_{sj}^{(k)}$ from 0 to j with at most k blocks. Then for $j = 1$ and $k = 1$ we have

$$l_1^{(1)} = l_1^{(0)} = l(0, 1) \quad (2.66)$$

and $\{P_{s1}^{(1)}\}$ consists only of one element $P_{s1}^{(1)} = (0, 1)$. For $j = 2$ and $k = 1$ two single blocks can be formed from the nonnegative integers 0, 1 and 2. They are 0, 2 and 0, 1, 2. For the block 0, 2, the corresponding directed path is the arc $(0,2)$. For the block 0, 1, 2, the corresponding directed path consists of the arcs $(0,1)$ and $(1,2)$. The shorter of these two path lengths is $l_2^{(1)}$ or

$$\begin{aligned} l_2^{(1)} &= \min\{l(0,2),\, l_1^{(1)} + l(1,2)\} \\ &= \min\{l_2^{(0)},\, \min_{i<2}[l_i^{(1)} + l(i,2)]\} \end{aligned} \tag{2.67}$$

where

$$\{P_{s2}^{(1)}\} = \{(0,2),\ (0,1)(1,2)\} \tag{2.68}$$

For $j = 3$, $k = 1$, the single blocks that can be formed from 0, 1, 2, 3 are 0, 3; 0, 1, 3; 0, 2, 3; and 0, 1, 2, 3. We can write from (2.65) and (2.67)

$$\begin{aligned} l_3^{(1)} &= \min\{l(0,3),\, l_1^{(1)} + l(1,3),\, l(0,2) + l(2,3),\, l_1^{(1)} + l(1,2) + l(2,3)\} \\ &= \min\{l_3^{(0)},\, l_1^{(1)} + l(1,3),\, l_2^{(1)} + l(2,3)\} \\ &= \min\{l_3^{(0)},\, \min_{i<3}[l_i^{(1)} + l(i,3)]\} \end{aligned} \tag{2.69}$$

where

$$\{P_{s3}^{(1)}\} = \{(0,3),\ (0,1)(1,3),\ (0,2)(2,3),\ (0,1)(1,2)(2,3)\} \tag{2.70}$$

Thus, knowing $l_1^{(1)}$ we can calculate $l_2^{(1)}$, and knowing $l_2^{(1)}$ we can calculate $l_3^{(1)}$. The above procedure can be generalized to yield the expression

$$l_j^{(1)} = \min\{l_j^{(0)},\, \min_{i<j}[l_i^{(1)} + l(i,j)]\}, \qquad j = 2, 3, \dots, n-1 \tag{2.71}$$

Likewise, for $j = n-1$ and $k = 2$ define

$$l_{n-1}^{(2)} = l_{n-1}^{(1)} \tag{2.72}$$

and for $j = n-2$ and $k = 2$, a two-block sequence consists of an increasing sequence from 0 to $n-1$ followed by the decreasing sequence $n-1$, $n-2$. Since $l_{n-1}^{(1)}$ denotes the length of a shortest directed path from 0 to $n-1$ in G with at most one block, $l_{n-2}^{(2)}$ can be written as

$$\begin{aligned} l_{n-2}^{(2)} &= \min\{l_{n-2}^{(1)},\, l_{n-1}^{(1)} + l(n-1, n-2)\} \\ &= \min\{l_{n-2}^{(1)},\, \min_{i>n-2}[l_i^{(2)} + l(i, n-2)]\} \end{aligned} \tag{2.73}$$

For $j = n-3$ and $k = 2$, a two-block sequence consists of an increasing sequence from 0 to $n-1$ followed by a decreasing sequence $n-1$, $n-2$, $n-3$ or $n-1$, $n-3$; or from 0 to $n-2$ followed by $n-2$, $n-3$. In either case, we can write from (2.72) and (2.73)

$$\begin{aligned} l_{n-3}^{(2)} &= \min\{l_{n-3}^{(1)}, l_{n-1}^{(1)} + l(n-1, n-3), l_{n-2}^{(1)} + l(n-2, n-3), \\ &\quad l_{n-1}^{(1)} + l(n-1, n-2) + l(n-2, n-3)\} \\ &= \min\{l_{n-3}^{(1)}, l_{n-1}^{(2)} + l(n-1, n-3), l_{n-2}^{(2)} + l(n-2, n-3)\} \\ &= \min\{l_{n-3}^{(1)}, \min_{i>n-3} [l_i^{(2)} + l(i, n-3)]\} \end{aligned} \tag{2.74}$$

This shows that knowing $l_{n-1}^{(2)} = l_{n-1}^{(1)}$, we can calculate $l_{n-2}^{(2)}$, and from $l_{n-2}^{(2)}$ we can calculate $l_{n-3}^{(2)}$. These equations can be generalized to yield

$$l_j^{(2)} = \min\{l_j^{(1)}, \min_{i>j} [l_i^{(2)} + l(i, j)]\}, \quad j = n-2, n-3, \dots, 1 \tag{2.75}$$

In general, for any k the expressions corresponding to (2.71) and (2.75) are given by

$$l_j^{(k)} = \min\{l_j^{(k-1)}, \min_{i<j} [l_i^{(k)} + l(i, j)]\} \tag{2.76}$$

for odd k and $j = 2, 3, \dots, n-1$, where $l_1^{(k)} = l_1^{(k-1)}$; and

$$l_j^{(k)} = \min\{l_j^{(k-1)}, \min_{i>j} [l_i^{(k)} + l(i, j)]\} \tag{2.77}$$

for even k and $j = n-2, n-3, \dots, 1$, where $l_{n-1}^{(k)} = l_{n-1}^{(k-1)}$. Therefore, we can find all the distances from 0 to j using successive approximations with kth-order approximations giving the lengths of the shortest directed paths from 0 to j ($j = 1, 2, \dots, n-1$) using at most k blocks. Since there are at most $n-1$ blocks, the algorithm should terminate if

$$l_j^{(k+1)} = l_j^{(k)} \quad \text{for all } j \tag{2.78}$$

If $l_j^{(k+1)} \neq l_j^{(k)}$ for some j when $k = n-1$, this indicates that the net contains a negative directed circuit.

Observe that from (2.76) and (2.77) we either try those indices i which are less than j or those which are greater than j. Thus, we perform roughly half the work of the Ford–Moore–Bellman algorithm. The computational complexity in this case remains to be $O(n^3)$. However, an additional advantage of the Yen algorithm is that we can erase $l_j^{(k)}$ as soon as $l_j^{(k+1)}$ is generated, thereby reducing the storage requirement of the program.

Example 2.5

We use the Yen algorithm to generate the distances from node 0 to node j ($j = 1, 2, 3, 4, 5$) in the net $G(V, E, l)$ of Fig. 2.15.

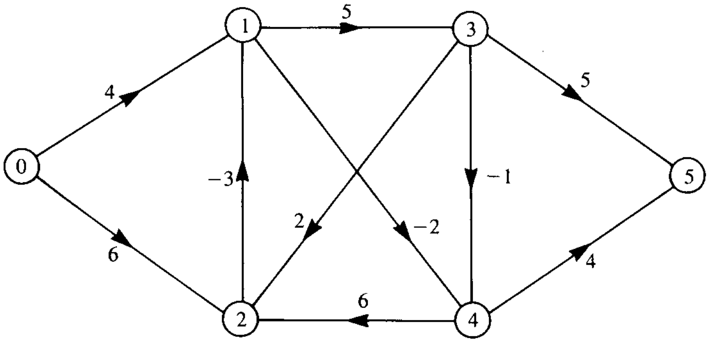

Fig. 2.15. A net $G(V, E, l)$ used to illustrate the Yen algorithm for the calculation of the distances from node 0 to nodes j $(j = 1, 2, 3, 4, 5)$.

Initialization $(k = 0)$

$$l_1^{(0)} = 4, \qquad l_2^{(0)} = 6, \qquad l_i^{(0)} = \infty, \qquad i = 3, 4, 5 \tag{2.79}$$

First iteration $(k = 1)$

$$l_j^{(1)} = \min\,\{l_j^{(0)}, \min_{i<j}\,[l_i^{(1)} + l(i, j)]\}, \qquad j = 2, 3, 4, 5 \tag{2.80}$$

where $l_1^{(1)} = l_1^{(0)} = 4$. We need only consider those $i < j$ such that $(i, j) \in E$, obtaining

$$l_2^{(1)} = \min\,\{l_2^{(0)}\} = 6 \tag{2.81a}$$

$$\begin{aligned} l_3^{(1)} &= \min\,\{l_3^{(0)}, l_1^{(1)} + l(1, 3)\} \\ &= \min\,\{\infty, 4 + 5\} = 9 \end{aligned} \tag{2.81b}$$

$$\begin{aligned} l_4^{(1)} &= \min\,\{l_4^{(0)}, \min\,[l_1^{(1)} + l(1, 4), l_3^{(1)} + l(3, 4)]\} \\ &= \min\,\{\infty, \min\,[4 - 2, 9 - 1]\} = 2 \end{aligned} \tag{2.81c}$$

$$\begin{aligned} l_5^{(1)} &= \min\,\{l_5^{(0)}, \min\,[l_3^{(1)} + l(3, 5), l_4^{(1)} + l(4, 5)]\} \\ &= \min\,\{\infty, \min\,[9 + 5, 2 + 4]\} = 6 \end{aligned} \tag{2.81d}$$

Second iteration $(k = 2)$

$$l_j^{(2)} = \min\,\{l_j^{(1)}, \min_{i>j}\,[l_i^{(2)} + l(i, j)]\}, \qquad j = 4, 3, 2, 1 \tag{2.82}$$

where $l_5^{(2)} = l_5^{(1)} = 6$. We need only consider those $i > j$ such that $(i, j) \in E$, giving

$$l_4^{(2)} = \min\{l_4^{(1)}\} = 2 \tag{2.83a}$$

$$l_3^{(2)} = \min\{l_3^{(1)}\} = 9 \tag{2.83b}$$

$$\begin{aligned} l_2^{(2)} &= \min\{l_2^{(1)}, \min[l_3^{(2)} + l(3,2), l_4^{(2)} + l(4,2)]\} \\ &= \min\{6, \min[9+2, 2+6]\} = 6 \end{aligned} \tag{2.83c}$$

$$l_1^{(2)} = \min\{l_1^{(1)}, l_2^{(2)} + l(2,1)\} = \min\{4, 6-3\} = 3 \tag{2.83d}$$

Third iteration ($k = 3$)

$$l_j^{(3)} = \min\{l_j^{(2)}, \min_{i<j}[l_i^{(3)} + l(i,j)]\}, \qquad j = 2, 3, 4, 5 \tag{2.84}$$

where $l_1^{(3)} = l_1^{(2)} = 3$. It is sufficient to consider only those $i < j$ such that $(i, j) \in E$, yielding

$$l_2^{(3)} = \min\{l_2^{(2)}\} = 6 \tag{2.85a}$$

$$l_3^{(3)} = \min\{l_3^{(2)}, l_1^{(3)} + l(1,3)\} = \min\{9, 3+5\} = 8 \tag{2.85b}$$

$$\begin{aligned} l_4^{(3)} &= \min\{l_4^{(2)}, \min[l_1^{(3)} + l(1,4), l_3^{(3)} + l(3,4)]\} \\ &= \min\{2, \min[3-2, 8-1]\} = 1 \end{aligned} \tag{2.85c}$$

$$\begin{aligned} l_5^{(3)} &= \min\{l_5^{(2)}, \min[l_3^{(3)} + l(3,5), l_4^{(3)} + l(4,5)]\} \\ &= \min\{6, \min[8+5, 1+4]\} = 5 \end{aligned} \tag{2.85d}$$

Fourth iteration ($k = 4$)

$$l_j^{(4)} = \min\{l_j^{(3)}, \min_{i>j}[l_i^{(4)} + l(i,j)]\}, \qquad j = 4, 3, 2, 1 \tag{2.86}$$

where $l_5^{(4)} = l_5^{(3)} = 5$. We need only consider those $i > j$ such that $(i, j) \in E$, obtaining

$$l_4^{(4)} = \min\{l_4^{(3)}\} = 1 \tag{2.87a}$$

$$l_3^{(4)} = \min\{l_3^{(3)}\} = 8 \tag{2.87b}$$

$$\begin{aligned} l_2^{(4)} &= \min\{l_2^{(3)}, \min[l_3^{(4)} + l(3,2), l_4^{(4)} + l(4,2)]\} \\ &= \min\{6, \min[8+2, 1+6]\} = 6 \end{aligned} \tag{2.87c}$$

$$l_1^{(4)} = \min\{l_1^{(3)}, l_2^{(4)} + l(2,1)\} = 3 \tag{2.87d}$$

The algorithm terminates at the end of the fourth iteration, since for

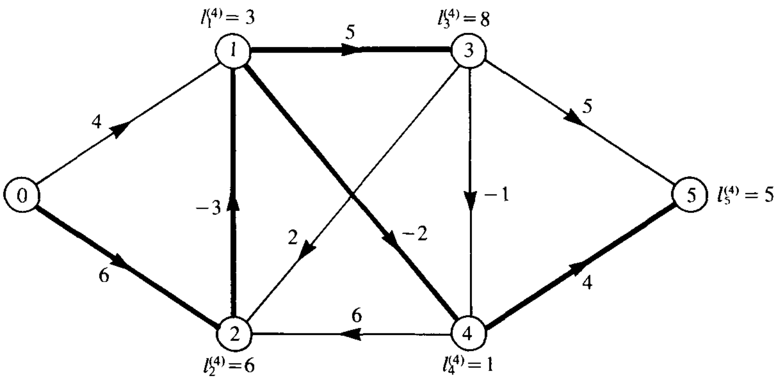

Fig. 2.16. A net used to show the distances from 0 to j ($j = 1, 2, 3, 4, 5$) in the net of Fig. 2.15 with the tree formed by the shortest directed paths being denoted by the heavy lines.

$k = 4$ we have

$$l_j^{(4)} = l_j^{(3)} \quad \text{for} \quad j = 1, 2, 3, 4, 5 \tag{2.88}$$

The distances from 0 to j ($j = 1, 2, 3, 4, 5$) in the net are as indicated in Fig. 2.16 with the tree formed by the shortest directed paths being denoted by the heavy lines.

Example 2.6

Consider the net $G(V, E, l)$ of Fig. 2.17. We use the Yen algorithm to calculate the distances from node 0 to nodes j ($j = 1, 2, \ldots, 13$).

Initialization ($k = 0$)

$$l_1^{(0)} = -1, \quad l_4^{(0)} = 2, \quad l_i^{(0)} = \infty, \qquad i = 2, 3, 5, 6, \ldots, 13 \tag{2.89}$$

The results of first three iterations are shown in Table 2.4. Observe from the table that for $k = 3$ we have

$$l_j^{(3)} = l_j^{(2)}, \qquad j = 1, 2, \ldots, 13 \tag{2.90}$$

Thus, the algorithm terminates at the end of the third iteration or $k = 3$, which is considerably less than $n - 1 = 13$ as required for a general net. The distances together with the tree formed by the shortest directed paths are as indicated in Fig. 2.18 with the heavy lines denoting the tree branches.

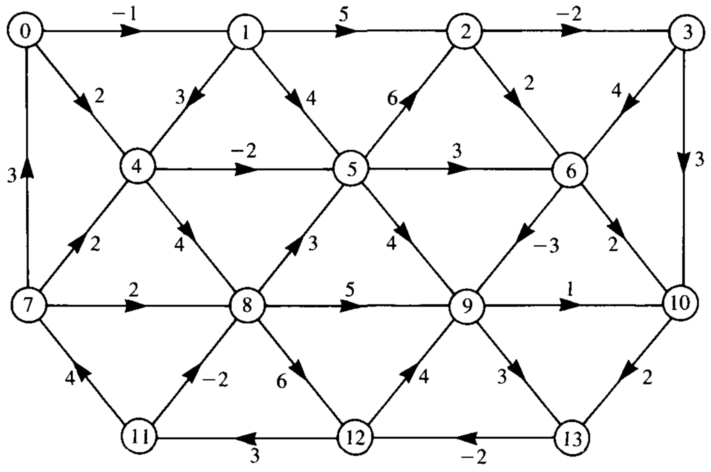

Fig. 2.17. A net $G(V, E, l)$ used to illustrate the Yen algorithm for the calculation of the distances from node 0 to nodes j ($j = 1, 2, \ldots, 13$).

TABLE 2.4

$k \backslash j$	1	2	3	4	5	6	7	8	9	10	11	12	13
	Nodes												
$l_j^{(1)}$	−1	4	2	2	0	3	∞	6	0	1	∞	12	3
$l_j^{(2)}$	−1	4	2	2	0	3	8	2	0	1	4	1	3
$l_j^{(3)}$	−1	4	2	2	0	3	8	2	0	1	4	1	3

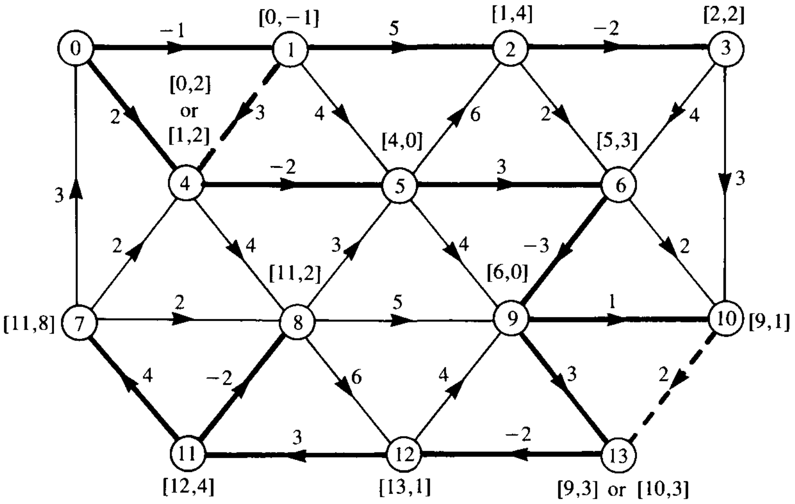

Fig. 2.18. A net used to show the distances from 0 to j ($j = 1, 2, \ldots, 13$) in the net of Fig. 2.17 with the tree formed by the shortest directed paths being denoted by the heavy lines.

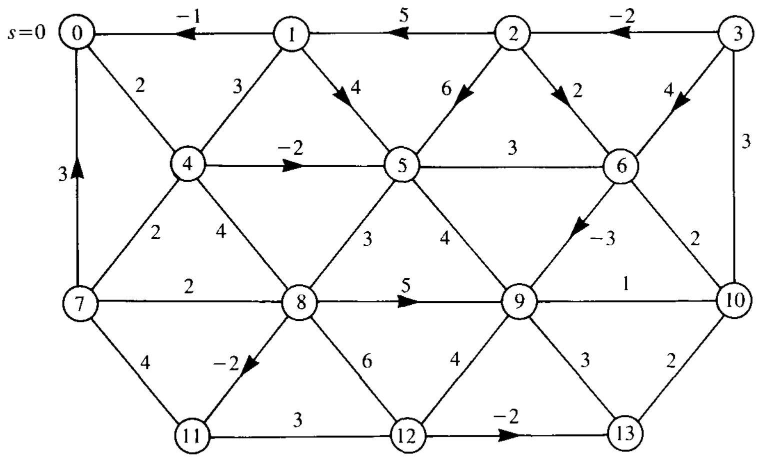

Fig. 2.19. A weighted mixed graph used to illustrate the Yen algorithm for the calculation of the distances from node 0 to nodes j ($j = 1, 2, \ldots, 13$).

Example 2.7

We use the Yen algorithm to calculate the distances from node 0 to nodes j ($j = 1, 2, \ldots, 13$) in the net $G(V, E, l)$ of Fig. 2.19. The details of the procedure are shown in Table 2.5.

The algorithm terminates at the end of the third iteration and the resulting distances and the tree rooted at node 0 are indicated in Fig. 2.20 with the heavy lines denoting the tree branches.

2.2.4 Ford–Fulkerson Algorithm

The present algorithm is applicable to any nets that do not contain any negative directed circuits, but the nets are allowed to have negative or zero arc lengths. The procedure is due to Ford and Fulkerson (1962), and is very

TABLE 2.5

	Nodes												
k \ j	1	2	3	4	5	6	7	8	9	10	11	12	13
$l_j^{(0)}$	∞	∞	∞	2	∞	∞	∞	∞	∞	∞	∞	∞	∞
$l_j^{(1)}$	∞	∞	∞	2	0	3	4	3	0	1	1	4	2
$l_j^{(2)}$	5	2	4	2	0	3	4	3	0	1	1	4	2
$l_j^{(3)}$	5	2	4	2	0	3	4	3	0	1	1	4	2

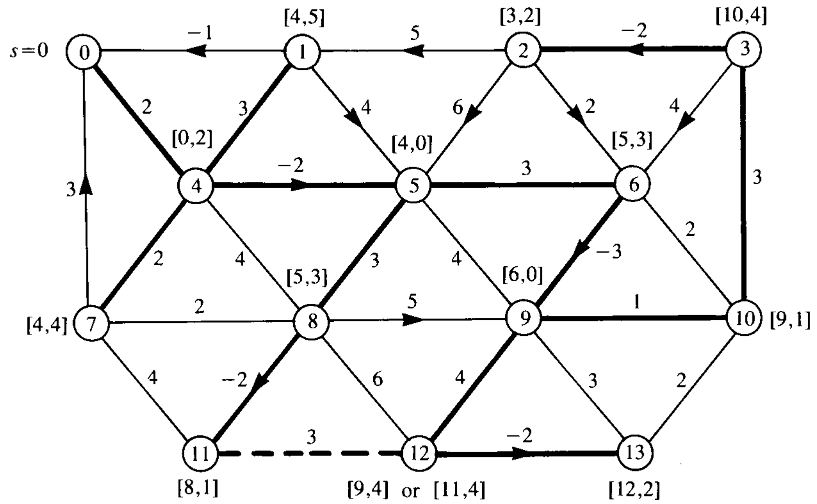

Fig. 2.20. A mixed graph used to show the distances from 0 to j ($j = 1, 2, \dots, 13$) in the net of Fig. 2.19 with the tree rooted at node 0 being denoted by the heavy lines.

similar to the Ford–Moore–Bellman algorithm in that it sequentially reduces the labels on the nodes by examining arcs. The final labels on the nodes will also permit us to trace back from every node the shortest directed path from the origin to that node.

Let $G(V, E, l)$ be an n-node net, in which a node is designated as the origin or source $s = 0$ and the other nodes are denoted by the integers $1, 2, \dots, n-1$. Each node i of G will be assigned a label consisting of an ordered pair of real numbers of the form $[x, l(i)]$. The first number indicates a node designation of G, and the second the conditional distance from s to i. These labels will be updated sequentially until the conditional distances from s become the true distances. The details of the Ford–Fulkerson algorithm are described below:

Step 1. Set $l(s) = 0$ and $l(i) = \infty$ for $i \neq s$. Assign each node i a label $[\cdot, l(i)]$.

Step 2. Find an arc $(i, j) \in E$ such that

$$l(i) + l(i, j) < l(j) \tag{2.91}$$

and change the label on node j to $[i, l(i) + l(i, j)]$, and repeat. If no such arc is found, terminate.

After termination of the algorithm, the second number of the label on a node j shows the distance from s to j in the net, and the first number is the initial node q of the last arc (q, j) to j in a shortest directed path from s to j.

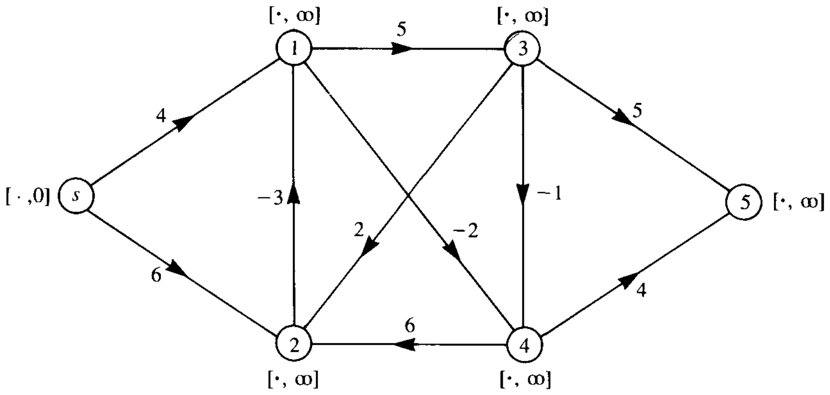

Fig. 2.21. A net $G(V, E, l)$ used to illustrate the Ford–Fulkerson algorithm for the generation of the shortest directed paths.

Thus, to identify the sequence of nodes of a shortest directed path from s to j, we follow the steps below:

Step 1. Set $i = j$.

Step 2. Identify k from the label $[k, l(i)]$ on node i. If i is not labeled, there is no such directed path.

Step 3. Set $i = k$. If $i = s$, terminate. Otherwise, return to Step 2.

Example 2.8

We use the Ford–Fulkerson algorithm to generate the shortest directed paths from node s to node j ($j = 1, 2, 3, 4, 5$) in the net $G(V, E, l)$ of Fig. 2.21, as follows:

Step 1. Assign the label $[\cdot, 0]$ to node s, and the label $[\cdot, \infty]$ to all other nodes j ($j = 1, 2, 3, 4, 5$) as indicated in Fig. 2.21.

Step 2. $l(1) = \infty > l(s) + l(s, 1) = 0 + 4 = 4$. Change the label $[\cdot, \infty]$ on node 1 to $[s, 4]$.
$l(2) = \infty > l(s) + l(s, 2) = 0 + 6 = 6$. Change the label $[\cdot, \infty]$ on node 2 to $[s, 6]$.
$l(3) = \infty > l(1) + l(1, 3) = 4 + 5 = 9$. Change the label $[\cdot, \infty]$ on node 3 to $[1, 9]$.
$l(4) = \infty > l(1) + l(1, 4) = 4 - 2 = 2$. Change the label $[\cdot, \infty]$ on node 4 to $[1, 2]$.
$l(5) = \infty > l(4) + l(4, 5) = 2 + 4 = 6$. Change the label $[\cdot, \infty]$ on node 5 to $[4, 6]$.
$l(1) = 4 > l(2) + l(2, 1) = 6 - 3 = 3$. Change the label $[s, 4]$ on node 1 to $[2, 3]$.

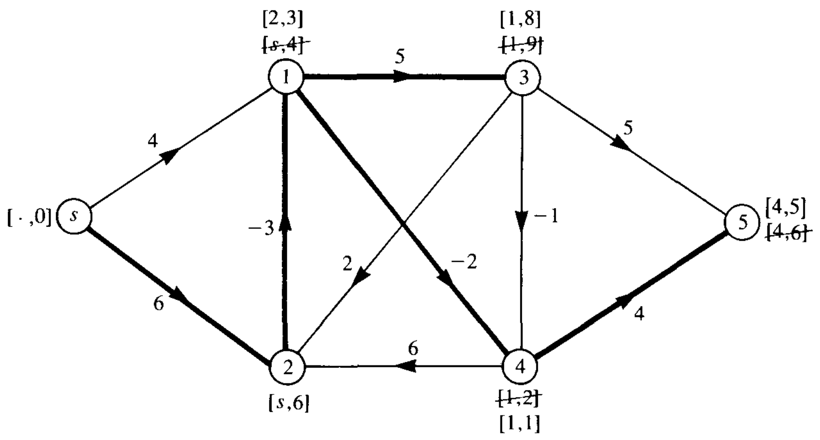

Fig. 2.22. The final labels of the nodes of the net of Fig. 2.21 after the application of the Ford–Fulkerson algorithm.

$l(3) = 9 > l(1) + l(1, 3) = 3 + 5 = 8$. Change the label $[1, 9]$ on node 3 to $[1, 8]$.
$l(4) = 2 > l(1) + l(1, 4) = 3 - 2 = 1$. Change the label $[1, 2]$ on node 4 to $[1, 1]$.
$l(5) = 6 > l(4) + l(4, 5) = 1 + 4 = 5$. Change the label $[4, 6]$ on node 5 to $[4, 5]$.

At the end of the above operations, no more arcs $(i, j) \in E$ can be found such that $l(i) + l(i, j) < l(j)$. The algorithm terminates and the final labels of the nodes are shown in Fig. 2.22. To identify the sequence of nodes of a shortest directed path from s to, say, $j = 5$, we follow the steps below:

Step 1. Set $i = 5$.
Step 2. Identify node 4 from the label $[4, 5]$ on node $i = 5$.
Step 3. Set $i = 4$. Return to Step 2.
Step 2. Identify node 1 from the label $[1, 1]$ on node $i = 4$.
Step 3. Set $i = 1$. Return to Step 2.
Step 2. Identify node 2 from the label $[2, 3]$ on node $i = 1$.
Step 3. Set $i = 2$. Return to Step 2.
Step 2. Identify node s from the label $[s, 6]$ on node $i = 2$.
Step 3. Set $i = s$. Terminate.

Thus, we can trace back the nodes from node 5 to node s as 4, 1, 2, and s. The node sequence from s to 5 is therefore $\{s, 2, 1, 4, 5\}$, and the

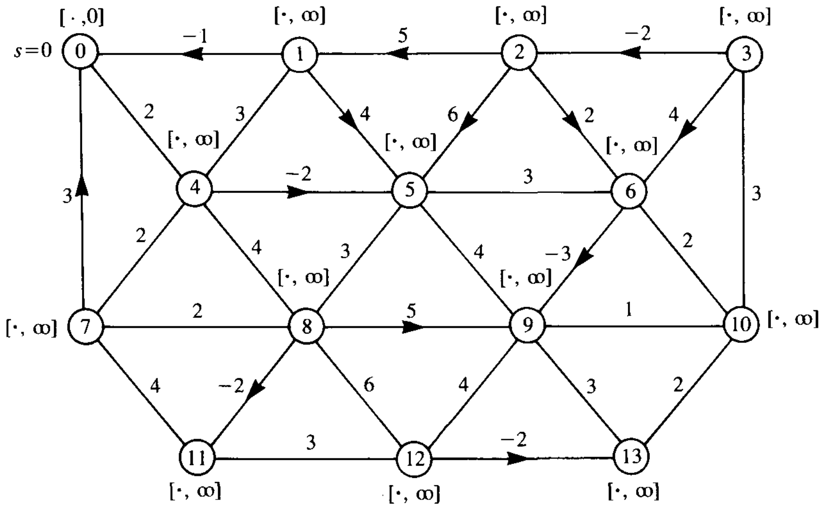

Fig. 2.23. A net $G(V, E, l)$ used to illustrate the Ford–Fulkerson algorithm for the generation of the shortest directed paths.

corresponding shortest directed path P_{s5} is found to be

$$P_{s5} = (s, 2)(2, 1)(1, 4)(4, 5) \tag{2.92}$$

This directed path together with others are shown in Fig. 2.22 with the heavy lines.

Example 2.9

We use the Ford–Fulkerson algorithm to generate the shortest directed paths from node s to nodes j ($j = 1, 2, \ldots, 13$) in the net $G(V, E, l)$ of Fig. 2.23, as follows:

Step 1. Assign the label $[\cdot, 0]$ to node s, and the label $[\cdot, \infty]$ to all other nodes j ($j = 1, 2, \ldots, 13$).

Step 2. $l(4) = \infty > l(s) + l(s, 4) = 0 + 2 = 2$. Label node 4 as $[s, 2]$.
$l(5) = \infty > l(4) + l(4, 5) = 2 - 2 = 0$. Label node 5 as $[4, 0]$.
$l(6) = \infty > l(5) + l(5, 6) = 0 + 3 = 3$. Label node 6 as $[5, 3]$.
$l(7) = \infty > l(4) + l(4, 7) = 2 + 2 = 4$. Label node 7 as $[4, 4]$.
$l(8) = \infty > l(4) + l(4, 8) > l(5) + l(5, 8) = 3$. Label node 8 as $[5, 3]$.
$l(9) = \infty > l(8) + l(8, 9) > l(5) + l(5, 9) > l(6) + l(6, 9) = 0$. Label node 9 as $[6, 0]$.
$l(10) = \infty > l(6) + l(6, 10) > l(9) + l(9, 10) = 1$. Label node 10 as $[9, 1]$.

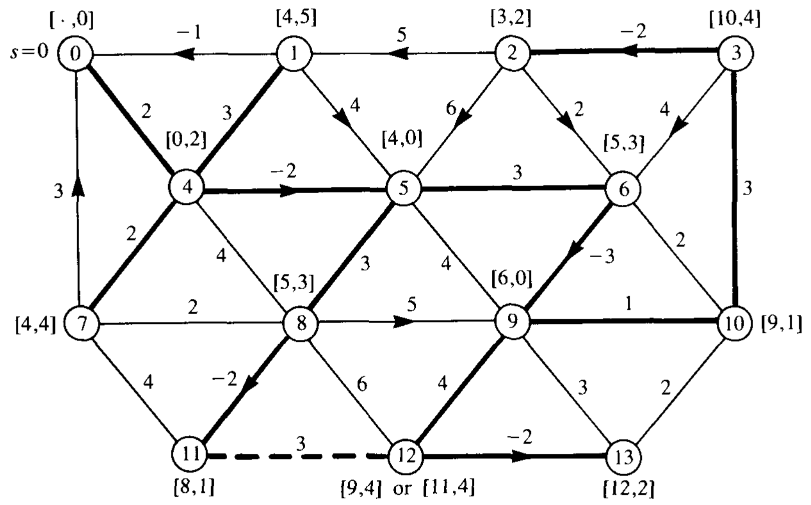

Fig. 2.24. The final labels of the nodes of the net of Fig. 2.23 after the application of the Ford–Fulkerson algorithm.

$l(11) = \infty > l(7) + l(7, 11) > l(8) + l(8, 11) = 1$. Label node 11 as $[8, 1]$.
$l(12) = \infty > l(8) + l(8, 12) > l(9) + l(9, 12) = l(11) + l(11, 12) = 4$. Label node 12 either as $[9, 4]$ or as $[11, 4]$.
$l(1) = \infty > l(4) + l(4, 1) = 5$. Label node 1 as $[4, 5]$.
$l(3) = \infty > l(10) + l(10, 3) = 4$. Label node 3 as $[10, 4]$.
$l(2) = \infty > l(3) + l(3, 2) = 2$. Label node 2 as $[3, 2]$.
$l(13) = \infty > l(9) + l(9, 13) = l(10) + l(10, 13) > l(12) + l(12, 13) = 2$. Label node 13 as $[12, 2]$.

The algorithm terminates and the final labels of the nodes are as shown in Fig. 2.24 with the heavy lines denoting the tree formed by all the shortest directed paths from s to j ($j = 1, 2, \ldots, 13$).

Example 2.10

We use the Ford–Fulkerson algorithm to generate the shortest directed paths from node s to nodes j ($j = 1, 2, \ldots, 13$) in the net $G(V, E, l)$ of Fig. 2.25, as follows:

Step 1. Assign the label $[\cdot, 0]$ to node s, and the label $[\cdot, \infty]$ to all other nodes j ($j = 1, 2, \ldots, 13$).

Step 2. $l(1) = \infty > l(s) + l(s, 1) = -1$. Label node 1 as $[s, -1]$.
$l(2) = \infty > l(1) + l(1, 2) = 4$. Label node 2 as $[1, 4]$.
$l(3) = \infty > l(2) + l(2, 3) = 2$. Label node 3 as $[2, 2]$.

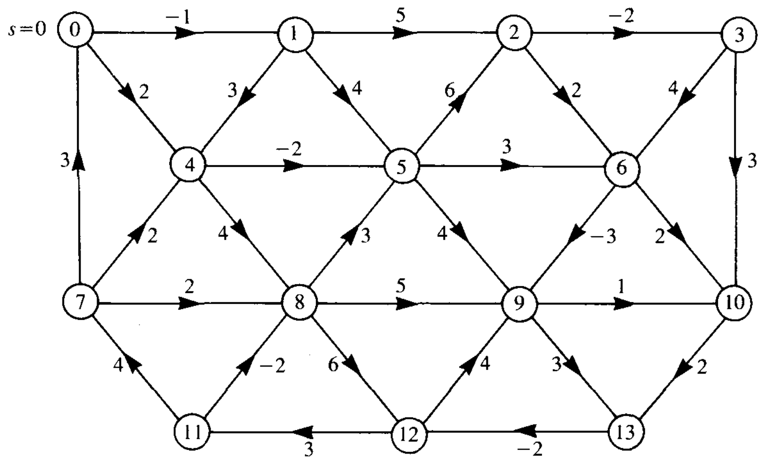

Fig. 2.25. A net $G(V, E, l)$ containing negative arc lengths used to illustrate the Ford–Fulkerson algorithm for the generation of the shortest directed paths.

$l(4) = \infty > l(s) + l(s, 4) = l(1) + l(1, 4) = 2$. Label node 4 as $[s, 2]$ or $[1, 2]$.
$l(5) = \infty > l(4) + l(4, 5) = 0$. Label node 5 as $[4, 0]$.
$l(6) = \infty > l(5) + l(5, 6) = 3$. Label node 6 as $[5, 3]$.
$l(8) = \infty > l(4) + l(4, 8) = 6$. Label node 8 as $[4, 6]$.
$l(9) = \infty > l(6) + l(6, 9) = 0$. Label node 9 as $[6, 0]$.
$l(10) = \infty > l(9) + l(9, 10) = 1$. Label node 10 as $[9, 1\}$.
$l(12) = \infty > l(8) + l(8, 12) = 12$. Label node 12 as $[8, 12]$.
$l(13) = \infty > l(9) + l(9, 13) = l(10) + l(10, 13) = 3$. Label node 13 as $[9, 3]$ or $[10, 3]$.
$l(12) = \infty > l(13) + l(13, 12) = 1$. Label node 12 as $[13, 1]$.
$l(11) = \infty > l(12) + l(12, 11) = 4$. Label node 11 as $[12, 4]$.
$l(8) = 6 > l(11) + l(11, 8) = 2$. Label node 8 as $[11, 2]$.
$l(7) = \infty > l(11) + l(11, 7) = 8$. Label node 7 as $[11, 8]$.

The algorithm terminates and the final labels of the nodes are as shown in Fig. 2.26 with the heavy lines denoting the tree formed by all the shortest directed paths from s to j ($j = 1, 2, \ldots, 13$).

We now proceed to justify the algorithm by showing that under the assumption that the net does not contain any negative directed circuits, the algorithm will terminate. At termination, the node labels give the distances from s to the nodes, and the shortest directed paths can be identified from these labels.

After initialization in Step 1 and at any stage of the iteration in Step 2, if

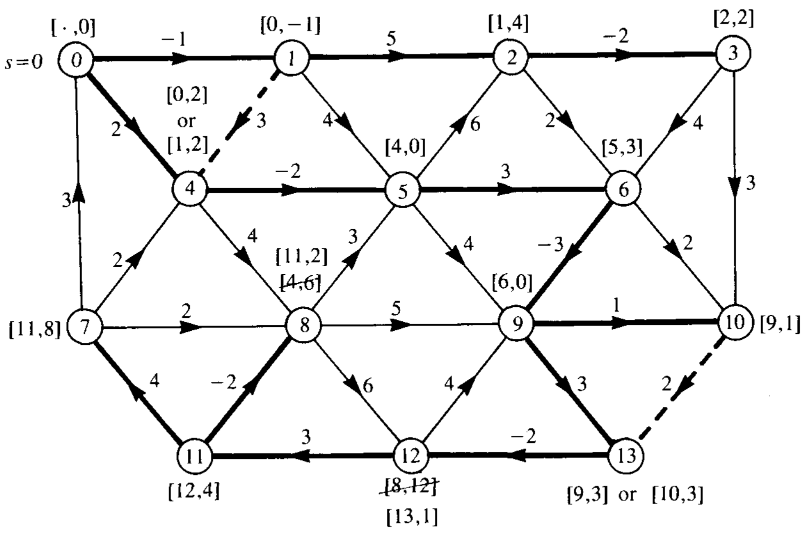

Fig. 2.26. The final labels of the nodes of the net of Fig. 2.25 after the application of the Ford–Fulkerson algorithm.

$l(j) < \infty$ for $j \neq s$, then node j has a label $[i_1, l(j)]$ such that

$$l(i_1) + l(i_1, j) \leqq l(j) \tag{2.93}$$

showing that $l(i_1) < \infty$. Notice that the existence of the label $[i_1, l(j)]$ on j indicates that at some early stage we have $l(i_1) + l(i_1, j) = l(j)$. It is possible that the label $[i_2, l(i_1)]$ on i_1 may be changed again at a later stage to produce an inequality in (2.93). Now the existence of a label $[i_2, l(i_1)]$ on i_1 signifies that

$$l(i_2) + l(i_2, i_1) \leqq l(i_1) \tag{2.94}$$

showing $l(i_2) < \infty$. It follows that if we start at any stage at a node j with $l(j) < \infty$ and follow the labels as outlined above, we generate a sequence of nodes $j, i_1, i_2, \ldots,$ which either terminates at s or continues indefinitely. If it continues indefinitely, the nodes of the sequence will have to be repeated and a directed circuit can be identified.

Suppose that the sequence of nodes

$$j_1, j_2, \ldots, j_k = j_1 \tag{2.95}$$

represents a directed circuit yielded by the labels as outlined above. If j_m was the last node of the directed circuit to receive a label from its predecessor, then at the time immediately before j_m was labeled from j_{m-1}

we had

$$l(j_m) > l(j_{m-1}) + l(j_{m-1}, j_m) \tag{2.96}$$

$$l(j_{m+1}) \geqq l(j_m) + l(j_m, j_{m+1}) \tag{2.97}$$

Observe that immediately after j_m received a label from j_{m-1}, $l(j_m)$ is reduced to $l(j_{m-1}) + l(j_{m-1}, j_m)$ to produce an inequality in (2.97). Thus, if the primes are used to denote the node numbers after j_m received a label from j_{m-1}, we have

$$l'(j_{m+1}) > l'(j_m) + l(j_m, j_{m+1}) \tag{2.98}$$

For other arcs around the directed circuit, we have

$$l'(j_{p+1}) \geqq l'(j_p) + l(j_p, j_{p+1}), \qquad p \neq m, \quad p = 1, 2, \ldots, k-1 \tag{2.99}$$

Summing (2.98) and (2.99) around the directed circuit and noting the strict inequality of (2.98) results in the strict inequality

$$\sum_{i=1}^{k-1} l(j_i, j_{i+1}) < 0 \tag{2.100}$$

contradicting our assumption that the net does not contain any negative directed circuits. Thus, at every stage of the iterations in Step 2, the sequence of nodes generated from the labels of any node j with $l(j) < \infty$ eventually arrives at s and terminates with $l(s) = 0$; for otherwise s would be labeled from some node and a negative directed circuit is identified.

After termination in Step 2, let

$$s = j_1, j_2, \ldots, j_{q-1}, j_q = j \tag{2.101}$$

be a directed path from s to j yielded by the labels as outlined above. Then we have

$$l(j_{p+1}) = l(j_p) + l(j_p, j_{p+1}), \qquad p = 1, 2, \ldots, q-1 \tag{2.102}$$

for otherwise the algorithm would not have been terminated in Step 2. Summing (2.102) along the directed path gives

$$l(j) = l(s) + \sum_{i=1}^{q-1} l(j_i, j_{i+1}) \tag{2.103}$$

Since $l(s) = 0$, (2.103) becomes

$$l(j) = \sum_{i=1}^{q-1} l(j_i, j_{i+1}) \tag{2.104}$$

Suppose that there were a shorter directed path from s to j other than the one shown in (2.101) yielded by the labels, say

$$s = i_1, i_2, \ldots, i_w = j \tag{2.105}$$

Then along this directed path we have as in (2.102)

$$l(i_{x+1}) = l(i_x) + l(i_x, i_{x+1}), \qquad x = 1, 2, \ldots, w-1 \tag{2.106}$$

the sum of which is

$$l(j) = \sum_{x=1}^{w-1} l(i_x, i_{x+1}) \tag{2.107}$$

Combining (2.104) and (2.107) gives

$$\sum_{i=1}^{q-1} l(j_i, j_{i+1}) = \sum_{x=1}^{w-1} l(i_x, i_{x+i}) \tag{2.108}$$

This shows that the two directed paths (2.101) and (2.105) are of the same length, contradicting the assumption that the latter is shorter than the former. Thus, if the net does not contain any negative directed circuits, the algorithm terminates and the resulting node labels give the distances and the shortest directed paths from s to all other nodes of the net that can be reached from s by directed paths.

2.3 MULTITERMINAL SHORTEST DIRECTED PATHS

In the foregoing, we discussed procedures for finding all shortest directed paths from a preassigned source node s to all other nodes. In the present section, we shall look for shortest directed paths between all pairs of nodes in a net. One obvious way for obtaining the solution is to apply the single source algorithms of Section 2.2 n times, each time with a different node chosen as the source node s. Since for each choice of s the computational complexity is $O(n^3)$ for a general n-node net, the resulting procedure would require $O(n^4)$. In the following, we describe several other more efficient algorithms for solving this problem. The methods are applicable to general nets without negative directed circuits.

2.3.1 Matrix Algorithm

Consider an n-node net $G(V, E, l)$ in which the nodes have been designated as $1, 2, \ldots, n$. For our purposes, we introduce two matrices based on G.

DEFINITION 2.2

Connection Matrix. The *connection matrix* $\boldsymbol{C} = [c_{ij}]$ of an n-node net $G(V, E, l)$ is an $n \times n$ matrix whose ith row and jth column element c_{ij} is the length of the arc (i, j) or $c_{ij} = l(i, j)$, where $c_{ii} = 0$ for all i and $c_{ij} = \infty$ for all $(i, j) \notin E$.

DEFINITION 2.3

Distance Matrix. The *distance matrix* $\boldsymbol{D} = [d_{ij}]$ of an n-node net $G(V, E, l)$ is an $n \times n$ matrix whose ith row and jth column element d_{ij} is the distance from node i to node j in G, where $d_{ii} = 0$ for all i.

We first define a special matrix binary operation, denoted by the symbol $\otimes$, called the *minaddition*. Given two square matrices of order n

$$\boldsymbol{A} = [a_{ij}] \tag{2.109}$$

$$\boldsymbol{B} = [b_{ij}] \tag{2.110}$$

the minaddition of $\boldsymbol{A}$ and $\boldsymbol{B}$ is a matrix

$$\boldsymbol{W} = \boldsymbol{A} \otimes \boldsymbol{B} = [w_{ij}] \tag{2.111}$$

of the same order whose ith row and jth column element w_{ij} is determined by the equation

$$w_{ij} = \min_k (a_{ik} + b_{kj}) \tag{2.112}$$

Recall that in the ordinary matrix product, the element w_{ij} is defined by

$$w_{ij} = \sum_k a_{ik} b_{kj} \tag{2.113}$$

If in (2.113) the summation is replaced by the minimum operation, and the product $a_{ik}b_{kj}$ is changed to addition, we obtain the minaddition operation (2.112). It is straightforward to verify that minaddition is associative, but is not commutative. In what follows, we shall use power, product and similar terms in the sense of minaddition. Thus, we write

$$\boldsymbol{C}^2 = \boldsymbol{C} \otimes \boldsymbol{C} \tag{2.114}$$

and $\boldsymbol{C}^3$ is the third power of $\boldsymbol{C}$ in the sense of minaddition. Also, we shall use the symbol $c_{ij}^{(k)}$ to denote the ith row and jth column element of $\boldsymbol{C}^k$:

$$\boldsymbol{C}^k = [c_{ij}^{(k)}] \tag{2.115}$$

It is not difficult to see that the entry $c_{ij}^{(2)}$ gives the length of a shortest directed path from i to j in G using at most two arcs, because in this case

$$c_{ij}^{(2)} = \min_k (c_{ik} + c_{kj}) \tag{2.116}$$

and we have tried every node k as a possible intermediate node of a two-arc or one-arc directed path from i to j, where the one-arc path corresponds to $k = i$ or $k = j$. The entries $c_{ij}^{(3)}$ of $\boldsymbol{C}^3 = \boldsymbol{C} \otimes \boldsymbol{C} \otimes \boldsymbol{C}$ represent the lengths of the shortest directed paths from i to j using at most three arcs. By induction, we can show that the entries $c_{ij}^{(k)}$ of $\boldsymbol{C}^k$ gives the lengths of the shortest directed paths from i to j using at most k arcs. Since we have already calculated $\boldsymbol{C}^2$, there is no need to calculate $\boldsymbol{C}^3$ first in order to calculate $\boldsymbol{C}^4$. We can calculate $\boldsymbol{C}^4$ directly from $\boldsymbol{C}^2$ by virtue of the fact that the minaddition is associative:

$$\boldsymbol{C}^4 = \boldsymbol{C} \otimes \boldsymbol{C} \otimes \boldsymbol{C} \otimes \boldsymbol{C} = (\boldsymbol{C} \otimes \boldsymbol{C}) \otimes (\boldsymbol{C} \otimes \boldsymbol{C}) = \boldsymbol{C}^2 \otimes \boldsymbol{C}^2 \tag{2.117}$$

Once we obtain $\boldsymbol{C}^4$, we can use it to calculate $\boldsymbol{C}^8$. This squaring process can be continued until $\boldsymbol{C}^k$, $k = 2^\alpha$, is obtained, where α is the least integer greater than or equal to $\log_2 (n-1)$. This follows from the fact that the number of arcs in any path in an n-node net is at most $n-1$. Therefore, we have

$$\boldsymbol{D} = \boldsymbol{C}^k = \boldsymbol{C}^{n-1} \quad \text{for all } k \geqq n-1 \tag{2.118}$$

Example 2.11

Consider the net $G(V, E, l)$ of Fig. 2.21, which is redrawn in Fig. 2.27.

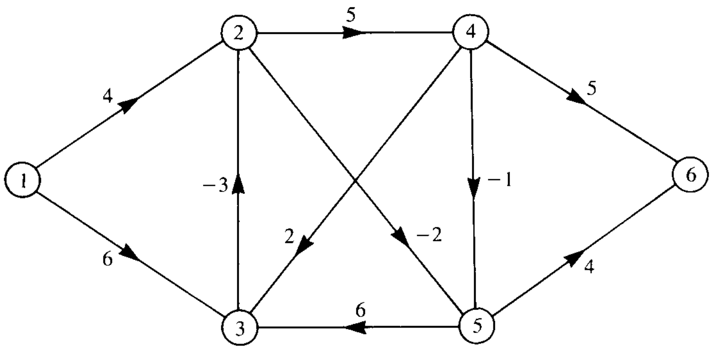

Fig. 2.27. A net $G(V, E, l)$ used to illustrate the construction of the connection matrix and the matrix binary operation called the minaddition.

The connection matrix of the net is found to be

$$\boldsymbol{C} = \begin{array}{c|cccccc} & 1 & 2 & 3 & 4 & 5 & 6 \\ \hline 1 & 0 & 4 & 6 & \infty & \infty & \infty \\ 2 & \infty & 0 & \infty & 5 & -2 & \infty \\ 3 & \infty & -3 & 0 & \infty & \infty & \infty \\ 4 & \infty & \infty & 2 & 0 & -1 & 5 \\ 5 & \infty & \infty & 6 & \infty & 0 & 4 \\ 6 & \infty & \infty & \infty & \infty & \infty & 0 \end{array} \tag{2.119}$$

the square of which in the sense of minaddition is given by

$$\boldsymbol{C}^2 = \boldsymbol{C} \otimes \boldsymbol{C} = \begin{array}{c|cccccc} & 1 & 2 & 3 & 4 & 5 & 6 \\ \hline 1 & 0 & 3 & 6 & 9 & 2 & \infty \\ 2 & \infty & 0 & 4 & 5 & -2 & 2 \\ 3 & \infty & -3 & 0 & 2 & -5 & \infty \\ 4 & \infty & -1 & 2 & 0 & -1 & 3 \\ 5 & \infty & 3 & 6 & \infty & 0 & 4 \\ 6 & \infty & \infty & \infty & \infty & \infty & 0 \end{array} \tag{2.120}$$

This matrix can again be squared to yield

$$\boldsymbol{C}^4 = \boldsymbol{C}^2 \otimes \boldsymbol{C}^2 = \begin{array}{c|cccccc} & 1 & 2 & 3 & 4 & 5 & 6 \\ \hline 1 & 0 & 3 & 6 & 8 & 1 & 5 \\ 2 & \infty & 0 & 4 & 5 & -2 & 2 \\ 3 & \infty & -3 & 0 & 2 & -5 & -1 \\ 4 & \infty & -1 & 2 & 0 & -3 & 1 \\ 5 & \infty & 3 & 6 & 8 & 0 & 4 \\ 6 & \infty & \infty & \infty & \infty & \infty & 0 \end{array} \tag{2.121}$$

It is straightforward to verify that

$$\boldsymbol{D} = \boldsymbol{C}^k = \boldsymbol{C}^4 \quad \text{for all } k \geqq 4 \tag{2.122}$$

We now count the number of additions and comparisons in (2.118). For each entry $c_{ij}^{(2)}$ of $\boldsymbol{C}^2$, (2.116) requires n additions and $n-1$ comparisons. For $\boldsymbol{C}^2$ we require a total of n^3 additions and $n^2(n-1)$ comparisons. Therefore, the computational complexity of the algorithm is $O(\alpha n^3)$.

In calculating $\boldsymbol{C}^k$, if instead of using the minaddition operation we use the ordinary matrix multiplication and addition, the ith row and jth column element $c_{ij}^{(k)}$ of C^k can be written explicitly as

$$c_{ij}^{(k)} = \sum_{j_1=1}^{n} \sum_{j_2=1}^{n} \cdots \sum_{j_{k-1}=1}^{n} c_{ij_1} c_{j_1 j_2} \cdots c_{j_{k-1} j} \tag{2.123}$$

where $c_{ij} = c_{ij}^{(1)}$ for all i and j. This shows that each product

$$c_{ij_1} c_{j_1 j_2} \cdots c_{j_{k-1} j} \tag{2.124}$$

in the sum represents a directed edge sequence

$$(i, j_1)(j_1, j_2) \cdots (j_{k-1}, j) \tag{2.125}$$

from i to j using exactly k arcs in the n-node complete directed graph with $c_{ij} = (i, j)$. Equation (2.123) gives all the directed edge sequences from i to j using exactly k arcs. It follows that if $\boldsymbol{C}$ is the connection matrix of $G(V, E, l)$ and if the summations in (2.123) are replaced by the minimum operations and the products are changed to additions, $c_{ij}^{(k)}$ is the length of a shortest directed path from i to j in G using at most k arcs. The reason for this is that using the minaddition operation and setting $c_{xx} = 0$ for all x, for any term in (2.123) corresponding to a directed edge sequence containing a directed circuit, there is a term corresponding to a directed path not longer than the directed edge sequence, because the net does not contain any negative directed circuits.

To reduce the number of operations in (2.118), Narahari Pandit (1961) proposed a revised matrix algorithm. The idea is that an entry of the 'product' matrix in the sense of minaddition, once calculated, immediately replaces the corresponding entries in the two matrices that form the product before the next entry of the product matrix is calculated. Clearly, in such a process, the order of the computation is important in that the entry calculated last uses the entries calculated earlier in the minaddition operations. We call the minaddition procedure the *forward process* if the calculations of the matrix product start from top row and from left to right in forming the minadditions. Let

$$\boldsymbol{C}^{(\mathrm{f})} = [c_{ij}^{(\mathrm{f})}] \tag{2.126}$$

be the matrix calculated by this process, written as

$$\boldsymbol{C}^{(\mathrm{f})} = \boldsymbol{C} \otimes_{\mathrm{f}} \boldsymbol{C} \tag{2.127}$$

where f signifies the forward process, and

$$\boldsymbol{C} = \boldsymbol{C}^{(1)} = [c_{ij}^{(1)}] = [c_{ij}] \tag{2.128}$$

The elements $c_{ij}^{(\mathrm{f})}$ of $\boldsymbol{C}^{(\mathrm{f})}$ can be expressed explicitly as

$$c_{ij}^{(\mathrm{f})} = \min_k \, [c_{ik}^{(p)} + c_{kj}^{(q)}] \tag{2.129}$$

where

$$p = 1, \qquad j \leqq k \tag{2.130a}$$

$$p = \mathrm{f}, \qquad j > k \tag{2.130b}$$

$$q = 1, \qquad i \leqq k \tag{2.130c}$$

$$q = \mathrm{f}, \qquad i > k \tag{2.130d}$$

After obtaining $\boldsymbol{C}^{(\mathrm{f})}$, the calculations of the matrix 'product' of $\boldsymbol{C}^{(\mathrm{f})}$ by itself start from bottom row and from right to left, reversing the order of the first matrix multiplication of (2.127) using the operation $\otimes_{\mathrm{f}}$. This reverse procedure is referred to as the *backward process*:

$$\boldsymbol{C}^{(\mathrm{b})} = [c_{ij}^{(\mathrm{b})}] = \boldsymbol{C}^{(\mathrm{f})} \otimes_{\mathrm{b}} \boldsymbol{C}^{(\mathrm{f})} \tag{2.131}$$

The elements $c_{ij}^{(\mathrm{b})}$ of $\boldsymbol{C}^{(\mathrm{b})}$ can be written explicitly as

$$c_{ij}^{(\mathrm{b})} = \min_k \, [c_{ik}^{(p)} + c_{kj}^{(q)}] \tag{2.132}$$

where

$$p = \mathrm{f}, \qquad k \leqq j \tag{2.133a}$$

$$p = \mathrm{b}, \qquad k > j \tag{2.133b}$$

$$q = \mathrm{f}, \qquad k \leqq i \tag{2.133c}$$

$$q = \mathrm{b}, \qquad k > i \tag{2.133d}$$

In the case where all the arcs of a net $G(V, E, l)$ are nonnegative, Farbey, Land and Murchland (1967) showed that one forward process followed by one backward process is sufficient to generate all the distances between all pairs of nodes. On the other hand, Hu (1967) proved that while two forward processes are not enough, three are always sufficient. More recently, Yang and Chen (1989) demonstrated that one forward process followed by one backward process, three forward processes, one backward process followed by one forward process, or three backward processes are sufficient to generate all the distances between all pairs of nodes in a net with negative arcs but with no negative directed circuits.

We illustrate the above results by the following example.

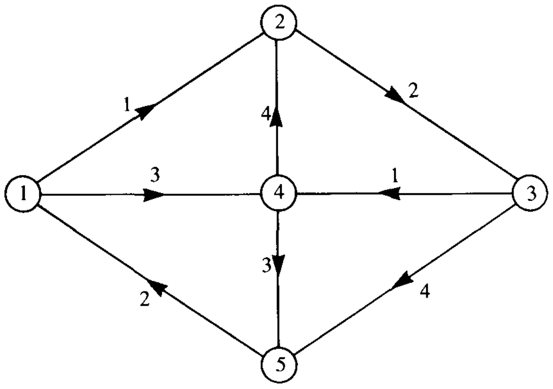

Fig. 2.28. A net $G(V, E, l)$ used to illustrate the forward and backward processes in the matrix calculation of the distances between all pairs of nodes.

Example 2.12

Consider the net $G(V, E, l)$ of Fig. 2.28, the connection matrix of which is found to be

$$\begin{array}{cc} & \begin{array}{ccccc} 1 & 2 & 3 & 4 & 5 \end{array} \\ \begin{array}{c} 1 \\ 2 \\ \boldsymbol{C}=3 \\ 4 \\ 5 \end{array} & \begin{bmatrix} 0 & 1 & \infty & 3 & \infty \\ \infty & 0 & 2 & \infty & \infty \\ \infty & \infty & 0 & 1 & 4 \\ \infty & 4 & \infty & 0 & 3 \\ 2 & \infty & \infty & \infty & 0 \end{bmatrix} \end{array} \qquad (2.134)$$

In the forward process, we calculate

$$c_{12}^{(f)} = \min\{0+1, 1+0, \infty+\infty, 3+4, \infty+\infty\} = 1 \qquad (2.135a)$$

$$c_{13}^{(f)} = \min\{0+\infty, 1+2, \infty+0, 3+\infty, \infty+\infty\} = 3 \qquad (2.135b)$$

At this point, the original entry $c_{13} = \infty$ in $\boldsymbol{C}$ is replaced by $c_{13}^{(f)} = 3$ before continuing the calculation. Note that in machine computation, the replacement operation is automatic, and there is no need to check to see if $c_{ij}^{(f)} = c_{ij}$. To continue, we calculate

$$c_{14}^{(f)} = \min\{0+3, 1+\infty, 3+1, 3+0, \infty+\infty\} = 3 \qquad (2.136a)$$

$$c_{15}^{(f)} = \min\{0+\infty, 1+\infty, 3+4, 3+3, \infty+0\} = 6 \qquad (2.136b)$$

At this stage, we replace the entry $c_{15} = \infty$ by $c_{15}^{(f)} = 6$ before proceeding to the computation of the next element. The final result at the end of the forward process is given by

$$\boldsymbol{C}^{(\mathrm{f})} = \boldsymbol{C} \otimes_{\mathrm{f}} \boldsymbol{C} = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array} \begin{array}{c} \begin{array}{ccccc} 1 & 2 & 3 & 4 & 5 \end{array} \\ \begin{bmatrix} 0 & 1 & 3 & 3 & 6 \\ \infty & 0 & 2 & 3 & 6 \\ 6 & 5 & 0 & 1 & 4 \\ 5 & 4 & 6 & 0 & 3 \\ 2 & 3 & 5 & 5 & 0 \end{bmatrix} \end{array} \tag{2.137}$$

This matrix is used to form $\boldsymbol{C}^{(\mathrm{b})}$ by the backward process, as follows:

$$c_{54}^{(\mathrm{b})} = \min\{2+3, 3+3, 5+1, 5+0, 0+5\} = 5 \tag{2.138a}$$

$$c_{53}^{(\mathrm{b})} = \min\{2+3, 3+2, 5+0, 5+6, 0+5\} = 5 \tag{2.138b}$$

$$\vdots$$

$$c_{23}^{(\mathrm{b})} = \min\{\infty+3, 0+2, 2+0, 3+6, 6+5\} = 2 \tag{2.138c}$$

$$c_{21}^{(\mathrm{b})} = \min\{\infty+0, 0+\infty, 2+6, 3+5, 6+2\} = 8 \tag{2.138d}$$

At this point, we replace the entry $c_{21}^{(\mathrm{f})} = \infty$ by $c_{21}^{(\mathrm{b})} = 8$ before proceeding to calculate $c_{15}^{(\mathrm{b})}$. At the end of the backward process, the matrix becomes

$$\boldsymbol{C}^{(\mathrm{b})} = \boldsymbol{C}^{(\mathrm{f})} \otimes_{\mathrm{b}} \boldsymbol{C}^{(\mathrm{f})} = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array} \begin{array}{c} \begin{array}{ccccc} 1 & 2 & 3 & 4 & 5 \end{array} \\ \begin{bmatrix} 0 & 1 & 3 & 3 & 6 \\ 8 & 0 & 2 & 3 & 6 \\ 6 & 5 & 0 & 1 & 4 \\ 5 & 4 & 6 & 0 & 3 \\ 2 & 3 & 5 & 5 & 0 \end{bmatrix} \end{array} = \boldsymbol{D} \tag{2.139}$$

the distance matrix, the elements of which are the distances between all pairs of nodes in the net $G(V, E, l)$ of Fig. 2.28.

2.3.2 Floyd–Warshall Algorithm

Recall that a subpath of a shortest directed path in a net $G(V, E, l)$ is itself a shortest directed path. In particular, each arc of a shortest directed path is itself a shortest directed path. An arc (i, j) is termed a *basic arc* if it is the shortest directed path from i to j. Thus, arcs belonging to shortest directed paths are all basic arcs. If there is a pair of nodes i and j not connected directly by a basic arc (i, j), we can replace such a nonbasic arc by a basic

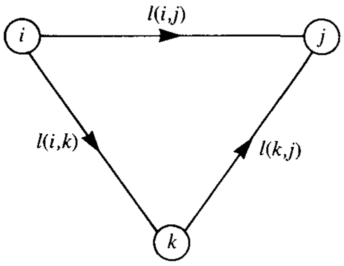

Fig. 2.29. Symbolic representation of the triple operation.

arc whose length is equal to the shortest distance from i to j in G, assuming that arcs not in G are arcs of infinite length in G. The present algorithm describes a simple procedure of creating basic arcs between each pair of nodes not connected by a basic arc, and was first proposed by Floyd (1962) and Warshall (1962).

In a given net $G(V, E, l)$, perform the following operation for a fixed k and all $i, j \neq k$:

$$l(i, j) \leftarrow \min [l(i, j), l(i, k) + l(k, j)] \tag{2.140}$$

As depicted in Fig. 2.29, the operation compares the length of an arc (i, j) with that of a two-arc directed path $(i, k)(k, j)$, the smaller of which is used to replace the length of arc (i, j), and is called the *triple operation*. We begin by setting $k = 1$ and apply the triple operation (2.140) for all $i, j = 2, 3, \ldots, n$. Then we set $k = 2$, and perform the triple operation for all $i, j = 1, 3, \ldots, n$. Notice that in performing the triple operation for $k = 2$, we use all the new arcs created when $k = 1$. We continue in this way until the triple operation is performed for $k = n$. We claim that at the end of the triple operation for $k = n$, the resulting net G' consists only of basic arcs. In other words, the arc length $l(i, j)$ in G' represents the shortest distance from i to j in $G(V, E, l)$. We now proceed to justify this assertion.

Consider an arbitrary shortest directed path, say, $(2, 5)(5, 3)(3, 4)(4, 6)$ as shown in Fig. 2.30(a). We show that the triple operation will create an arc $(2, 6)$, the length of which is equal to the sum of all the basic arcs in $(2, 5)(5, 3)(3, 4)(4, 6)$. Once we verify the assertion for this particular directed path, the procedure can be generalized to any arbitrary shortest directed path, from which a formal proof can be stated. However, we do not find it necessary; only the verification of this arbitrarily chosen path is provided here.

When $k = 2$, the triple operation does not create any new arcs for the directed path of Fig. 2.30(a). When $k = 3$, the triple operation creates a new arc $(5, 4)$ with

$$l(5, 4) = l(5, 3) + l(3, 4) \tag{2.141}$$

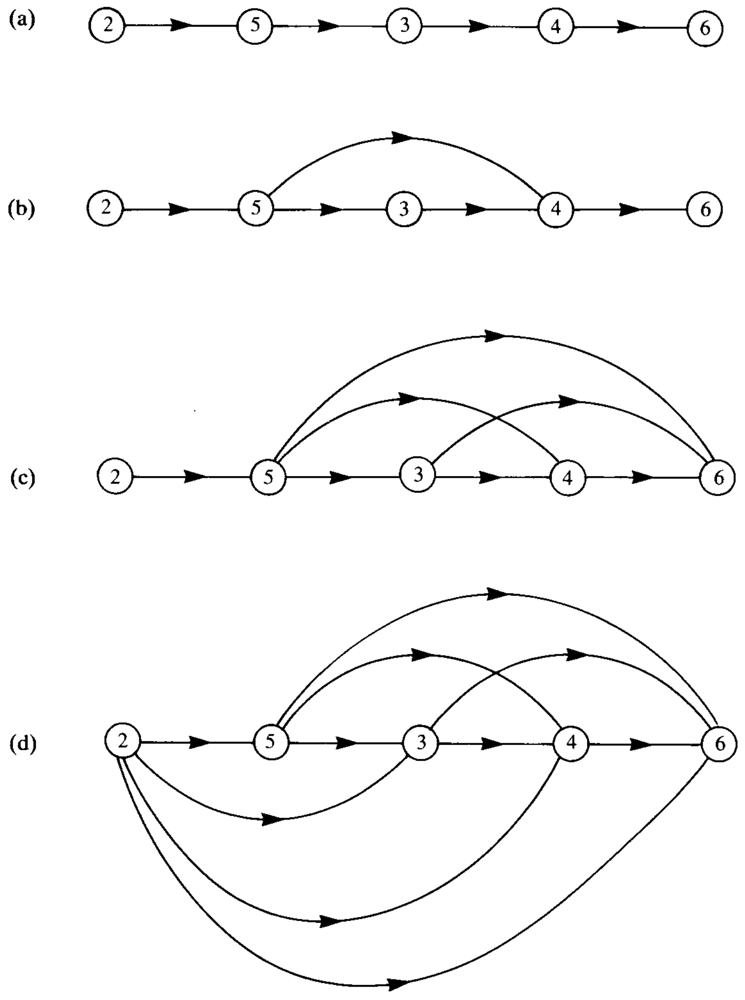

Fig. 2.30. A graph used to illustrate the triple operation: (a) the original graph and $k = 2$, (b) $k = 3$, (c) $k = 4$, (d) $k = 5$.

as shown in Fig. 2.30(b). When $k = 4$, two more arcs (3, 6) and (5, 6) as shown in Fig. 2.30(c) are created with

$$l(3, 6) = l(3, 4) + l(4, 6) \tag{2.142a}$$

$$l(5, 6) = l(5, 4) + l(4, 6) = l(5, 3) + l(3, 4) + l(4, 6) \tag{2.142b}$$

When $k = 5$, three more arcs (2, 3), (2, 4) and (2, 6) are created as indicated in Fig. 2.30(d) with

$$l(2,3) = l(2,5) + l(5,3) \tag{2.143a}$$

$$l(2,4) = l(2,5) + l(5,4) = l(2,5) + l(5,3) + l(3,4) \tag{2.143b}$$

$$l(2,6) = l(2,5) + l(5,6) = l(2,5) + l(5,3) + l(3,4) + l(4,6) \tag{2.143c}$$

confirming that (2, 6) is a basic arc.

The Floyd–Warshall algorithm is applicable to any net $G(V, E, l)$ that does not contain any negative directed circuits. For our purposes, we assume that $l(i, i) = 0$ $(i = 1, 2, \ldots, n)$ and $l(i, j) = \infty$ for $(i, j) \notin E$.

Step 1. Set $k = 0$.
Step 2. $k = k + 1$.
Step 3. For all $i \neq k$, $l(i, k) \neq \infty$ and all $j \neq k$, $l(k, j) \neq \infty$, perform the triple operation

$$l(i,j) \leftarrow \min\,[l(i,j),\ l(i,k) + l(k,j)] \tag{2.144}$$

Step 4. If $l(i, i) \geqq 0$ $(i = 1, 2, \ldots, n)$ and $k < n$, return to Step 2.
If $l(i, i) < 0$ for some i, there is no solution. Stop.
If $l(i, i) \geqq 0$ $(i = 1, 2, \ldots, n)$ and $k = n$, stop.

We remark that in Step 4 if $l(i, i) < 0$ for some i, then a negative directed circuit containing the node i exists in G, and no meaningful solution can be found. If $l(i, i) \geqq 0$ for all i and $k = n$, the solution is found and the arc length $l(i, j)$ in the resulting net G' gives the distance from i to j in G for all i and j. Once the distances between all pairs of nodes are calculated, the shortest directed paths themselves can be determined by using a recursive relation similar to that of (2.12). Alternatively, we can use a bookkeeping mechanism to record information about the directed paths themselves, as follows.

When we perform the triple operation, we keep track of the internal nodes of the directed paths. We use an $n \times n$ matrix

$$\boldsymbol{P} = [p_{ij}] \tag{2.145}$$

for storage and updating purposes. The ith row and jth column entry p_{ij} gives the first internal node on the directed path from i to j. If $p_{ij} = x$, then the first arc of the shortest directed path from i to j is (i, x). Likewise, if $p_{xj} = y$, then the first arc of the shortest directed path from x to j is (x, y). In this way, we can trace back to any of the other nodes. Initially, we assume that in going from i to j the first arc is (i, j). Thus, we set $p_{ij} = j$ for all i and j. When we perform the triple operation, we also update the information in

$\boldsymbol{P}$ with its entries being changed in accordance with the following rule:

$$p_{ij} = p_{ik}, \quad \text{if} \quad l(i, j) > l(i, k) + l(k, j) \tag{2.146a}$$

$$= p_{ij}, \quad \text{if} \quad l(i, j) \leqq l(i, k) + l(k, j) \tag{2.146b}$$

We recognize that the triple operation performed on the net G can easily be performed on the connection matrix $\boldsymbol{C} = [c_{ij}]$ of G. When $k = 1$, we compare every entry $c_{ij}\ (i \neq 1, j \neq 1)$ with $c_{i1} + c_{1j}$. If c_{ij} is greater than $c_{i1} + c_{1j}$, we replace the entry c_{ij} by $c_{i1} + c_{1j}$. Otherwise, the entry remains unaltered. For $k = 2$, we compare every entry $c_{ij}\ (i \neq 2, j \neq 2)$ with $c_{i2} + c_{2j}$. If c_{ij} is greater than $c_{i2} + c_{2j}$, we replace c_{ij} by $c_{i2} + c_{2j}$. Otherwise, the entry remains the same. Notice that in computing for $k = 2$, we have already used the results for $k = 1$. We continue in this way until $k = n$ is computed. Observe that each entry is compared with the sum of two other entries, one in the same row and one in the same column. For each entry, we have to check $(n - 1)^2$ entries. Since there are n nodes to consider, the computational complexity of the Floyd–Warshall algorithm is $O(n^3)$.

Example 2.13

We use the Floyd–Warshall algorithm to find the distances between all pairs of nodes in the net $G(V, E, l)$ of Fig. 2.31. Initially, we set $l(i, i) = 0$ for all i and $l(i, j) = \infty$ for $(i, j) \notin E$.

Step 1. Set $k = 0$.

Step 2. Set $k = k + 1$ or $k = 1$.

Step 3.

$$l(4, 2) \leftarrow \min[\infty, 2 + 1] = 3 \tag{2.147a}$$

$$l(4, 3) \leftarrow \min[-1, 2 + 4] = -1 \tag{2.147b}$$

$$l(4, 5) \leftarrow \min[-1, 2 + 4] = -1 \tag{2.147c}$$

At the end of this step, an arc $(4, 2)$ is created with $l(4, 2) = 3$.

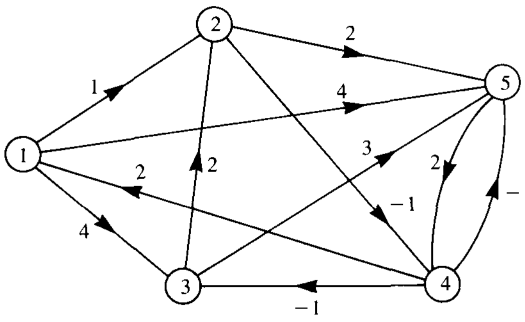

Fig. 2.31. A net $G(V, E, l)$ used to illustrate the Floyd–Warshall algorithm for the calculation of the distances between all pairs of nodes.

Step 4. Return to Step 2.

Step 2. $k = 2$.

Step 3. From here on we list only those arcs whose lengths will be changed after performing the triple operation.

$$l(1, 4) \leftarrow \min[\infty, 1 - 1] = 0 \tag{2.148a}$$

$$l(3, 4) \leftarrow \min[\infty, 2 - 1] = 1 \tag{2.148b}$$

$$l(1, 5) \leftarrow \min[4, 1 + 2] = 3 \tag{2.148c}$$

At the end of this step, two new arcs $(1, 4)$ and $(3, 4)$ are created with $l(1, 4) = 0$ and $l(3, 4) = 1$, and the arc length of $(1, 5)$ is changed from 4 to 3.

Step 4. Return to Step 2.

Step 2. $k = 3$

Step 3.

$$l(4, 2) \leftarrow \min[3, -1 + 2] = 1 \tag{2.149}$$

At the end of this step, the arc length $l(4, 2)$ is changed from 3 to 1.

Step 4. Return to Step 2.

Step 2. $k = 4$

Step 3.

$$l(2, 1) \leftarrow \min[\infty, -1 + 2] = 1 \tag{2.150a}$$

$$l(3, 1) \leftarrow \min[\infty, 1 + 2] = 3 \tag{2.150b}$$

$$l(1, 3) \leftarrow \min[4, 0 - 1] = -1 \tag{2.150c}$$

$$l(5, 1) \leftarrow \min[\infty, 2 + 2] = 4 \tag{2.150d}$$

$$l(2, 3) \leftarrow \min[\infty, -1 - 1] = -2 \tag{2.150e}$$

$$l(5, 2) \leftarrow \min[\infty, 2 + 1] = 3 \tag{2.150f}$$

$$l(2, 5) \leftarrow \min[2, -1 - 1] = -2 \tag{2.150g}$$

$$l(3, 5) \leftarrow \min[3, 1 - 1] = 0 \tag{2.150h}$$

$$l(5, 3) \leftarrow \min[\infty, 2 - 1] = 1 \tag{2.150i}$$

$$l(1, 5) \leftarrow \min[3, 0 - 1] = -1 \tag{2.150j}$$

At the end of this step, the resulting net is shown in Fig. 2.32.

Step 4. Return to Step 2.

Step 2. $k = 5$.

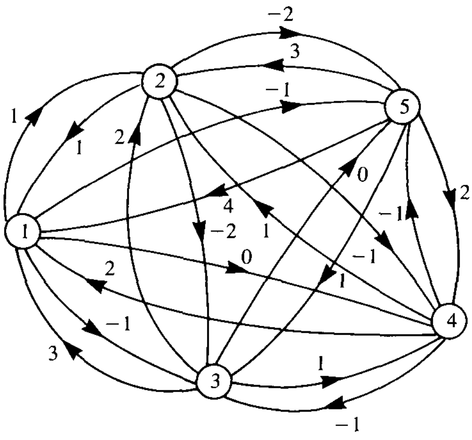

Fig. 2.32. The final net after the application of the Floyd–Warshall algorithm to the net of Fig. 2.31.

Step 3. Nothing is changed.

Step 4. Stop. The final net is shown in Fig. 2.32. Each of its arcs (i, j) is a basic arc, meaning that its length is the distance from i to j in the net of Fig. 2.31 or 2.32.

Having found the distances between all pairs of nodes, we need to ascertain the internal nodes along a shortest directed path. For this we define a 5×5 matrix $\boldsymbol{P}$ of (2.145). Initially, we let $p_{ij} = j$ for all i and j or

$$\boldsymbol{P} = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array} \begin{array}{c} \begin{array}{ccccc} 1 & 2 & 3 & 4 & 5 \end{array} \\ \begin{bmatrix} 1 & 2 & 3 & 4 & 5 \\ 1 & 2 & 3 & 4 & 5 \\ 1 & 2 & 3 & 4 & 5 \\ 1 & 2 & 3 & 4 & 5 \\ 1 & 2 & 3 & 4 & 5 \end{bmatrix} \end{array} = [p_{ij}] \tag{2.151}$$

In (2.147*a*), when we set $l(4, 2)$ to $l(4, 1) + l(1, 2) = 2 + 1 = 3$, we also set

$$p_{42} = p_{41} = 1.$$

Likewise, in (2.148) when we set

$$l(1, 4) = l(1, 2) + l(2, 4) = 1 - 1 = 0 \tag{2.152a}$$

$$l(3, 4) = l(3, 2) + l(2, 4) = 2 - 1 = 1 \tag{2.152b}$$

$$l(1, 5) = l(1, 2) + l(2, 5) = 1 + 2 = 3 \tag{2.152c}$$

we also set

$$p_{14} = p_{12} = 2 \tag{2.153a}$$

$$p_{34} = p_{32} = 2 \tag{2.153b}$$

$$p_{15} = p_{12} = 2 \tag{2.153c}$$

respectively. At the end of computation, the resulting matrix becomes

$$\boldsymbol{P} = \begin{bmatrix} 1 & 2 & 2 & 2 & 2 \\ 4 & 2 & 4 & 4 & 4 \\ 2 & 2 & 3 & 2 & 2 \\ 1 & 3 & 3 & 4 & 5 \\ 4 & 4 & 4 & 4 & 5 \end{bmatrix} \tag{2.154}$$

To find a shortest directed path from 1 to 5, we first look at the entry p_{15} and find the first arc $(1, x)$. Since $p_{15} = 2$, $x = 2$. We next look at the entry $p_{25} = 4$. The second arc of the shortest directed path from 1 to 5 is $(2, 4)$. We then look at the entry $p_{45} = 5$, giving the third arc $(4, 5)$. Thus, a shortest directed path from 1 to 5 in the net of Fig. 2.31 is $(1, 2)(2, 4)(4, 5)$, whose length is -1 as indicated in Fig. 2.32.

2.4 ENUMERATION OF THE SHORTEST DIRECTED PATHS BY DECOMPOSITION

In many applications, a net $G(V, E, l)$ is sparse in that many of its node pairs are not connected directly by arcs. In such situations, we can regard the net as being composed of two overlapping nets $G_\alpha(V_\alpha, E_\alpha, l)$ and $G_\beta(V_\beta, E_\beta, l)$, as depicted symbolically in Fig. 2.33 with

$$V = V_\alpha \cup V_\beta \tag{2.155a}$$

$$Y = V_\alpha \cap V_\beta \tag{2.155b}$$

$$E = E_\alpha \cup E_\beta \tag{2.156a}$$

$$E_Y = E_\alpha \cap E_\beta \tag{2.156b}$$

where Y denotes the set of overlapping nodes and E_Y denotes the set of arcs (i, j) with $i, j \in Y$. The net $G(V, E, l)$ of Fig. 2.34, for example, may be decomposed into two overlapping nets $G_\alpha(V_\alpha, E_\alpha, l)$ and $G_\beta(V_\beta, E_\beta, l)$ as shown in Fig. 2.35 with $Y = \{2, 6\}$ and $E_Y = \{(6, 2)\}$. The nets G_α and G_β are the sectional directed graphs of G defined by the node sets V_α and V_β, and denoted by $G_\alpha = G[V_\alpha]$ and $G_\beta = G[V_\beta]$, respectively.

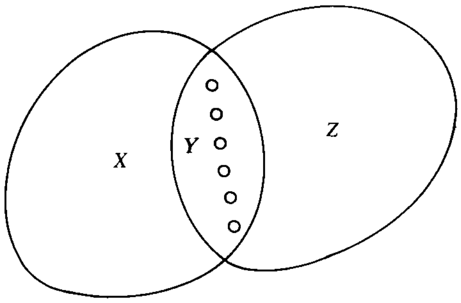

Fig. 2.33. Symbolic representation of a net as being composed of two overlapping nets.

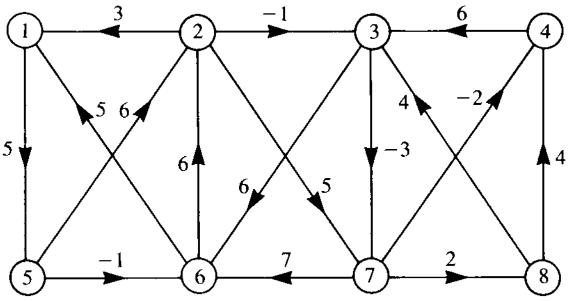

Fig. 2.34. A net $G(V, E, l)$ used to illustrate the decomposition of a net into two overlapping nets.

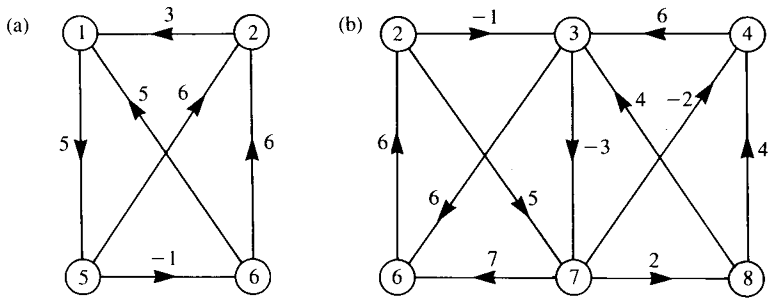

Fig. 2.35. A decomposition of the net of Fig. 2.34 into two overlapping nets: (a) $G_\alpha(V_\alpha, E_\alpha, l)$, (b) $G_\beta(V_\beta, E_\beta, l)$.

Let the node set V be partitioned into three subsets Y, X and Z:

$$V = X \cup Y \cup Z \tag{2.157}$$

where

$$X = V_\alpha - Y \tag{2.158a}$$

$$Z = V_\beta - Y \tag{2.158b}$$

If $\boldsymbol{D}$ is the distance matrix of $G(V, E, l)$, it can be partitioned in accordance with (2.157), as follows:

$$\boldsymbol{D} = [d_{ij}] = \begin{array}{c|ccc} & X & Y & Z \\ \hline X & \boldsymbol{D}_{XX} & \boldsymbol{D}_{XY} & \boldsymbol{D}_{XZ} \\ Y & \boldsymbol{D}_{YX} & \boldsymbol{D}_{YY} & \boldsymbol{D}_{YZ} \\ Z & \boldsymbol{D}_{ZX} & \boldsymbol{D}_{ZY} & \boldsymbol{D}_{ZZ} \end{array} \tag{2.159}$$

To simplify our notation, we shall use the symbol $d_{ij}(W)$ to denote the length of a shortest directed path from i to j in G subject to the restriction that all the nodes of the directed path belong to a subset W of the node set V. Thus, $d_{ij} = d_{ij}(V)$. Also, we use the symbol

$$\boldsymbol{D}_{XY}(W) = [d_{ij}(W)] \tag{2.160}$$

to denote the matrix of distances $d_{ij}(W)$ with $i \in X$ and $j \in Y$ with $\boldsymbol{D}_{XY} = \boldsymbol{D}_{XY}(V)$.

Recall that the original net $G(V, E, l)$ is composed of two nets $G_\alpha(V_\alpha, E_\alpha, l)$ and $G_\beta(V_\beta, E_\beta, l)$ overlapping at the nodes of Y. First, we apply the Floyd–Warshall algorithm to the net G_α, and obtain the matrix

$$\begin{bmatrix} \boldsymbol{D}_{XX}(V_\alpha) & \boldsymbol{D}_{XY}(V_\alpha) \\ \boldsymbol{D}_{YX}(V_\alpha) & \boldsymbol{D}_{YY}(V_\alpha) \end{bmatrix} \tag{2.161}$$

of distances $d_{ij}(V_\alpha)$ between each pair of nodes in G_α. We then replace the lengths of the arcs between each pair of nodes of Y in G_β by those of $\boldsymbol{D}_{YY}(V_\alpha)$. Note that arcs not in E_β are considered as arcs in E_β with infinite length. Let the resulting net be designated as $G'_\beta(V_\beta, E'_\beta, l')$. We shall now apply the Floyd–Warshall algorithm to G'_β again. At the end of the triple operations, we claim that we obtain the matrix

$$\begin{bmatrix} \boldsymbol{D}_{YY}(V) & \boldsymbol{D}_{YZ}(V) \\ \boldsymbol{D}_{ZY}(V) & \boldsymbol{D}_{ZZ}(V) \end{bmatrix} \tag{2.162}$$

of distances $d_{ij}(V) = d_{ij}$ between each pair of nodes of V_β in G, the original net.

To verify our assertion, let P_{ij} be a shortest directed path from i to j in G with $i, j \in V_\beta$. If all the nodes of P_{ij} lie in Z, then $d_{ij}(V) = d_{ij}(Z)$ and the triple operations on G'_β will yield the correct distance. Thus, assume that P_{ij} contains nodes in X and/or Y. Such a path P_{ij} is symbolically shown in Fig. 2.36. Since i and j are in V_β and Y is the set of overlapping nodes, the subpaths of P_{ij} that contain the nodes of X must begin and end in the set Y. The subpath $P_{y_1y_2}$ from y_1 to y_2 and the subpath $P_{y_3y_4}$ from y_3 to y_4 in Fig. 2.36 are two typical such subpaths. They may be the shortest directed paths

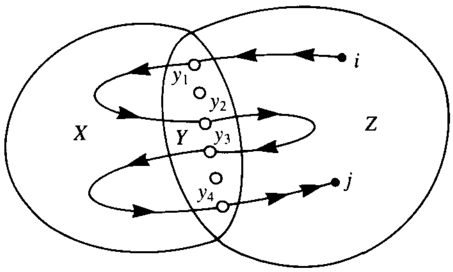

Fig. 2.36. A symbolic representation of a path P_{ij} containing nodes in X and/or Y.

from y_1 to y_2 and from y_3 to y_4 in G_α, respectively. Knowing $d_{y_1y_2}(V_\alpha)$ and $d_{y_3y_4}(V_\alpha)$ from (2.161), we effectively created two arcs (y_1, y_2) and (y_3, y_4) in G'_β with $l'(y_1, y_2) = d_{y_1y_2}(V_\alpha)$ and $l'(y_3, y_4) = d_{y_3y_4}(V_\alpha)$, as shown in Fig. 2.37. The shortest directed path P_{ij} from i to j can be replaced by another directed path of the same length in G'_β consisting of the subpath P_{iy_1}, (y_1, y_2), the subpath $P_{y_2y_3}$, (y_3, y_4), and the subpath P_{y_4j}. This means that the triple operations performed on G'_β will give the correct distance from i to j in G.

Having obtained the distances between each pair of nodes in V_β, we now replace the distances between each pair nodes of Y in G_α by those of $D_{YY}(V)$ of (2.162), and denote the resulting net by $G'_\alpha(V_\alpha, E'_\alpha, l')$. We apply the Floyd–Warshall algorithm again to G'_α. At the end of the triple operations, we obtain the matrix

$$\begin{bmatrix} \boldsymbol{D}_{XX}(V) & \boldsymbol{D}_{XY}(V) \\ \boldsymbol{D}_{YX}(V) & \boldsymbol{D}_{YY}(V) \end{bmatrix} \tag{2.163}$$

of distances $d_{ij}(V) = d_{ij}$ between each pair of nodes of V_α in G, the original net.

Finally, to calculate the elements in $\boldsymbol{D}_{XZ} = \boldsymbol{D}_{XZ}(V)$ and $\boldsymbol{D}_{ZX} = \boldsymbol{D}_{ZX}(V)$,

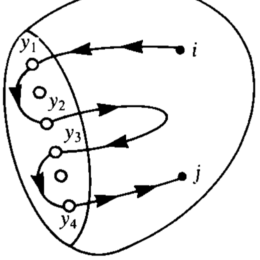

Fig. 2.37. A symbolic representation of the creation of two arcs (y_1, y_2) and (y_3, y_4) of equivalent lengths.

we use the formulas

$$d_{ij} = d_{ij}(V) = \min_{y \in Y} [d_{iy}(V) + d_{yj}(V)] \tag{2.164a}$$

$$d_{ji} = d_{ji}(V) = \min_{y \in Y} [d_{jy}(V) + d_{yi}(V)] \tag{2.164b}$$

for $i \in X$ and $j \in Z$. This follows directly from the observation that any directed path from $i \in X$ to $j \in Z$ or from j to i must pass at least one node y in Y, the minimum of which is certainly the minimum distance in G.

We now demonstrate the reduction in the number of operations required when distances are calculated by the decomposition procedure. Recall that the Floyd–Warshall algorithm requires approximately n^3 operations to find the distances between every pair of nodes in an n-node net. If the node set is decomposed as in (2.157) with $|X| = n_1$, $|Y| = n_2$ and $|Z| = n_3$, then we need approximately $(n_1 + n_2)^3$ operations in calculating (2.161), $(n_2 + n_3)^3$ operations in calculating (2.162), $(n_1 + n_2)^3$ in recalculating (2.163), and $2n_1n_2n_3$ operations in applying (2.164). Thus, the total number of operations required is

$$2(n_1 + n_2)^3 + (n_2 + n_3)^3 + 2n_1n_2n_3 \tag{2.165}$$

As a specific example, let $n_1 = n_3 = 20$ and $n_2 = 5$. The decomposition algorithm requires approximately 50875 operations, whereas the direct application of the Floyd–Warshall algorithm will need 91125 operations, showing that decomposition can reduce the number of operations by a factor 1.8.

We illustrate the decomposition procedure by the following examples.

Example 2.14

Consider the sparse net $G(V, E, l)$ of Fig. 2.38 which can be decomposed into two nets $G_\alpha(V_\alpha, E_\alpha, l)$ and $G_\beta(V_\beta, E_\beta, l)$ as shown in Fig. 2.39.

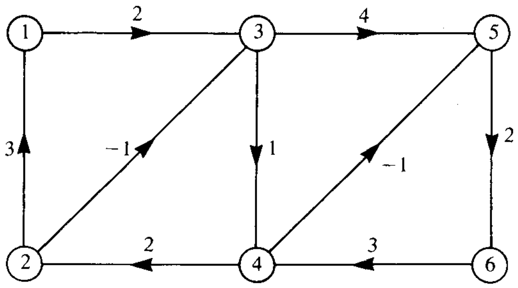

Fig. 2.38. A sparse net $G(V, E, l)$ used to illustrate the decomposition procedure.

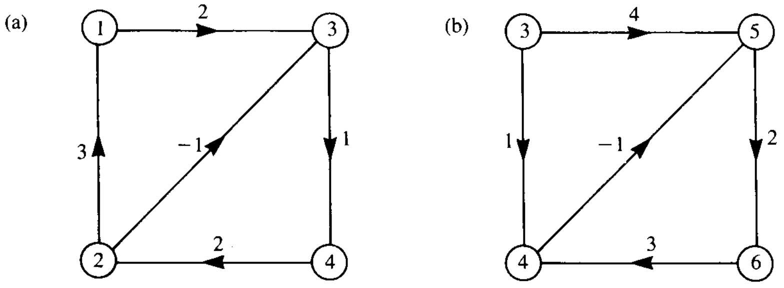

Fig. 2.39. The net of Fig. 2.38 decomposed into two overlapping nets: (a) $G_\alpha(V_\alpha, E_\alpha, l)$, (b) $G_\beta(V_\beta, E_\beta, l)$.

Following (2.157) and (2.158), we can make the following identifications:

$$X = \{1, 2\}, \qquad Y = \{3, 4\}, \qquad Z = \{5, 6\} \tag{2.166a}$$

$$V_\alpha = \{1, 2, 3, 4\}, \qquad V_\beta = \{3, 4, 5, 6\} \tag{2.166b}$$

Applying the Floyd–Warshall algorithm to G_α, we obtain the net of Fig. 2.40, whose distance matrix is given by

$$\begin{array}{c|cc:cc} & 1 & 2 & 3 & 4 \\ \hline 1 & 0 & 5 & 2 & 3 \\ 2 & 3 & 0 & -1 & 0 \\ \hdashline 3 & 6 & 3 & 0 & 1 \\ 4 & 5 & 2 & 1 & 0 \end{array} = \begin{bmatrix} \boldsymbol{D}_{XX}(V_\alpha) & \boldsymbol{D}_{XY}(V_\alpha) \\ \boldsymbol{D}_{YX}(V_\alpha) & \boldsymbol{D}_{YY}(V_\alpha) \end{bmatrix} \tag{2.167}$$

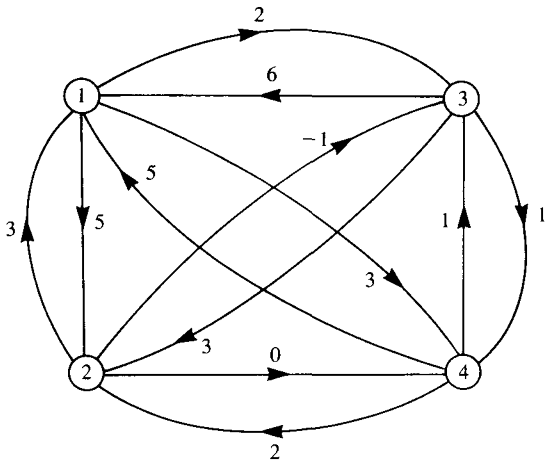

Fig. 2.40. The resulting net after the application of the Floyd–Warshall algorithm to G_α.

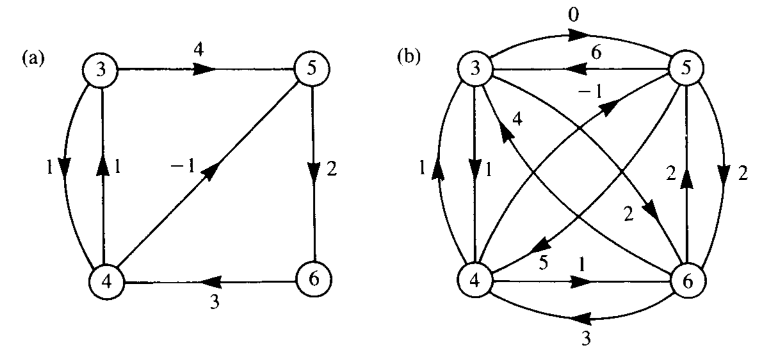

Fig. 2.41. (a) The net $G'_\beta(V_\beta, E'_\beta, l')$ obtained from G_β after the insertion of an arc (4, 3) of length 1. (b) The resulting net after the application of the Floyd–Warshall algorithm to G'_β.

We next insert an arc (4, 3) of length 1 in G_β to yield the net G'_β of Fig. 2.41(a), and then apply the Floyd–Warshall algorithm to G'_β. The resulting net is shown in Fig. 2.41(b), the distance matrix of which is found to be

$$\begin{array}{c|cc:cc|} & 3 & 4 & 5 & 6 \\ \hline 3 & 0 & 1 & 0 & 2 \\ 4 & 1 & 0 & -1 & 1 \\ \hdashline 5 & 6 & 5 & 0 & 2 \\ 6 & 4 & 3 & 2 & 0 \end{array} = \begin{bmatrix} \boldsymbol{D}_{YY}(V) & \boldsymbol{D}_{YZ}(V) \\ \boldsymbol{D}_{ZY}(V) & \boldsymbol{D}_{ZZ}(V) \end{bmatrix} \tag{2.168}$$

We now insert an arc (4, 3) of length 1 in G_α to yield the net G'_α of Fig. 2.42(a), and then apply the Floyd–Warshall algorithm to G'_α. The

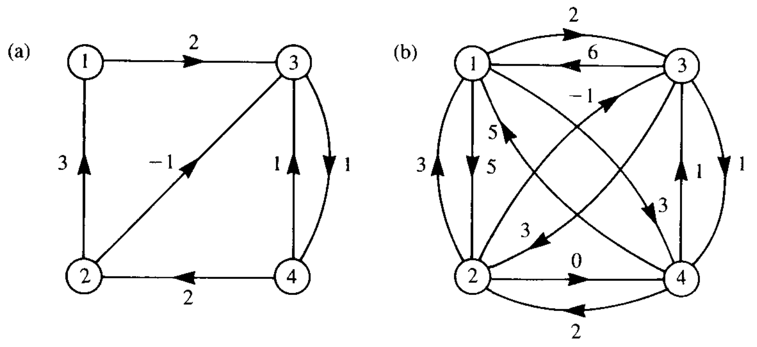

Fig. 2.42. (a) The net $G'_\alpha(V_\alpha, E'_\alpha, l')$ obtained from G_α after the insertion of an arc (4, 3) of length 1. (b) The resulting net after the application of the Floyd–Warshall algorithm to G'_α.

resulting net is shown in Fig. 2.42(b), the distance matrix of which is obtained as

$$\begin{array}{c|cc:cc} & 1 & 2 & 3 & 4 \\ \hline 1 & 0 & 5 & 2 & 3 \\ 2 & 3 & 0 & -1 & 0 \\ \hdashline 3 & 6 & 3 & 0 & 1 \\ 4 & 5 & 2 & 1 & 0 \end{array} = \begin{bmatrix} \boldsymbol{D}_{XX}(V) & \boldsymbol{D}_{XY}(V) \\ \boldsymbol{D}_{YX}(V) & \boldsymbol{D}_{YY}(V) \end{bmatrix} \tag{2.169}$$

Finally, we use (2.164) to calculate the elements of $\boldsymbol{D}_{XZ}$ and $\boldsymbol{D}_{ZX}$, as follows:

$$d_{15} = \min [d_{13} + d_{35}, d_{14} + d_{45}] = \min[2 + 0, 3 - 1] = 2 \tag{2.170a}$$

$$d_{16} = \min [d_{13} + d_{36}, d_{14} + d_{46}] = \min [2 + 2, 3 + 1] = 4 \tag{2.170b}$$

$$d_{25} = \min [d_{23} + d_{35}, d_{24} + d_{45}] = \min [-1 + 0, 0 - 1] = -1 \tag{2.170c}$$

$$d_{26} = \min [d_{23} + d_{36}, d_{24} + d_{46}] = \min [-1 + 2, 0 + 1] = 1 \tag{2.170d}$$

$$d_{51} = \min [d_{53} + d_{31}, d_{54} + d_{41}] = \min [6 + 6, 5 + 5] = 10 \tag{2.171a}$$

$$d_{61} = \min [d_{63} + d_{31}, d_{64} + d_{41}] = \min [4 + 6, 3 + 5] = 8 \tag{2.171b}$$

$$d_{52} = \min [d_{53} + d_{32}, d_{54} + d_{42}] = \min [6 + 3, 5 + 2] = 7 \tag{2.171c}$$

$$d_{62} = \min [d_{63} + d_{32}, d_{64} + d_{42}] = \min [4 + 3, 3 + 2] = 5 \tag{2.171d}$$

Combining (2.168)–(2.170) gives the complete distance matrix of G:

$$\boldsymbol{D} = \begin{array}{c|cc:cc:cc} & 1 & 2 & 3 & 4 & 5 & 6 \\ \hline 1 & 0 & 5 & 2 & 3 & 2 & 4 \\ 2 & 3 & 0 & -1 & 0 & -1 & 1 \\ \hdashline 3 & 6 & 3 & 0 & 1 & 0 & 2 \\ 4 & 5 & 2 & 1 & 0 & -1 & 1 \\ \hdashline 5 & 10 & 7 & 6 & 5 & 0 & 2 \\ 6 & 8 & 5 & 4 & 3 & 2 & 0 \end{array} = \begin{bmatrix} \boldsymbol{D}_{XX} & \boldsymbol{D}_{XY} & \boldsymbol{D}_{XZ} \\ \boldsymbol{D}_{YX} & \boldsymbol{D}_{YY} & \boldsymbol{D}_{YZ} \\ \boldsymbol{D}_{ZX} & \boldsymbol{D}_{ZY} & \boldsymbol{D}_{ZZ} \end{bmatrix} \tag{2.172}$$

each of whose elements d_{ij} is the distance from i to j in the original net $G(V, E, l)$ of Fig. 2.38.

2.5 SUMMARY AND SUGGESTED READING

In this chapter, we discussed the problem of finding the shortest directed paths from one node to all the other nodes and between all pairs of nodes in a weighted directed network called a net. The weights of the arcs of the net are real numbers that can be positive, zero or negative. However, for the problem to be meaningful, we assume that all the nets under consideration do not contain any negative directed circuits. The shortest directed path problem is fundamental in that it often occurs as a subproblem of other optimization problems.

For the problem of finding the shortest directed paths from one node to all other nodes, we presented four algorithms. The Dijkstra algorithm is most efficient because the required computation for an n-node net is $O(n^2)$. However, the algorithm is applicable only to those nets whose arc lengths are nonnegative. The basic idea of the Dijkstra algorithm is that it iterates on the length of the shortest directed paths. Initially, the shortest among the shortest directed paths is identified. At each iteration, the next shortest among the shortest directed paths is generated. The algorithm terminates at the end of the $(n-1)$th iteration. The Ford–Moore–Bellman algorithm is general and is applicable to nets with negative arc lengths. The algorithm iterates on the number of arcs in a directed path. At the end of the kth iteration, the labels on the nodes represent the lengths of those shortest directed paths from the source node containing $k+1$ or fewer arcs. The algorithm terminates with at most $n-1$ iterations, and requires $O(n^3)$ rather than $O(n^2)$ as in the Dijkstra algorithm. The Yen algorithm is basically the same as the Ford–Moore–Bellman algorithm except that it iterates on the number of blocks in a directed path. At the end of the kth iteration, it gives the lengths of all the shortest directed paths using at most k blocks. The algorithm terminates with at most $n-1$ iterations, and requires roughly half the work of the Ford–Moore–Bellman algorithm. An additional advantage of the Yen algorithm is that the storage requirement of the program is also reduced. Finally, the Ford–Fulkerson algorithm is very similar to the Ford–Moore–Bellman algorithm in that it sequentially reduces the labels on the nodes by examining the arcs until the conditional distances become the true distances.

For the problem of finding the shortest directed paths between all pairs of nodes, we presented two algorithms. The matrix algorithm uses the binary operation minaddition. The elements of the kth power of the connection matrix of a net in the sense of minaddition represent the lengths of the shortest directed paths in the net using at most k arcs. The algorithm terminates with at most α iterations, where α is the least integer greater than or equal to $\log_2(n-1)$, and requires $O(\alpha n^3)$ computations. To reduce the number of operations, we introduced two minaddition procedures known as the forward process and the reverse process. It can be shown that for a net with nonnegative arc lengths one forward process followed by one backward process is sufficient to generate all the distances between all pairs of nodes, and that while two forward processes are not enough, three are always sufficient. The Floyd–Warshall algorithm iterates on a set of nodes and uses the triple operation to create basic arcs between each pair of nodes not connected by a basic arc. At the end of the triple operation, the resulting net consists only of basic arcs. The computational complexity of the algorithm is $O(n^3)$, the same as if the Dijkstra algorithm was repeated for each node chosen as the source node.

In many applications, the nets are sparse in that many of the node pairs are not connected directly by arcs. In such cases, we can decompose a net into two overlapping nets. We showed that the distances between all pairs of nodes in the original net can be calculated from those of the component nets, and that the resulting reduction in the number of operations may be considerable.

Finally, we mention an intuitive way of solving the shortest path problem suggested by Minty (1957) for the case of undirected nets. One simply builds a physical model of the net using strings to connect the nodes, the lengths of which are proportional to the given arc lengths. By taking the source in one hand and the sink in the other, one solves the shortest path problem by stretching the source and the sink.

For general information and other aspects of the shortest directed path problem, see Dreyfus (1969) and Yen (1975). Yen's extensive survey article is 169 pages long and contains a bibliography of 300 papers on the subject. For the algorithms discussed in this chapter, we refer the reader to the original papers cited in the text. For the matrix algorithm, see Pollack and Wiebenson (1960), Narahari Pandit (1961), and Hu (1967). For decomposition of a sparse net, see Land and Stairs (1967), Hu (1968), Hu and Torres (1969), Yen (1971*a*), and Blewett and Hu (1977).

REFERENCES

Beardwood, J., Halton, J. H. and Hammersley, J. M. (1959), "The shortest path through many points," *Proc. Cambridge Phil. Soc.*, vol. 55, pp. 299–327.

Bellman, R. E. (1958), "On a routing problem," *Quart. Appl. Math.*, vol. 16, pp. 87–90.

Bilde, O. and Krarup, J. (1969), "A modified cascade algorithm for shortest paths," *Metra*, vol. 8, pp. 231–241.

Blewett, W. J. and Hu, T. C. (1977), "Tree decomposition algorithm for large networks," *Networks*, vol. 7, pp. 289–296.

Butas, L. F. (1968), "A directionally oriented shortest path algorithm," *Transportation Res.*, vol. 2, pp. 253–268.

Carson, J. S. and Law, A. M. (1977), "A note on Spira's algorithm for all-pairs shortest paths problem," *SIAM J. Computing*, vol. 6, pp. 696–699.

Cartaino, T. F. and Dreyfus, S. E. (1957), "Application of dynamic programming to the airplane minimum time-to-climb problem," *Aero Engr. Rev.*, vol. 16, pp. 74–77.

Chen, W. K. (1966*a*), "Boolean matrices and switching nets," *Math. Mag.*, vol. 39, pp. 1–8.

Chen, W. K. (1966*b*), "On directed trees and directed k-trees of a digraph and their generation," *SIAM J. Applied Math.*, vol. 14, pp. 550–559.

Cooke, K. L. and Halsey, E. (1966), "The shortest route through a network with times," *J. Math. Anal. and Appl.*, vol. 14, pp. 493–498.

Dantzig, G. B. (1967), "All shortest routes in a graph," in *Theory of Graphs*, International Symposium. New York: Gordon and Breach, pp. 91–92.

Dial, R. B. (1969), "Algorithm 360: Shortest path forest with topological ordering," *Comm. ACM*, vol. 12, pp. 632–633.

Dial, R. B., Glover, F., Karney, D. and Klingman, D. (1979), "A computational analysis of alternative algorithms for finding shortest path trees," *Networks*, vol. 9, pp. 215–248.

Dijkstra, E. W. (1959), "A note on two problems in connexion with graphs," *Numerische Math.*, vol. 1, pp. 269–271.

Dreyfus, S. E. (1969), "An appraisal of some shortest path algorithms," *Operations Res.*, vol. 17, pp. 395–412.

Farbey, B. A., Land, A. H. and Murchland, J. D. (1967), "The cascade algorithm for finding all shortest distances in a directed graph," *Mangement Sci.*, vol. 14, pp. 19–28.

Floyd, R. W. (1962), "Algorithm 97, shortest path," *Comm. ACM*, vol. 5, p. 345.

Ford, L. R. Jr (1956), "Network flow theory," The RAND Corp., P-923, Santa Monica, Calif.

Ford, L. R. Jr and Fulkerson, D. R. (1962), *Flows in Networks*, Chapter 3, Princeton, N.J.: Princeton University Press.

Frederickson, G. N. (1987), "Fast algorithms for shortest paths in planar graphs, with applications," *SIAM J. Computing*, vol. 16, pp. 1004–1022.

Fredman, M. L. (1976), "New bounds on the complexity of the shortest path problem," *SIAM J. Computing*, vol. 5, pp. 83–89.

Glover, F., Klingman, D. and Napier, A. (1974), "A note on finding all shortest paths," *Transportation Sci.*, vol. 8, pp. 3–12.

Hakimi, S. L. and Yau, S. S. (1965), "Distance matrix of a graph and its realizability," *Quart. Appl. Math.*, vol. 22, pp. 305–317.

Hesse, R. (1972), "Solution of the shortest route problem using the assignment technique," *Decision Sci.*, vol. 3, pp. 1–13.

Hoffman, W. and Pavley, R. (1959), "A method for the solution of the *N*th best path problem," *J. ACM*, vol. 6, pp. 506–514.

Hoffman, A. J. and Winograd, S. (1972), "Finding all shortest distances in a directed network," *IBM J. Res. Develop.*, vol. 16, pp. 412–414.

Hu, T. C. (1967), "Revised matrix algorithms for shortest paths," *SIAM J. Applied Math.*, vol. 15, pp. 207–218.

Hu, T. C. (1968), "A decomposition algorithm for shortest paths in a network," *Operations Res.*, vol. 16, pp. 91–102.

Hu, T. C. (1982), *Combinatorial Algorithms*, Chapter 1, Reading, Mass.: Addison–Wesley.

Hu, T. C. and Torres, W. T. (1969), "Shortcut in the decomposition algorithm for shortest paths in a network," *IBM J. Res. Develop.*, vol. 13, pp. 387–390.

Ibaraki, T. (1970), "Shortest path problems visiting specified nodes," *Trans. Inst. Elec. Comm. Engr Japan,* vol. 53-A, pp. 639–646.

Johnson, D. B. (1973), "A note on Dijkstra's shortest path algorithm," *J. ACM,* vol. 20, pp. 385–388.

Johnson, D. B. (1977), "Effcient algorithms for shortest paths in sparse networks," *J. ACM,* vol. 24, pp. 1–13.

Kershenbaum, A. (1981), "A note on finding shortest path trees," *Networks,* vol. 11, pp. 399–400.

Knuth, D. E. (1977), "A generalization of Dijkstra's algorithm," *Inf. Process. Lett.,* vol. 6, pp. 1–5.

Land, A. H. and Stairs, S. W. (1967), "The extension of the cascade algorithm to large graphs," *Management Sci.,* vol. 14, pp. 29–33.

Lawler, E. L. (1972), "A procedure for computing the K best solutions to discrete optimization problems and its application to the shortest path problem," *Management Sci.,* vol. 18, pp. 401–405.

Mills, G. (1966), "A decomposition algorithm for the shortest-route problem," *Operations Res.,* vol. 14, pp. 279–291.

Mills, G. (1968), "A heuristic approach to some shortest route problems," *Can. Operational Res. Soc. J.,* vol. 6, pp. 20–25.

Minieka, E. (1974), "On computing sets of shortest paths in a graph," *Comm. ACM,* vol. 17, pp. 351–353.

Minty, G. J. (1957), "A comment on the shortest-route problem," *Operations Res.,* vol. 5, p. 724.

Minty, G. J. (1958), "A variant on the shortest-route problem," *Operations Res.,* vol. 6, pp. 882–883.

Moffat, A. and Takaoka, T. (1987), "An all pairs shortest path algorithm with expected time $O(n^2 \log n)$," *SIAM J. Computing,* vol. 16, pp. 1023–1031.

Moore, E. F. (1957), "The shortest path through a maze," *Proc. Int. Symp. on the Theory of Switching,* Part II, April 2–5, p. 285. The Annals of the Computation Laboratory of Harvard University, vol. 30, Cambridge, Mass.: Harvard University Press.

Mori, M. and Nishimura, T. (1968), "Solution of the routing problem through a network by matrix method with auxiliary nodes," *Transportation Res.,* vol. 1, pp. 165–180.

Murchland, J. D. (1969), "Bibliography of the shortest route problem," London School of Business Studies, Report LBS-TNT-6.2 (revised 1969).

Nakamori, M. (1972), "A note on the optimality of some all-shortest-path algorithms," *J. Operations Res. Soc. Japan,* vol. 15, pp. 201–204.

Narahari Pandit, S. N. (1961), "The shortest-route problem—an addendum," *Operations Res.,* vol. 9, pp. 129–132.

Nemhauser, G. L. (1972), "A generalized permanent label setting algorithm for the shortest path between specified nodes," *J. Math. Anal. and Appl.,* vol. 38, pp. 328–334.

Nicholson, T. A. J. (1966), "Finding the shortest route between two points in a network," *Computer J.,* vol. 9, pp. 275–280.

Pape, U. (1974), "Implementation and efficiency of Moore-algorithms for the shortest route problem," *Math. Programming,* vol. 7, pp. 212–222.

Peart, R. M., Randolph, P. H. and Bartlett, T. E. (1960), "The shortest-route problem," *Operations Res.,* vol. 8, pp. 866–867.

Perko, A. (1965), "Some computational notes on the shortest route problem," *Computer J.,* vol. 8, pp. 19–20.

Petitfrere, M. (1972), "Sur l'Algorithme de Dijkstra l'Obtention des plus Courts Chemins dans un Graphe," *Cahiers du Centre d'Etudes de Recherche Operationelle (Belgium),* vol. 13, pp. 111–123.

Pollack, M. and Wiebenson, W. (1960), "Solution of the shortest-route problem—a review," *Operations Res.,* vol. 8, pp. 224–230.

Shier, D. R. (1973), "A decomposition algorithm for optimality problems in tree-structured networks," *Discrete Math.,* vol. 6, pp. 175–189.

Shimbel, A. (1951), "Application of matrix algebra to communication nets," *Bull. Math. Biophysics,* vol. 13, pp. 165–178.

Spira, P. M. (1973), "A new algorithm for finding all shortest paths in a graph of positive arcs in average time $O(n^2 \log_2 b)$," *SIAM J. Computing,* vol. 2, pp. 28–32.

Tabourier, Y. (1973), "All shortest distances in a graph: An improvement to Dantzig's inductive algorithm," *Discrete Math.,* vol. 4, pp. 83–87.

Tarjan, R. E. (1981), "A unified approach to path problems," *J. ACM,* vol. 28, pp. 577–593.

Wagner, R. A. (1976), "A shortest path algorithm for edge-sparse graphs," *J. ACM,* vol. 23, pp. 50–57.

Warshall, S. (1962), "A theorem on Boolean matrices," *J. ACM,* vol. 9, pp. 11–12.

Weimer, D. L. (1963), "A serial technique to determine minimum paths," *Comm. ACM,* vol. 6, p. 664.

Whiting, P. D. and Hillier, J. A. (1960), "A method for finding the shortest route through a road network," *Operational Res. Quart.,* vol. 11, pp. 37–40.

Williams, T. A. and White, G. P. (1973), "A note on Yen's algorithm for finding the length of all shortest paths in N-node nonnegative-distance networks," *J. ACM,* vol. 20, pp. 389–390.

Yang, L. and Chen, W. K. (1989), "An extension of the revised matrix algorithm," *IEEE Proc. Int. Symp. on Circuits and Systems,* Portland, Oregon, May 8–11, pp. 1996–1999.

Yen, J. Y. (1970), "An algorithm for finding shortest routes from all source nodes to a given destination in general networks," *Quart. Appl. Math.,* vol. 27, pp. 526–530.

Yen, J. Y. (1971*a*), "On Hu's decomposition algorithm for shortest paths in a network," *Operations Res.,* vol. 19, pp. 983–985.

Yen, J. Y. (1971*b*), "On the efficiencies of algorithms for detecting negative loops in networks," *Santa Clara Business Rev.,* pp. 52–58.

Yen, J. Y. (1972*a*), "Finding the lengths of all shortest paths in N-node nonnegative-distance complete networks using $\frac{1}{2}N^3$ additions and N^3 comparisons," *J. ACM,* vol. 19, pp. 423–425.

Yen, J. Y. (1972*b*), "On the efficiency of a direct search method to locate negative cycles in a network," *Management Sci.*, vol. 19, pp. 333–335.

Yen, J. Y. (1975), "Shortest path network problems," *Math. Systems in Economics,* vol. 18, pp. 1–169.