Contents

4 Shortest Chain Algorithms **327**

4.1 LP Formulation of the Unconstrained Shortest Chain Problem as a Minimum Cost Flow Problem 329

4.2 Label Setting Methods for Shortest Chains from a Specified Origin 336

4.3 Label Correcting Methods for Shortest Chains from a Specified Origin 342

4.4 Shortest Chains From a Specified Origin in an Acyclic Network 355

4.5 Matrix Methods for Shortest Chains Between All Pairs of Nodes 357

4.6 Sensitivity Analysis in the Shortest Chain Problem 362

4.7 Algorithm for Ranking Chains in Nondecreasing Order of Length 364

4.8 Exercises 367

4.9 References 379

Chapter 4

Shortest Chain Algorithms

The problem of finding a shortest chain from a specified origin (or source) node to a specified destination (or sink) node is a fundamental problem that appears in many applications. It generates essential information in transportation, routing, and communications applications; here the lengths of the arcs are the geographical distances (or the travel times) associated with them. This problem appears as a subproblem in each step in solving multicommodity flow problems using the revised simplex method on an arc chain formulation (Section 5.11); in this application the lengths of the arcs are numbers derived by the algorithm, and they depend on the dual solution in that step. In critical path methods (CPM, see Chapter 7) arcs in the acyclic project network correspond to jobs, the length of each is the negative of the time duration required to complete the corresponding job, and the shortest chain problem appears in the task of scheduling the various jobs over time. There are also applications in equipment replacement (Garcia-Diaz and Liebman [1980]), preparation of travel time and distance charts (Golden and Magnanti [1978]), vehicle routing and scheduling (Christofides, Mingozzia and Toth [1981], Golden and Magnanti [1978]), capacity planning (Doulbiez and Ras [1975]), design and/or expansion of transportation and communication networks (Golden and Magnanti [1978], Schwartz and Stern [1980]), and in several other areas. The shortest chain problem also appears as a subproblem in algorithms for the solution of other network flow and discrete optimization problems. In this

chapter we discuss various algorithms for finding shortest chains in a network.

What we call a "chain" is called a "path" in some books. So, in the literature, the shortest chain problem is often referred to as the "shortest path problem" or the "shortest route problem."

Suppose there is an edge $(i;j)$ in the original network of length $c_{i;j}$. If $c_{i;j} \geqq 0$ we replace $(i;j)$ by the pair of arcs $(i,j),(j,i)$ both of length $c_{i;j}$. On the other hand, if $c_{i;j} < 0$, this operation creates a negative length circuit $(i,j),(j,i)$ in the transformed network (notice that this is not a circuit in the original network; see Section 1.2.2). As explained later, negative length circuits in the network create difficulties for solving shortest chain problems in it; hence we cannot replace a negative length edge $(i;j)$ by the pair of arcs $(i,j),(j,i)$ of the same length. When the original network contains negative length edges, but otherwise no negative length circuits, the problem of finding shortest simple chains in it requires special techniques based on matching algorithms (Tobin [1975]) but we will not discuss them because of their limited applicability.

Hence, we assume that if there are any edges in the original network, their lengths are $\geqq 0$, and we will replace each of them by a pair of arcs of the same length as described above. So in the sequel we assume that the network on which our shortest chain problem is defined is a directed connected network $\mathrm{G} = (\mathcal{N}, \mathcal{A}, c)$ with $c = (c_{ij})$ as the vector of arc lengths, and $|\mathcal{N}| = n$.

The length of any chain is the sum of the lengths of arcs in it. If there are parallel arcs with tail node i and head node j in G, only the shortest among them will be used on the shortest chains. Call this the arc (i,j) and eliminate all other arcs parallel to it from further consideration. We continue to denote the set of arcs by $\mathcal{A}$. Let $m = |\mathcal{A}|$.

We will discuss the problem of finding the shortest chains from a fixed origin to all the other nodes in G. Given a chain containing some circuits, all the arcs on these circuits can be eliminated, leaving a simple chain; this elimination will not increase the length of the chain if the lengths of these circuits are $\geqq 0$. Hence, if a chain exists from the origin to j, and G contains no negative length circuits, there exists a shortest chain from the origin to j which is simple. All shortest chains

found by algorithms discussed in this chapter will be simple chains.

If shortest chains from the origin to all the other nodes in G exist, by the result in Exercise 4.1 later on, there exists an outtree rooted at the origin such that the unique path in it between the origin and any other node is a shortest chain to that node. Such an outtree is called a **shortest chain tree** (also called a **shortest path tree** in the literature). The shortest chain tree stored using the predecessor labels, provides a convenient data structure for storing the shortest chains out of the origin.

Another problem that we will consider is that of finding shortest chains between every pair of nodes in G.

4.1 LP Formulation of the Unconstrained Shortest Chain Problem as a Minimum Cost Flow Problem

To find a shortest chain from 1 to n in G $= (\mathcal{N}, \mathcal{A}, c)$ is equivalent to the following problem of sending one unit of flow from 1 to n at minimum cost with 0, ∞, c as the lower bound, capacity and unit cost coefficient vectors for arc flows.

$$\begin{array}{rrcl} \text{Minimize} & \sum (c_{ij} f_{ij} : \text{ over } (i,j) & \in & \mathcal{A}) \\ \text{Subject to} & -f(i,\mathcal{N}) + f(\mathcal{N},i) & = & \begin{cases} -1 & \text{if } i = 1 \\ 0 & \text{if } i \neq 1, n \\ 1 & \text{if } i = n \end{cases} \quad (4.1) \\ & f_{ij} \geqq 0, \text{ for all } (i,j) & \in & \mathcal{A} \end{array}$$

The chain may traverse through an arc any number of times since this is an unconstrained shortest chain problem. The flow on each arc represents the number of times the chain traverses it (if the capacity of an arc is made = 1, then we get the flow formulation of the constrained shortest chain problem in which you cannot pass through that arc more than once). (4.1) has a redundant constraint that can be eliminated; we choose it to be the one corresponding to the origin, node 1. The

dual of the resulting problem involves dual variables which are node prices π_i. It is

$$\begin{array}{rrcll} \text{Maximize } \pi_n & - & \pi_1 & & \\ \text{Subject to} \quad \pi_j & - & \pi_i \leqq c_{ij}, & \text{for each } (i,j) \in \mathcal{A} & (4.2) \\ \pi_1 & = & 0. & & \end{array}$$

Every basic vector for (4.1) consists of flow variables associated with arcs in a spanning tree in G. A feasible basic vector for it corresponds to a spanning tree $\mathbb{T}$ such that the unique path in $\mathbb{T}$ from 1 to n is a chain and vice versa. In the corresponding BFS the flow amounts are equal to 1 on all the arcs of that chain, and 0 on all the other arcs. Hence, in every BFS $f = (f_{ij})$ of (4.1), all f_{ij} are 0 or 1, and $\{(i,j) : f_{ij} = 1\}$ is the set of arcs on a simple chain from 1 to n. Thus each extreme point of the set of feasible solutions of (4.1), i.e., each BFS of (4.1), is the incidence vector of the set of arcs on a simple chain from 1 to n in G and vice versa.

Suppose there is a negative length circuit $\vec{\mathbb{C}}$ in G. Summing the dual constraints in (4.2) corresponding to arcs (i,j) on $\vec{\mathbb{C}}$ leads to the inconsistent inequality $\mathbf{0} \leqq$ **a negative number**. So, the dual problem (4.2) is infeasible, and by the duality theorem of LP, if (4.1) is feasible, the objective value in it is unbounded below. The solution of (4.1) which makes the objective function unbounded below corresponds to a flow that traverses around $\vec{\mathbb{C}}$ an infinite number of times. Hence, when the network contains a negative length circuit, unconstrained shortest chain algorithms will detect one such circuit and terminate with the unboundedness conclusion.

As an example consider the network in Figure 4.1. Arc lengths are entered on the arcs. A feasible flow vector $f(\mu)$, for $\mu \geqq 0$, is marked with the flow on each arc entered inside a box if it is $\neq 0$. As $\mu \to \infty$ the objective value of $f(\mu) \to -\infty$.

Even when G contains a negative length circuit, the constrained shortest chain problem of finding a minimum length simple chain from 1 to n is a well-posed problem. (4.1) has a finite number of BFSs, each associated with a simple chain from 1 to n in G. Hence this problem

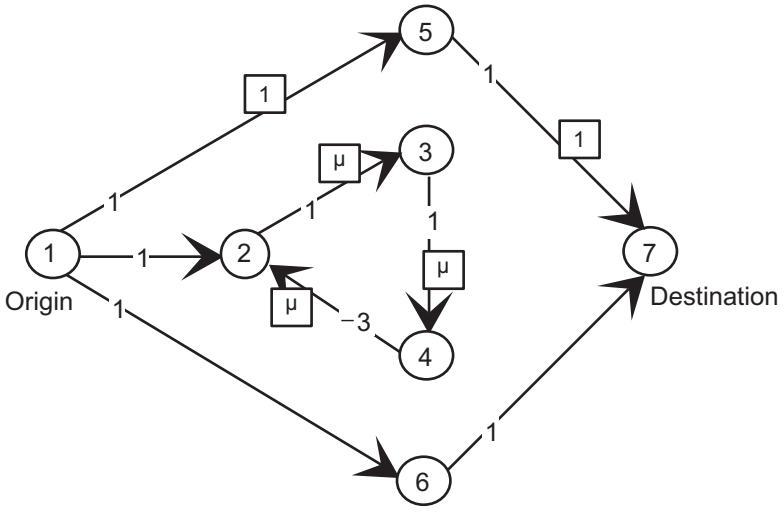

Figure 4.1: Network with a negative length circuit containing nodes 2, 3, 4.

is equivalent to that of finding a minimum cost BFS among the finite number of BFSs of the unbounded LP (4.1). In Section 2.2 we have seen that the maximum capacity cut problem is also a problem of this type. In general these are NP-hard combinatorial optimization problems. At the moment the only known algorithms for tackling them are enumerative algorithms whose computational requirements grow exponentially with the size of the problem in the worst case.

The network in Figure 4.1 has the special property that any node which lies on a negative length circuit does not lie on any chain from the origin to the destination. Under this property it is possible to solve the shortest simple chain problem efficiently; see Exercise 4.3.

In the shortest simple chain problem, unboundedness in the presence of negative length circuits occurs due to traversal around such a circuit an infinite number of times. From this, one may be tempted to think that the simple technique of imposing an upper bound of 1 on all the variables in (4.1) will take care of this difficulty. This capacitated

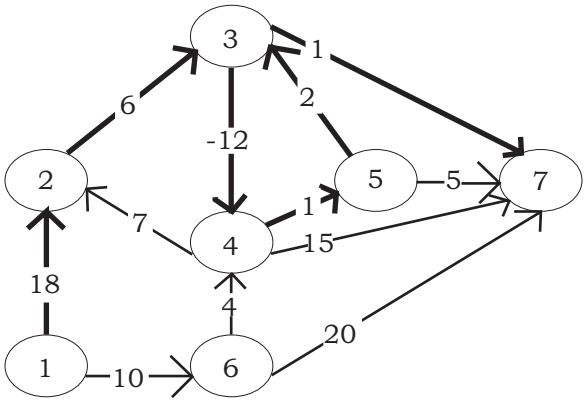

Figure 4.2:

problem has an integer optimum solution as long as there is a chain from 1 to n in G; suppose it is $\bar{f} = (\bar{f}_{ij})$. The set of arcs (i, j) on which $\bar{f}_{ij} = 1$ defines a chain from 1 to n which we denote by $\mathcal{C}_0$. $\mathcal{C}_0$ may not be simple; it may include some negative length circuits. Eliminate all circuits from $\mathcal{C}_0$; suppose this leaves the simple chain $\mathcal{C}_1$ from 1 to n. Unfortunately, $\mathcal{C}_1$ may not be a shortest simple chain from 1 to n in G as illustrated by the network in Figure 4.2. Arc lengths are entered on the arcs. The shortest simple chain from 1 to 7 in this network is 1, (1, 6), 6, (6, 4), 4, (4, 5), 5, (5, 3), 3, (3, 7), 7 of length 18. Problem (4.1) with capacities of 1 on all the arcs in this network has the optimum flow with flow amount of 1 on all the thick arcs, and 0 on others. The set of thick arcs includes the negative length circuit containing nodes 3,4,5. Elimination of this circuit from the set of thick arcs leaves the simple chain 1, (1, 2), 2, (2, 3), 3, (3, 7), 7, which is not a shortest simple chain from 1 to 7 in this network.

Let $\hat{f} = (\hat{f}_{ij})$ be a basic feasible flow vector for (4.1) and $\hat{\pi} = (\hat{\pi}_i)$, a node price vector in G. Let $\hat{\mathcal{C}}$ be the simple chain from 1 to n in G corresponding to $\hat{f}$; i.e., $\hat{f}$ is the incidence vector of $\hat{\mathcal{C}}$. By the duality theory of LP $\hat{f}, \hat{\pi}$ are respectively optimal to (4.1), (4.2), iff

$$\begin{aligned} \hat{\pi}_n &= \text{opt. obj. value in (4.2)} = \text{opt. obj. value in (4.1)} \\ &= \text{length of a shortest chain from 1 to } n \text{ in G} \end{aligned} \tag{4.3}$$

$$\hat{\pi}_j - \hat{\pi}_i = c_{ij} \text{ whenever } \hat{f}_{ij} = 1, \text{ i.e., for all } (i,j) \text{ on } \hat{\mathcal{C}} \tag{4.4}$$

$$\hat{\pi}_j - \hat{\pi}_i \stackrel{\leq}{=} c_{ij} \text{ for all } (i,j) \in \mathcal{A} \tag{4.5}$$

(4.5) are just the conditions for the node price vector $\hat{\pi}$ to be dual feasible. Conversely, given a dual feasible node price vector $\hat{\pi}$, if there exists a chain from 1 to n consisting only of arcs (i,j) for which $\hat{\pi}_j - \hat{\pi}_i = c_{ij}$, then that chain is a shortest chain.

Assume that G contains no negative length circuits. One can derive the above facts directly without appealing to the duality theory of LP. For this, define $\hat{\pi}_1 = 0$, and for $i \in \mathcal{N}, i \neq 1$, let $\hat{\pi}_i$ be ∞ if there exists no chain from 1 to i in G, or the length of a shortest chain from 1 to i otherwise. For $(i,j) \in \mathcal{A}$, if $\hat{\pi}_i < \infty$, $\hat{\pi}_i + c_{ij}$ is the length of a chain from 1 to j (it consists of a shortest chain from 1 to i and then the arc (i,j)), and hence $\hat{\pi}_i + c_{ij} \stackrel{\geq}{=} \hat{\pi}_j$, since $\hat{\pi}_j$ is the length of a shortest chain from 1 to j. So, $\hat{\pi} = (\hat{\pi}_i)$ must satisfy (4.5). Also the shortest chains themselves consist of arcs satisfying these conditions as equations. Also, given π feasible to (4.2) or (4.5), any chain from 1 to q consisting of arcs for which (4.5) holds as an equation must have length π_q, and hence is a shortest chain from 1 to q.

From this discussion it is clear that the value of the dual variable π_i in an optimum dual solution can be interpreted as being the length of a shortest chain from 1 to i in G. And, the vector $\hat{\pi} = (\hat{\pi}_i)$ of shortest chain lengths out of node 1 satisfy

$$\begin{aligned} \hat{\pi}_1 &= 0 \\ \hat{\pi}_j &= \text{min.}\{\hat{\pi}_i + c_{ij} : i \in \mathbf{B}_j\}, \text{ for all } j \in \mathcal{N}, j \neq 1 \end{aligned} \tag{4.6}$$

(4.6) are known as the **Bellman-Ford equations** for the shortest chain problem with node 1 as the origin node. They represent neces-

sary conditions for the vector of shortest chain lengths out of node 1. Conversely, if the vector $\pi = (\pi_i)$ satisfies the Bellman-Ford equations, and there is a chain from 1 to i of length π_i, then that chain is a shortest chain from 1 to i.

(4.1) is the LP formulation for finding a shortest chain from node 1 to the specified destination, node n. But its dual (4.2) involves variables which represent the lengths of the shortest chains from 1 to all the other nodes in G. Therefore it usually happens that to find a shortest chain from 1 to a specified destination, one may have to find the shortest chains from 1 to all the other nodes in the network.

Exercises

4.1 G $= (\mathcal{N}, \mathcal{A}, c)$ is a directed network with the origin at node 1. The simple chain $\mathcal{C}_i$ is a shortest chain from 1 to i in G. For every node p on $\mathcal{C}_i$ prove that $\mathcal{C}_i(p)$, the portion of $\mathcal{C}_i$ between 1 and p, is a shortest chain from 1 to p in G.

Let the simple chain $\mathcal{C}_j$ be a shortest chain from 1 to $j \neq i$, and $p \neq 1$ a common node on $\mathcal{C}_i$ and $\mathcal{C}_j$. Prove that $\mathcal{C}_i(p)$ and $\mathcal{C}_j(p)$ must have the same length. See Figure 4.3. Using this prove that there must exist simple chains from 1 to i, j respectively which are shortest chains to i, j satisfying: either these chains have no common node other than the origin 1; or if they have any common node $p \neq 1$, the portions of both these chains between 1 and p are identical.

Using this prove that if shortest chains exist in G from 1 to all the other nodes, then there exists a spanning outtree, $\mathbb{T}$, rooted at 1, on which all the paths from the root are shortest chains. Hence show that in this case there exists an optimum feasible basic vector for (4.1), corresponding to an outtree rooted at 1 which is a shortest chain tree.

4.2 G $= (\mathcal{N}, \mathcal{A}, c)$ is a connected directed network with c as the vector of arc lengths, and $|\mathcal{N}| = n$. Show that the following minimum cost flow problem is a formulation for the problem of finding shortest chains from the origin, 1, to all the other nodes in G.

$$\text{Minimize} \quad \sum (c_{ij} f_{ij} : \text{ over } (i,j) \in \mathcal{A})$$

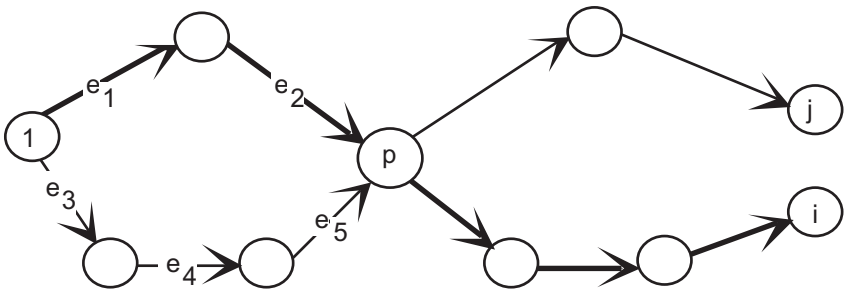

Figure 4.3: $\mathcal{C}_i$ is thick, and $\mathcal{C}_j$ is normal. $\mathcal{C}_i(p)$ consists of arcs e_1, e_2; $\mathcal{C}_j(p)$ of e_3, e_4, e_5.

$$\begin{aligned} \text{Subject to} \quad f(i,\mathcal{N}) - f(\mathcal{N},i) &= \begin{cases} n-1 & \text{, if } i = 1 \\ -1 & \text{, if } i \neq 1 \end{cases} \qquad (4.7) \\ f_{ij} &\geqq 0, \quad \text{for all } (i,j) \in \mathcal{A} \end{aligned}$$

Prove that every BFS for (4.7) is nondegenerate. Show that the dual of (4.7) is

$$\begin{aligned} \text{Maximize} \quad & \sum \ (\pi_i - \pi_1) \\ \text{Subject to} \quad & c_{ij} - \ (\pi_j - \pi_i) \ \geqq 0, \text{ for all } (i,j) \in \mathcal{A} \end{aligned}$$

Show that we can set $\pi_1 = 0$ in the dual problem, and in that case, π_i in an optimum dual solution gives the shortest chain distance from the origin to node i.

Suppose a shortest chain tree $\mathbb{T}_1$ for this problem is given. Now consider the same problem with a different origin node, say node 2. For the flow version of the new problem, $\mathbb{T}_1$ is dual feasible but primal infeasible. Develop a special adaptation of the dual simplex algorithm of LP to compute a shortest chain tree rooted at node 2, starting with $\mathbb{T}_1$ (Florian, Nguyen, and Pallatino [1981]).

4.3 G $= (\mathcal{N}, \mathcal{A}, c)$ is a directed network with c as the arc length vector. Suppose it is required to find a shortest simple chain from 1 to n in G

that has no common nodes with any negative length circuit. Develop a procedure for this problem using any of the shortest chain algorithms discussed in later sections. If there are no common nodes between any chain from 1 to n and any negative length circuit in G, show that this procedure finds a shortest chain from 1 to n in G.

4.4 Suppose there are some negative length circuits in $G = (\mathcal{N}, \mathcal{A}, c)$. Solve (4.1) with the capacity constraint of 1 on all the arcs, let $\bar{f} = (\bar{f}_{ij})$ be an optimum integer flow vector for this problem. If $\bar{f}$ is the incidence vector of a simple chain from 1 to n, prove that it is a shortest simple chain from 1 to n in G.

4.2 Label Setting Methods for Shortest Chains from a Specified Origin

Let $G = (\mathcal{N}, \mathcal{A}, c)$ be the connected directed network with node 1 as the origin for this problem. Here we discuss an algorithm due to E. W. Dijkstra [1959]. The arc length vector c must be $\geqq 0$ for using this algorithm. It builds the shortest chain tree in a connected fashion one arc per step. Once an arc is included in this tree, it is not removed later on. The new node on the arc that is included in each step is the best (i.e., closest to the origin) among all the out-of-tree nodes at that stage. So, the arc selection in each step is a **greedy selection** and the algorithm itself is known as a **greedy algorithm**. We now provide the main result on which the algorithm is based.

THEOREM 4.1 *Let* $G = (\mathcal{N}, \mathcal{A}, c)$ *be a directed connected network with arc length vector* $c \geqq 0$. *Let* $\mathbb{T}$ *be an outtree rooted at the origin, node 1, spanning the nodes in the set* $\mathbf{X} \subset \mathcal{N}$, *which is a shortest chain tree. For each* $i \in \mathbf{X}$, *let* π_i *be the length of a shortest chain from 1 to* i *in* G. *Let* $p \in \mathbf{X}$, $q \in \bar{\mathbf{X}} = \mathcal{N} \setminus \mathbf{X}$ *satisfy:* $(p, q) \in \mathcal{A}$, *and*

$$\pi_p + c_{pq} = \textit{ min. } \{\pi_i + c_{ij} : \textit{ over } i \in \mathbf{X}, j \in \bar{\mathbf{X}}, \ (i, j) \in \mathcal{A}\} \qquad (4.8)$$

Then adding the arc (p, q) *to* $\mathbb{T}$ *and defining* $\pi_q = \pi_p + c_{pq}$, *extends* $\mathbb{T}$ *into* $\mathbb{T}'$ *which is a shortest chain tree spanning the nodes in* $\mathbf{X} \cup \{q\}$.

Proof Let $\mathcal{C}_0$ denote the chain obtained by adding the arc (p, q) at the end of the chain from 1 to p in $\mathbb{T}$. Let $\mathcal{C}$ be any chain from 1 to q. Let (r, s) be the first arc in the cut $(\mathbf{X}, \bar{\mathbf{X}})$ as you travel along $\mathcal{C}$. Let $\mathcal{C}_1$ be the portion of $\mathcal{C}$ from 1 to r, and $\mathcal{C}_2$ the portion from s to q. Then $\mathcal{C}$ consists of the chains $\mathcal{C}_1$ and $\mathcal{C}_2$ and the arc (r, s). So, the length of $\mathcal{C} = c_{rs}$ + length of $\mathcal{C}_1$ + length of $\mathcal{C}_2 \geqq c_{rs}$+ length of $\mathcal{C}_1$ (since length of $\mathcal{C}_2 \geqq 0$ as $c \geqq 0$) $\geqq c_{rs} + \pi_r$ (since π_r is the shortest chain length from 1 to r) $\geqq \pi_p + c_{pq}$ (by (4.8)) = length of $\mathcal{C}_0$. Hence $\mathcal{C}_0$ is a shortest chain from 1 to q. ∎

Beginning with the trivial tree consisting of the single node 1, the result in Theorem 4. 1 can be used repeatedly to obtain in $(n - 1)$ steps the shortest chain tree in G, adding one arc and node per step. At the stage when the tree spans r nodes, selecting the next arc to add to the tree using (4.8) can take up to $O(r(n - r))$ additions and comparisons, since there may be that many arcs in the cut at that stage. If carried out directly, the computational effort in the entire algorithm may therefore be $\sum_r O(r(n - r)) = O(n^3)$. Dijkstra observed that repeated minimization over the cut leads to repeated examination of arcs, and pointed out that cut examination can be replaced by setting and updating node labels called **temporary labels** on out-of-tree nodes. By this, each arc is examined precisely once in the algorithm, and the overall computational effort will be at most $O(n^2)$.

Nodes may be in three possible states: **permanently labeled, temporary labeled,** or **unlabeled** in this algorithm. The label in node i is always of the form $(P(i), d_i)$, where $P(i)$ is the predecessor index of i, and d_i is the length of the present chain to i, which is the predecessor path of i in reverse order. A node becomes permanently labeled when it is included in the shortest chain tree, and then its label will not change subsequently. Hence, this method and all variants of it are called *label setting methods*. For each permanently labeled node, the present chain from 1 to it will be a shortest chain. **X, Y, N** denote the sets of permanently labeled, temporary labeled, unlabeled nodes respectively. In each step, one node is transferred from **Y** to **X**. The

labels on nodes in $\mathbf{Y}$ are updated in each step.

THE LABEL SETTING METHOD

Step 1 Label the origin, 1, with the permanent label ($\emptyset$, 0). Label each $j \in \mathbf{A}_1$ with the temporary label (1, c_{1j}). $\mathbf{X} = \{1\}$, $\mathbf{Y} = \mathbf{A}_1$, $\mathbf{N} = \mathcal{N} \backslash (\mathbf{X} \cup \mathbf{Y})$. Go to Step 2.

Step 2 If $\mathbf{Y} = \emptyset$ go to Step 3. If $\mathbf{Y} \neq \emptyset$, find an $i \in \mathbf{Y}$ for which the distance index d_i in the label is the least among all the nodes in $\mathbf{Y}$ at this stage. Break ties for this i arbitrarily. Make the current label on i permanent and move it from $\mathbf{Y}$ to $\mathbf{X}$. If the label on i is (P(i), π_i), (P(i), i) is the arc included in the shortest chain tree in this step.

For each $j \in \mathbf{Y}$, let d_j be its present distance index. If $(i, j) \in \mathcal{A}$ and $d_j > \pi_i + c_{ij}$, change the label on j to $(i, \pi_i + c_{ij})$, otherwise leave the temporary label on j unchanged.

Temporary label each $j \in \mathbf{N} \cap \mathbf{A}_i$ with $(i, \pi_i + c_{ij})$ and move it from $\mathbf{N}$ to $\mathbf{Y}$.

If $\mathbf{X} \neq \mathcal{N}$ repeat this Step 2.

Step 3 The present labels define a shortest chain tree spanning the nodes in $\mathbf{X}$. Since $\mathbf{Y} = \emptyset$, if $\mathbf{X} \neq \mathcal{N}$, there exists no chain from 1 to any node in the set $\mathbf{N}$ at this stage. Terminate.

Discussion

As an example consider the network G in on the left of Figure 4.4 with arc lengths entered on the arcs. Figures 4.4 to 4.6 illustrate the various steps in the label setting method to find a shortest chain tree rooted at node 1 in this network. Permanent labels are marked in bold face, temporary labels in regular style. In-tree arcs are marked with thick lines. Figure 4.6 right side network contains the shortest chain tree rooted at node 1.

Suppose all the c_{ij} are equal to 1. In this case the shortest chain problem is that of finding a chain from the origin to the destination consisting of the smallest number of arcs. In the shortest augmenting

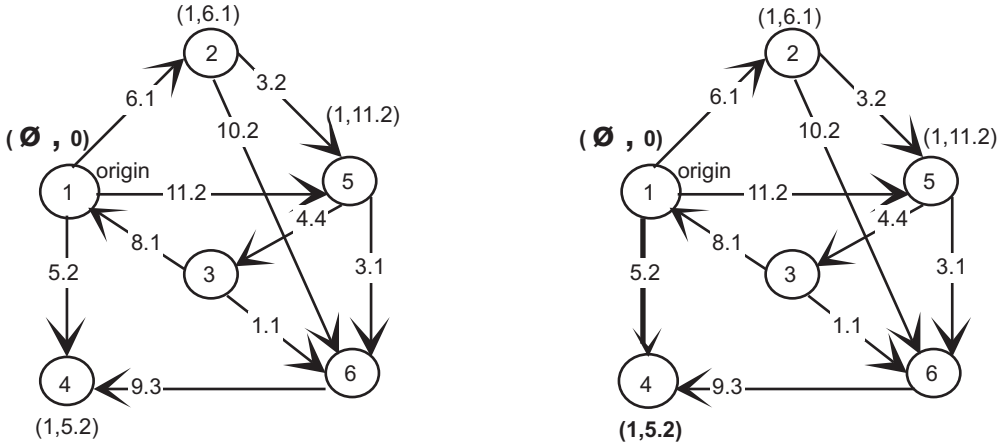

Figure 4.4:

path method of Section 2.3.3 for the maximum value flow problem, in each step the problem of finding a shortest augmenting path is exactly a problem of this type in the residual network at that stage. It can be verified that the breadth first search method for it discussed in Section 2.3.3 is a specialization of this label setting method to it.

THEOREM 4.2 *Assuming that* $c \geqq 0$, *the in-tree chain from 1 to any permanently labeled node is a shortest chain.*

Proof Assume that the following statements hold at some stage.

(i) For each $j \in \mathbf{X}$, the present in-tree chain to j is a shortest chain.

(ii) For each $j \in \mathbf{Y}$, the present chain from 1 to j traced by the current labels (permanent or temporary) traverses only through nodes in $\mathbf{X}$ before reaching j, and has minimum length among all chains satisfying this property.

Let $i \in \mathbf{Y}$ be the temporarily labeled node whose label is made permanent next. For $j \in \mathbf{Y}$ let d_j be the present distance label on j. Then by the choice of i

$$d_i = \text{ min. } \{d_j : j \in \mathbf{Y}\} \tag{4.9}$$

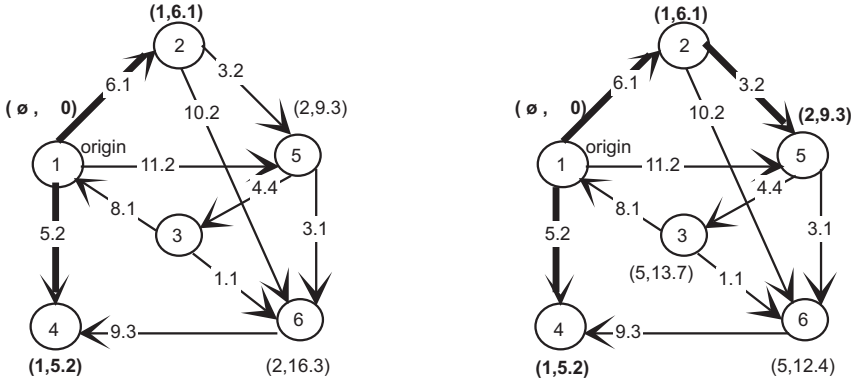

Figure 4.5:

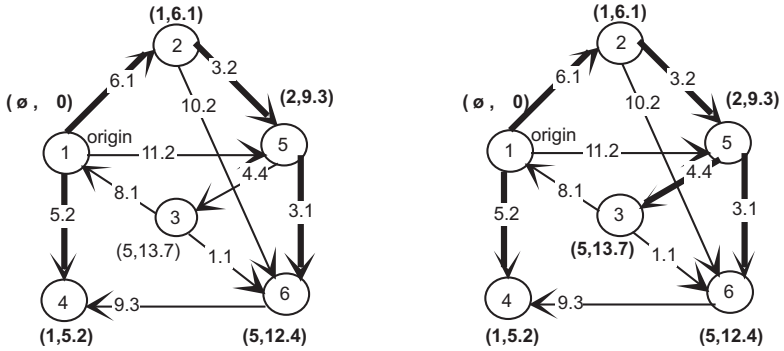

Figure 4.6:

Suppose the present chain to i is not a shortest. Let $\mathcal{C}$ be a shortest chain from 1 to i. Then length of $\mathcal{C}$ must be $< d_i$, and by (i) it must pass through some nodes $j \in \mathbf{Y}$, $j \neq i$ before reaching i, suppose $q \in \mathbf{Y}$ is the first such node on $\mathcal{C}$. Let δ be the length of the portion of $\mathcal{C}$ from q to i, $\delta \geqq 0$ since $c \geqq 0$. The portion of $\mathcal{C}$ from 1 to q is a shortest chain to q, and all nodes on it other than q are from $\mathbf{X}$, so, by (ii) its length is d_q. So, length of $\mathcal{C} = d_q + \delta \geqq d_q \geqq d_i$ by (4.9). Since the length of $\mathcal{C} < d_i$ by the hypothesis, this is a contradiction. So, the present chain to i must be a shortest chain, thus (i) continues to hold after transferring i from $\mathbf{Y}$ to $\mathbf{X}$. By the updating process of temporary labels, (ii) also continues to hold after this transfer.

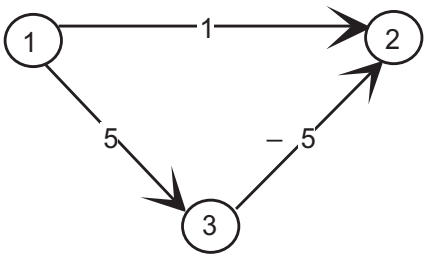

Figure 4.7:

Clearly (i), (ii) hold initially. By the above arguments and induction, (i), (ii) hold after each step, and the assertion in the theorem remains valid. ∎

THEOREM 4.3 *The computational effort in this method is bounded above by* $O(n^2)$ *where* $n = |\mathcal{N}|$.

Proof Each time Step 2 is executed, the choice of node i requires at most $(n-2)$ comparisons, and the updating of temporary labels requires an additional $(n-2)$ additions and comparisons at most. Step 2 is executed at most $(n-1)$ times during the algorithm. Hence the result follows. ∎

The label setting method does not work unless $c \geqq 0$. One can verify this by applying it on the network in Figure 4.7.

A major computational burden in this method is the repeated search among temporary labels to find the minimum. Special data structures and sorting techniques have been developed to make this search efficient; see Denardo and Fox [1979], Dial [1965], Dial, Glover, Karney, and Klingman [1979], Fredman and Tarjan [1987], Imai and Iri [1984], and Johnson [1973, 1977].

Once a point is permanently labeled in this method, the shortest chain from the origin to that point is known. Thus, if shortest chains from the origin to only a specified subset of points are desired, the method can be terminated as soon as all those points are permanently labeled. This is an important advantage of label setting methods. The

label correcting methods discussed in the next section do not have this advantage; there the full shortest chain tree has to be found before any shortest chain is known with certainty.

4.3 Label Correcting Methods for Shortest Chains from a Specified Origin

Let $G = (\mathcal{N}, \mathcal{A}, c)$ be the connected directed network with node 1 as the origin for this problem. Here we discuss variants of the primal simplex method applied to the LP formulation (4.1). Every basis for (4.1) corresponds to a spanning tree in G, and a pivot step in it corresponds to exchanging an out-of-tree arc with an in-tree arc in its fundamental cycle. These methods always maintain a spanning outtree rooted at the origin, and change it in each step by changing the labels on one or more nodes. Hence these methods are known as **label correcting methods**.

For each $j \in \mathcal{N}$ such that $(1, j) \notin \mathcal{A}$, these methods usually introduce an artificial arc $(1, j)$ associated with a large length α. Taking $\alpha = 1 + n(\max.\{|c_{pq}| : (p, q) \in \mathcal{A}\})$ is sufficient. After this change, let $\mathbb{T}_0$ be the spanning outtree rooted at 1 consisting of the arcs $(1, j)$ for each $j \neq 1$. $\mathbb{T}_0$ is normally used as the initial outtree in most of these methods. If an artificial arc $(1, j)$ is contained in the shortest chain tree obtained at the termination of the method, it implies that there exists no chain from 1 to that particular node j in the original network. Thus, eliminating all the artificial arcs from the final outtree obtained in the method leaves a shortest chain tree spanning all the nodes that can be reached from 1 by a chain in the original network.

These methods work for general c, so we do not require $c \geqq 0$ in this section. Each of them will terminate after a finite number of steps with either a negative length circuit, or a shortest chain tree. These methods maintain node labels of the form $(P(i), d_i)$ on node i, for each $i \in \mathcal{N}$, where $P(i)$ is the predecessor index of i. At any stage, we denote the set of arcs $\{(P(i), i): i \neq 1\}$ by the symbol $\mathbf{E}$. As long as the node labels represent a tree, $\mathbf{E}$ is the set of arcs in the outtree at that stage; in this case for each $i \in \mathcal{N}$, d_i in the label on i is either the length

of the present chain from 1 to i, or a number $\geqq$ that length. When a negative length circuit is obtained in the method, the node labels may no longer represent a spanning outtree; at that time $\mathbf{E}$ will form two or more connected components, an outtree (not spanning) rooted at 1, and one or more disjoint negative length circuits. We will first discuss the classical primal method for the shortest chain problem to provide the basic ideas behind this class of algorithms.

THE CLASSICAL PRIMAL METHOD FOR THE SHORTEST CHAIN PROBLEM

In this method node labels always define a spanning outtree and they are of the form $(\mathrm{P}(i),\ \pi_i)$ where π_i is always the length of the present chain to i. Artificial arcs are eliminated from the network once they leave the outtree.

Initialization Begin with the spanning outtree $\mathbb{T}_0$ discussed above, and the node labels that go with it.

General step For each $i \in \mathcal{N}$ let $(\mathrm{P}(i),\ \pi_i)$ be the current label on it. Look for an arc $(i, j) \in \mathcal{A}$ violating the dual constraint corresponding to it in (4.2), i.e., satisfying $\pi_j > \pi_i + c_{ij}$. If there is no such arc, the present outtree is a shortest chain tree; terminate. Otherwise, with this arc (i, j) do the following. Let $\delta = \pi_j - \pi_i - c_{ij}$. So, $\delta > 0$.

Ancestor checking operation Check whether j is an ancestor of i in the present tree. If it is, the circuit consisting of the arc (i, j) and the portion of the predecessor path of i between j and i is a negative length circuit of length -δ; terminate. Otherwise continue.

Label correction Replace the in-tree arc incident into node j by (i, j); i.e., change the label on j to $(i, \pi + c_{ij})$. This has the effect of reducing the length of the chain to j by δ.

Correcting the distance index of descendents For each descendent t of j leave its predecessor index unchanged, but change its distance from the present π_t to $\pi_t - \delta$.

Go to the next step.

Discussion

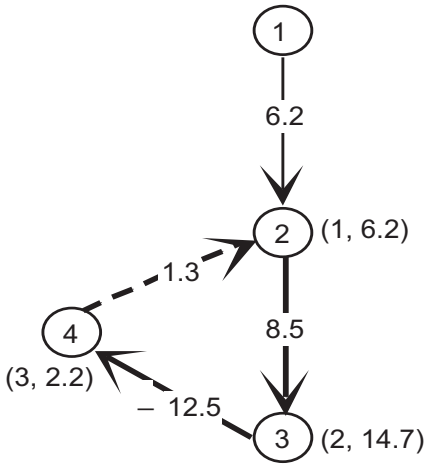

Figure 4.8:

As an example consider the network in Figure 4.8. Numbers on the arcs are their lengths, and node labels are given by the side of nodes. Arc (4, 2) violates dual feasibility as $\pi_2 = 6.2 > \pi_4 + c_{42} = 2.2 + 1.3 = 3.5$. Here $\delta = 2.7$. Node 2 is on the predecessor path of node 4; this identifies the thick negative length circuit of length -2.7, and the method terminates.

As another example consider the network on the left of Figure 4.4 with arc lengths entered on the arcs. In Figure 4.9, on the left, we introduce the dashed artificial arcs (1, 3), (1, 6) with length ∞. The various outtrees obtained in applying this method on this network are given in Figures 4.9 to 4.10. Outtrees are marked by thick lines and in each step the node labels are entered by the side of the nodes.

Each outtree corresponds to a unique π-vector, and in each step the π_i for at least one node i strictly decreases. So, no outtree can reappear, and the method
must terminate after at most a finite number of steps. If a negative length circuit is not discovered, at termination we have an outtree in

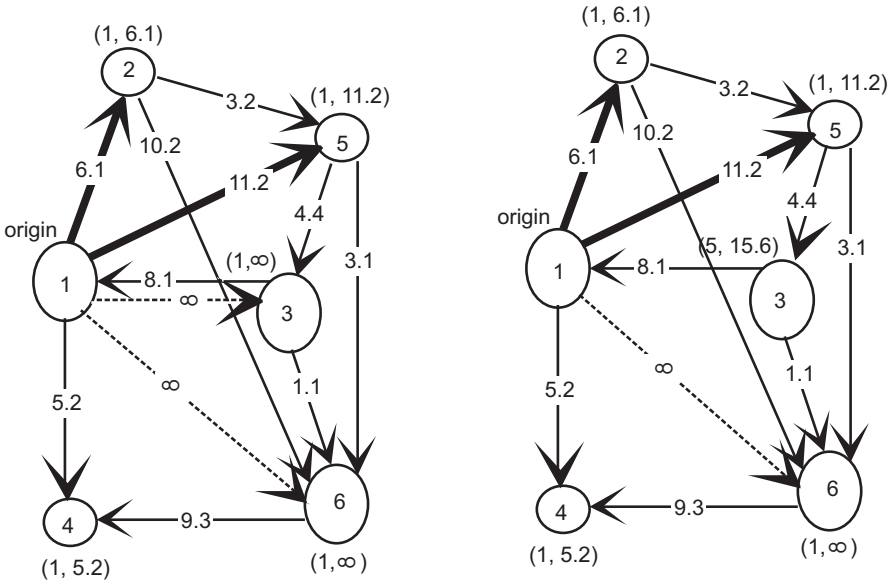

Figure 4.9: Node 1 is the origin.

which the optimality conditions discussed earlier hold; hence it is a shortest chain tree.

The computational complexity of this method depends critically on the procedure used to look for an arc (i, j) violating dual feasibility. Under some rules for this search, the computational requirements in the method grow exponentially with the size of the problem in the worst case. There are some search procedures under which the method turns out to be nicely polynomially bounded. We will discuss some of these efficient implementations. Some of these implementations select a node i first by some rule, and then they examine successively all arcs in the forward star (or reverse star) of this node for dual feasibility. This operation is called **branching out** of node i. Candidate nodes for branching out are maintained in a list.

The ancestor checking operation, and correcting the distance index of descendents, each adds an $O(n)$ effort per step to the worst case computational complexity. For correcting the distance indices of descendents efficiently, the predecessor indices alone are not adequate; we need to use some other list structures discussed in Chapter 1 to repre-

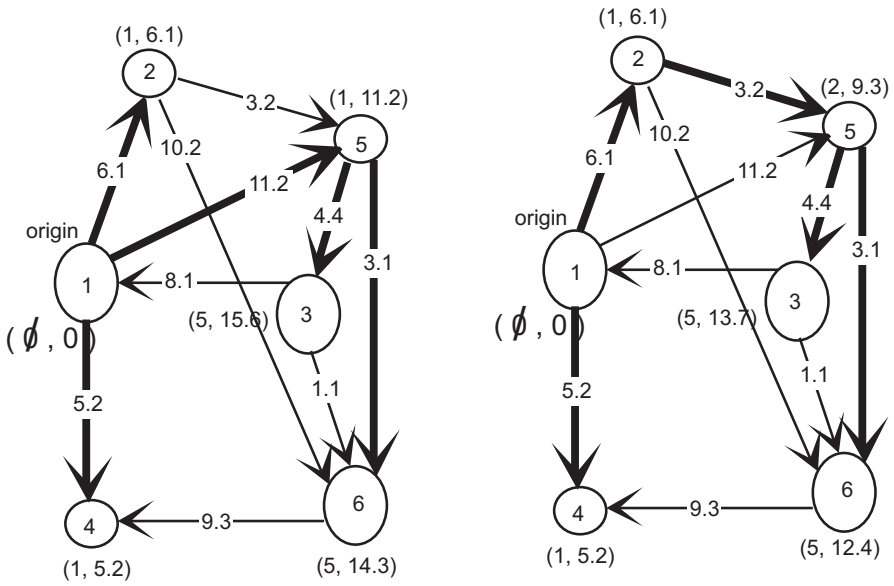

Figure 4.10:

sent the tree and we need to update them in every step. This makes each step expensive. That's why in the implementations discussed below, these operations are not carried out.

When arc (i, j) violates dual feasibility, and we do not carry out the ancestor checking operation, j may be an ancestor of i and we may not detect it. If j is an ancestor of i, and we carry out the label correction on node j, the node labels will no longer represent a spanning tree; from then on the set $\mathbf{E}$ will consist of a negative length circuit containing node j. These implementations have special rules to detect the occurrence of this possibility.

If the operation of correcting the distance index of descendents is not carried out in each step, in the label $(\mathrm{P}(i), d_i)$ on node i, d_i may not be equal to the length of the present chain to node i; it will be a number $\geq$ that length. However this will be detected when all the arcs on this chain are examined, and the distance indices will get corrected eventually, and if a negative length circuit is not identified, the

distance indices on the nodes become the actual lengths of the chains at termination.

Elimination of the ancestor checking and correcting the distance labels of descendents operations in each step makes it possible for the following implementations to have a better worst case computational complexity bound. Clearly, the ancestor checking operation should be eliminated if it is known that no negative length circuits exist in G. Otherwise, computational tests reveal that the extra work in carrying out these operations is worthwhile as it avoids scanning nodes just for correcting their labels, and definitely pays off in better average performance, particularly as the density of the network increases.

The Bellman-Ford-Moore Label Correcting Algorithm

We will now discuss an implementation known as the **Bellman-Ford-Moore label correcting algorithm** or the **BFM method** which can be interpreted as an iterative approach for solving the Bellman-Ford equations (4.6). It goes through several iterations; the distance indices in the rth iteration are denoted by $\pi_i^r, i \in \mathcal{N}, r = 1, 2, \ldots$. In each iteration it corrects the distance labels on all the nodes using recursive equations of the form (4.10) given below. So, the method is one of successive approximations for (4.6), obtaining the $(r+1)$th order approximation from the rth.

THE BFM METHOD

Initialization Begin with the spanning outtree $\mathbb{T}_0$ discussed above, and the node labels that go with it. Set iteration count equal to 1.

General iteration $r+1$ for $r \geq 1$ Let $(\mathrm{P}(i),\ \pi_i^r)$ be the label on node $i \in \mathcal{N}$ at the end of the previous step. For each $j \in \mathcal{N}$ compute

$$\pi_j^{r+1} = \text{ min. } \{\pi_j^r; \pi_i^r + c_{ij} : \text{ over } i \in \mathbf{B}_j\} \tag{4.10}$$

If $\pi_j^{r+1} = \pi_j^r$ for all $j \in \mathcal{N}$; terminate, the present labels define a shortest chain tree rooted at node 1. If this condition does not hold, for each j such that $\pi_j^{r+1} < \pi_j^r$ do the following: Let $i = u$ be an index attaining the minimum in (4.10). Change the label on j to (u, π_j^{r+1}). If $r + 1 \leqq n - 1$ go to the next iteration. Otherwise (i.e., if $r + 1 = n$) a negative length circuit has been detected; identify it in the present set $\mathbf{E}$ of arcs, and terminate.

Discussion

For each $j \in \mathcal{N}$ and $r \geqq 0$, π_j^r here represents the length of a shortest chain from 1 to j among those that contain no more than r arcs. This interpretation is clearly valid for $r = 1$. Set up an induction hypothesis that this statement is true for some r. Now we will show that this implies that it must be true for $r + 1$ too. A shortest chain from 1 to j of no more than $r+1$ arcs either contains $\leqq r$ arcs (in which case its length is π_j^r by the induction hypothesis) or contains $r+1$ arcs and has some final arc incident into j, say (u, j). The portion of this chain from 1 to u must be a shortest chain to u containing r arcs, so the length of this chain to j is $\pi_u^r + c_{uj}$. Hence minimizing $\pi_u^r + c_{uj}$ over $u \in \mathbf{B}_j$, which gives π_j^{r+1}, yields the length of the shortest chain to j with $r + 1$ or less arcs in the latter case. So, the statement must hold for $r + 1$ too, and by induction it holds for all r.

If there is no negative length circuit in G, with the introduction of the artificial arcs in Step 1, there exists an unconstrained shortest chain to j with $\leqq (n - 1)$ arcs for each $j \in \mathcal{N}$, and hence π_j^r will stabilize at the length of the unconstrained shortest chain to j in $\leqq (n - 1)$ iterations. Such stability (i.e., $\pi_j^{r+1} = \pi_j^r$ for each $j \in \mathcal{N}$) will not be reached after n iterations only if there is a negative length circuit in G. If stability is attained, we then have an outtree in which the optimality conditions discussed earlier hold, and hence it is a shortest chain tree.

In each iteration all the nodes are examined. Examining node j in an iteration requires $O(|\mathbf{B}_j|)$ additions and comparisons. So, the computational effort in an iteration is $O(m)$, and since there are at most n iterations in the method, the overall computational effort is at most $O(nm)$.

As mentioned above, if the BFM method goes through n iterations without stabilizing, it is an indication that a negative length circuit has been detected. There are other conditions signaling the detection of a negative length circuit in this method. One is that $\pi_1^r < 0$ for any r (1 is the origin node). This signifies the detection of a negative length circuit containing node 1. Another condition is that $\pi_j^{r+1} < \pi_j^r$ holds for at least $(n-r)$ distinct nodes j, for some $r = 1$ to $n-1$; see Exercise 4.6. When any of these conditions hold, the negative length circuit itself can be located among the set of arcs **E** at that stage, and the method terminated.

The Dynamic Breadth-First Search Label Correcting Algorithm

Let the **present label depth of node** j, denoted by a_j, be the number of arcs on the present chain to j. Node labels on node j in this algorithm are of the form $(\mathrm{P}(j),\ d_j,\ a'_j)$, where $\mathrm{P}(j)$, d_j have the same meaning as before, and a'_j is the **label depth index of node** j; it may not be equal to the label depth of j in intermediate stages since the correction operation on descendents is not carried out, but it will be corrected and will become equal to the true label depth on all the nodes, at termination. $a'_j \leqq a_j$ for all j always, as the label depth index of a node can only increase in this algorithm. Like breadth-first search, this variant performs a sequence of iterations in which in iteration r, only those nodes with label depth index r are branched out. Since the label depth index of a node can change during the algorithm, this variant can be viewed as a dynamic breadth-first search algorithm.

At any stage, for each h, $n(h)$ denotes the number of nodes j for which $a'_j = h$. In each iteration of this algorithm, nodes from a specified subset called **list** are selected according to a criterion, for branching out. The iteration is completed when all the eligible nodes in the list are branched out. While this is going on, a new subset of nodes called **next list** is being built. The next list at the end of an iteration becomes the list for the next iteration.

THE DBFS LABEL CORRECTING ALGORITHM

Initialization Begin with the spanning outtree $\mathbb{T}_0$ discussed above, and the node labels that go with it. $a'_1 = 0$ and $a'_j = 1$ for all $j \neq 1$. List $= \mathcal{N}\backslash\{1\}$. Next list $= \emptyset$. $n(0) = 1, n(1) = n - 1$ and $n(h) = 0$ for all $h > 1$. Set iteration count equal to 1. Go to iteration 1.

General iteration r: 1. Select a node to branch out If list $= \emptyset$ go to Step 3 given below. Otherwise select a node i from the list to branch out, and delete it from the list. Let the present label on i be $(\mathrm{P}(i),\ d_i,\ a'_i)$. If $a'_i = r$ go to Step 2. Otherwise repeat this step.

2. Branching out of selected node i Do the following for each $j \in \mathbf{A}_i$. Let the present label on j be $(\mathrm{P}(j),\ d_j,\ a'_j)$. If $d_j \overset{\leq}{=} d_i + c_{ij}$ continue. If $d_j > d_i + c_{ij}$, change the predecessor index of j to i, its distance label to $d_i + c_{ij}$, and if $a'_j \neq r+1$ subtract 1 from $n(a'_j)$.

If $n(a'_j) = 0$, the network contains a negative length circuit. One such circuit can be identified by beginning at j and tracing a path using the predecessor labels until a node repeats.

Otherwise change a'_j to $r+1$, add 1 to $n(r+1)$, and include j in next list.

After all this work is completed return to Step 1.

3. Set up candidate set for next iteration If next list $= \emptyset$, the present labels define a shortest chain tree rooted at the origin, node 1; terminate. Otherwise make next list into list, next list $= \emptyset$, and go to the next iteration.

Discussion

First assume that there is no negative length circuit in G. For each $j \in \mathcal{N}$ let π^*_j denote the unknown length of a shortest chain from 1 to j, and b_j the smallest number of arcs in a shortest chain from 1 to j. Define $\mathbf{L}(r) = \{\ j : b_j = r\}$.

Observe that at the start of iteration r, the list consists entirely of those nodes i for which $a'_i = r$, since in the previous step these and only

these nodes are put in the next list. Also, a node in the list is branched out in this iteration only if its label depth index remains equal to r when the algorithm tries to select that node.

(i) At the start of iteration r in this algorithm $\mathbf{L}(r)$ is a subset of the list, and $d_i = \pi_i^*$ for all $i \in \mathbf{L}(r)$. Also, in this iteration, all nodes in $\mathbf{L}(r)$ are branched out.

We will now establish by induction that these statements are correct. From the initialization step, they clearly hold for $r = 1$. Set up an induction hypothesis that they hold for $r = \bar{r}$. We now show that the induction hypothesis implies that they must also hold for $r = \bar{r}+1$. Let $i \in \mathcal{N}$ be such that $b_i = \bar{r} + 1$. Then there is a shortest chain from 1 to i that has an arc, (u, i) say, as the last arc, with $b_u = \bar{r}$. By the induction hypothesis, u is in the list at the start of iteration $\bar{r}$, and u is branched out in this iteration. There may be several shortest chains from 1 to i with $\bar{r} + 1$ arcs; let u denote the first node on such a chain that is branched out by the algorithm in iteration $\bar{r}$. So, before u is branched out in this iteration we have $d_i > \pi_i^*$, and d_i will be set $= \pi_i^*$ and a_i' to $\bar{r} + 1$ when u is branched out, and i included in the next list. At the end of this iteration $\bar{r}$, the next list is made into the list for iteration $\bar{r}+1$; hence i is in it, and $d_i = \pi_i^*, a_i' = \bar{r}+1$ at that time. Also, since $d_i = \pi_i^*$, a_i' cannot increase during this iteration, it remains $= \bar{r} + 1$ and node i will be branched out. So, the above statements hold for $r = \bar{r} + 1$. Hence by induction they hold for all r.

(ii) From (i) it follows that if there are no negative length circuits in G, this algorithm terminates with a shortest chain tree rooted at node 1 after at most $(n-2)$ iterations. Each iteration involves at most $O(m)$ effort, so the overall computational complexity of this algorithm is at most $O(nm)$.

(iii) Now consider the case where there may be negative length circuits in G. If $n(h) = 0$ for some $h = 1$ to $r-1$ during iteration r, then the network contains a negative length circuit. This is a consequence of the fact that $d_i = \pi_i^*$ for all $i \in \mathbf{L}(r)$ at the start of iteration r and thereafter, if there are no negative length circuits in G.

So, this algorithm either finds a shortest chain tree rooted at the origin, or a negative length circuit, after at most n iterations. The worst case computational complexity of the algorithm is $O(nm)$.

This algorithm, due to Goldfarb, Hao, and Kai [1991] is based on earlier work of Glover, Klingman and Philips [1985]. Computational results show that its performance is very good. On randomly generated networks with 100 to 5000 nodes, and 10,000 to 60,000 arcs, it required between 0.13 to 3.4 CPU seconds of SUN-3 computer to solve each problem.

Exercises

4.5 Let G $= (\mathcal{N}, \mathcal{A}, c = (c_{ij}))$ be a directed network with c as the vector of arc lengths, and $\mu = (\mu_i)$ a vector of node prices in G. For each $(i, j) \in \mathcal{A}$ let $c'_{ij} = c_{ij} - (\mu_j - \mu_i)$, and $c' = (c'_{ij})$. For any circuit in G, prove that its length with arc length vectors c or c' is the same. Also prove that for any $i, j \in \mathcal{N}$ any shortest chain from i to j with c as the arc length vector is also a shortest chain with c' as the arc length vector, and vice versa.

4.6 In the BFM method prove that if $\pi_j^{r+1} < \pi_j^r$ holds for at least $n - r$ distinct nodes j for some $r = 1$ to $n - 1$, then the method has detected a negative length circuit which can be located among the set of arcs **E** at that stage.

4.7 Find the shortest chains from 1 to all the other nodes in the network in Figure 4.11. Arc lengths are entered on the arcs.

4.8 Let G $= (\mathcal{N}, \mathcal{A}, c)$ be a directed network with $|\mathcal{N}| = n$, $|\mathcal{A}| = m$ and 1 as the origin node. Let $e_1, \ldots, e_m$ be an ordering of the arcs in $\mathcal{A}$. To find the shortest chains from 1 to all the other nodes, the classical primal method is applied using the following selection rule. Arcs are examined in the specific order $e_1, \ldots, e_m$ for dual feasibility. When all the arcs are examined once, a sweep is said to have been completed, and then the method goes to the next sweep. When an entire sweep produces no improvement in the distance index of any node (i.e., when every arc satisfies dual feasibility) the method terminates.

If there are no negative length circuits in G, and r is the smallest number of arcs in a shortest chain from 1 to a node j, prove that the

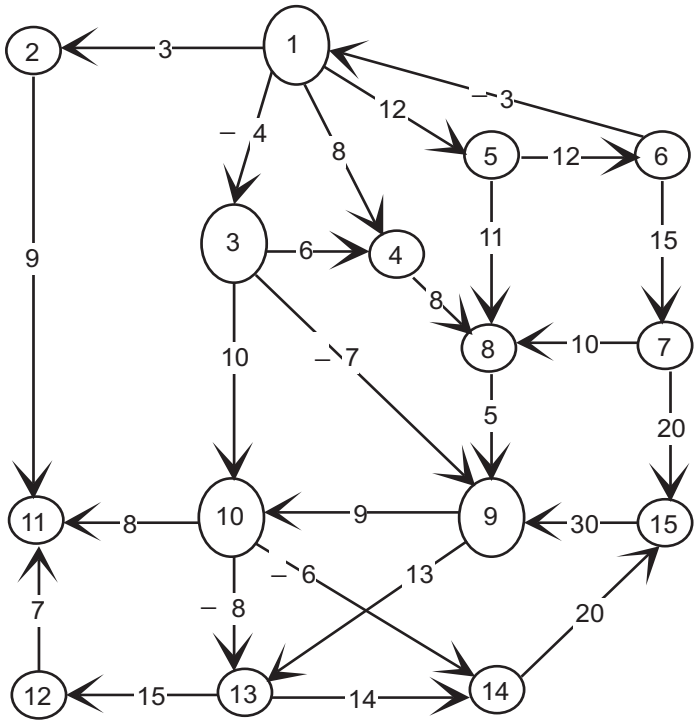

Figure 4.11:

present chain in the method becomes a shortest chain to j by the end of the rth sweep (i.e., the label on j will not change after the rth sweep). Using this, develop a version whose worst case computational complexity is $O(nm)$.

4.9 Consider the following implementations of the primal method for finding a shortest chain tree in $G = (\mathcal{N}, \mathcal{A}, c)$ rooted at node 1. Ancestor checking and correcting the distance index on descendents operations are not carried out in each step. These implementations go through several iterations. In each iteration they select nodes for branching out always from the top of a sequence $\mathbf{S}_1$ of nodes, in which nodes are arranged from top to bottom. As the iteration is progressing, all the nodes whose labels have changed are arranged in another

sequence $\mathbf{S}_2$ according to one of the following disciplines. These implementations differ in the way nodes j whose labels have been revised during the iteration are entered in the sequence list $\mathbf{S}_2$.

FIFO/NO MOVE If j is not in $\mathbf{S}_2$, it is inserted at the bottom of $\mathbf{S}_2$. If j is already in $\mathbf{S}_2$, it is left in its current position.

FIFO/MOVE If j is not in $\mathbf{S}_2$, it is inserted at the bottom of $\mathbf{S}_2$. If j is already in $\mathbf{S}_2$, it is moved from its present position to the bottom of $\mathbf{S}_2$.

Both these implementations begin iteration 1 with $\mathbf{S}_1 = \{1\}$ after initialization. If there are no negative length circuits in G, prove that these implementations terminate with a shortest chain tree after at most n iterations, with the sequence $\mathbf{S}_2$ becoming $\emptyset$ at the end of the last iteration. Using this result show that these implementations can be operated so that their worst case computational complexity to find either a negative length circuit or a shortest chain tree is $O(nm)$ (Shier and Witzgall [1981]).

4.10 Consider the following implementations of the primal method for finding a shortest chain tree in $G = (\mathcal{N}, \mathcal{A}, c)$ rooted at node 1. Correcting the distance index on descendents operation is not carried out in each step. The implementations select nodes for branching out always from the top of a sequence of nodes in which nodes are arranged from top to bottom. As the algorithm is progressing, each node j whose label has changed is added to the sequence according to one of the following disciplines.

LIFO/NO MOVE If j is not in the sequence at this time, it is added at the top. If it is already in the sequence it is left in its current position.

LIFO/MOVE If j is not in the sequence at this time, it is added at the top. If it is already in the sequence, it is moved from its position to the top.

Both these implementations begin iteration 1 with $\mathbf{S}_1 = \{1\}$ after initialization. Even when there is no negative length circuit in G, show that the number of nodes branched out before termination in these

implementations can grow exponentially with n in the worst case (Shier and Witzgall [1981]).

4.4 Shortest Chains From a Specified Origin in an Acyclic Network

Let G $= (\mathcal{N}, \mathcal{A}, c = (c_{ij}))$ be an acyclic network with c as the vector of arc lengths, $|\mathcal{N}| = n$, and $|\mathcal{A}| = m$. We assume that the nodes in G are numbered serially using an acyclic numbering, i.e., $i < j$ for each $(i, j) \in \mathcal{A}$. If it is required to find the shortest chains from an origin, node p, since there are no arcs (j, p) for $j < p$, there exist no chains from such nodes j to p. Hence all nodes $j < p$ and all arcs incident to them can be eliminated. The resulting network is again acyclic, and the nodes can be renumbered in it by subtracting $p-1$ from the present number. This transforms the problem into one in an acyclic network with node 1 as the origin. Hence in the sequel we assume that node 1 is the origin.

Since G contains no circuits, there are no negative length circuits even if $c < 0$, and the following special algorithm always finds the shortest chains in it. The algorithm is a recursive procedure that comes from the application of the principle of optimality of dynamic programming to the problem. Nodes are labeled in it in the order 1 to n at the rate of one per step, and all node labels assigned are permanent labels.

ACYCLIC SHORTEST CHAIN ALGORITHM

Step 1 Label the origin, node 1, with $(\emptyset, 0)$. Go to Step 2.

General step r, for $r \geq 2$ At this stage all nodes $1, \ldots, r-1$ would have been labeled already. Let the distance index on node i be $\pi_i, i = 1$ to $r - 1$. Find

$$\pi_r = \text{ min. } \{\pi_i + c_{ir} : i \in \mathbf{B}_r\} \tag{4.11}$$

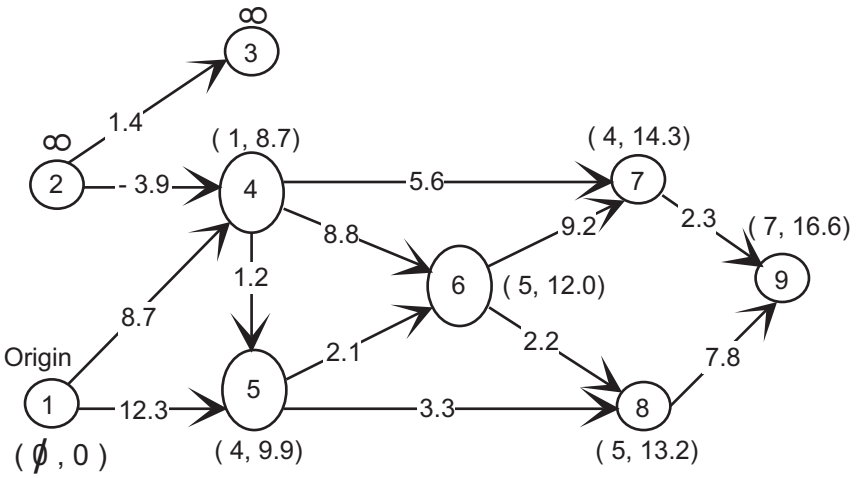

Figure 4.12:

Use the convention that the minimum in the empty set is ∞. Let $\mathrm{P}(r)$ be equal to one of the i that attain the minimum in (4.11) if π_r is finite. Label node r with $(\mathrm{P}(r),\, \pi_r)$. If $r = n$, terminate. Otherwise go to the next step.

Discussion

Since $i < j$ for all $(i, j) \in \mathcal{A}$, (4.11) guarantees that the following conditions, which are the optimality conditions established earlier for the present outtree to be a shortest chain tree, will hold.

$$\pi_j - \pi_i \begin{cases} \leqq c_{ij} & \text{for all } (i,j) \in \mathcal{A} \\ = c_{ij} & \text{if } i = \mathrm{P}(j) \end{cases}$$

So, the final outtree is a shortest chain tree rooted at 1. For $i \in \mathcal{N}$ if $\pi_i = \infty$ there exists no chain from 1 to i in G; otherwise π is the length of a shortest chain to i. The computational effort in Step r of the algorithm is $|\mathbf{B}_r|$ additions and comparisons, so the overall computational complexity of this algorithm is $\mathrm{O}(m)$.

As an example, consider the acyclic network in Figure 4.12, with arc lengths entered on the arcs. Nodes are given an acyclic numbering. Node labels obtained in the algorithm are entered by the side of the nodes in Figure 4.12. This algorithm finds an important application in critical path methods on project networks discussed in Chapter 7.

4.5 Matrix Methods for Shortest Chains Between All Pairs of Nodes

Here we consider the problem of computing shortest chains between every pair of points in the directed network $G = (\mathcal{N}, \mathcal{A}, c)$. This can be solved by applying the algorithms of Section 4.3 with each node as the origin separately, but the algorithms discussed in this section will find all these shortest chains simultaneously.

All the shortest chains and their lengths can be stored very conveniently by maintaining two square matrices, a **label matrix** and a **distance matrix** . Both the rows and columns of each matrix are associated with nodes in $\mathcal{N}$. The entry (called the **label**) in the label matrix in the row of node i and the column of node j, is the predecessor of j in the chain from i to j. So, if this entry is p, (p, j) is the last arc in the present chain from i to j. The remaining arcs on this chain can be traced by looking up the the label in the row of i and the column of p, and continuing until node i is reached. The entry in the distance matrix in the row of i and the column of j is the length of the present chain from i to j.

For $i, j \in \mathcal{N}$, if $(i, j) \notin \mathcal{A}$, introduce an artificial arc (i, j) with length equal to ∞, or a large positive number as discussed before. The methods discussed in this section work with the distance matrix using matrix theoretic operations. Hence, these methods are classified as **matrix methods**, and they terminate either by finding a negative length circuit, or by finding all shortest chains. In the latter case if the shortest chain obtained from a node p to a node q contains an artificial arc, it implies that there is no chain from p to q in the original network.

An Inductive Algorithm for Finding All Shortest Chains

This algorithm proceeds inductively on the number of nodes in the network. It takes exactly n steps. In the rth step, it obtains all the shortest chains in the partial network induced by the subset of nodes $\{1, \ldots, r\}$. In the $(r+1)$th step it brings node $r+1$ into the set of included nodes. This algorithm is due to Dantzig [1967].

INDUCTIVE ALGORITHM

Step 1 Begin with the partial network of node 1 alone. The initial matrices are

Label Matrix

	to 1
from 1	1

Distance Matrix

	to 1
from 1	0

General step $r+1$ **for** $r \geq 1$ Let $(L^r_{ij} : i, j = 1 \text{ to } r)$, (d^r_{ij}) be the label and distance matrices at the end of Step r. L^r_{ii} will be i and d^r_{ii} will be 0 for all $i = 1$ to r. Compute

$$d^{r+1}_{i,r+1} = \text{min.}\{c_{i,r+1}; d^r_{ij} + c_{j,r+1}, j = 1 \text{ to } r, j \neq i\} \quad (4.12)$$

$$d^{r+1}_{r+1,i} = \text{min.}\{c_{r+1,i}; c_{r+1,j} + d^r_{ji}, j = 1 \text{ to } r, j \neq i\} \quad (4.13)$$

$$d^{r+1}_{r+1,r+1} = \text{min.}\{0; d^{r+1}_{r+1,j} + d^{r+1}_{j,r+1}, j = 1 \text{ to } r\} \quad (4.14)$$

For $i = 1$ to r, define

$$L^{r+1}_{i,r+1} = \begin{cases} i, \text{ if } d^{r+1}_{i,r+1} = c_{i,r+1} \\ \text{or a } j \text{ attaining min. in (4.12) otherwise} \end{cases} \quad (4.15)$$

$$L^{r+1}_{r+1,i} = \begin{cases} r+1, \text{ if } d^{r+1}_{r+1,i} = c_{r+1,i} \\ \text{or a } j \text{ attaining min. in (4.13) otherwise} \end{cases} \quad (4.16)$$

Counting the self-loop at node $r+1$ of length 0 as a simple circuit, $d^{r+1}_{r+1,r+1}$ is the length of the shortest simple circuit containing node $r+1$ in the partial network induced by the subset of nodes $\{ 1, \ldots, r+1 \}$. If $d^{r+1}_{r+1,r+1} < 0$, let p be a j attaining the min.in (4.14). Let $\mathcal{C}_1$ be the chain consisting of only the arc $(r+1, p)$ if $d^{r+1}_{r+1,p} = c_{r+1,p}$; otherwise it consists of the arc $(r+1, q)$ and the chain from q to p determined by the label matrix at the end of Step r, where $q = L^{r+1}_{r+1,p}$. Similarly, let $\mathcal{C}_2$ be the chain consisting of only the arc $(p, r+1)$ if $d^{r+1}_{p,r+1} = c_{p,r+1}$; otherwise it consists of the chain from p to u determined by the label matrix at the end of Step r, and the arc $(u, r+1)$, where $u = L^{r+1}_{p,r+1}$. Then the simple circuit $\vec{\mathbb{C}}_1$ obtained by combining the chains $\mathcal{C}_1$ and $\mathcal{C}_2$ is a negative length circuit of length $d^{r+1}_{r+1,r+1}$; terminate.

If $d^{r+1}_{r+1,r+1} = 0$, define $L^{r+1}_{r+1,r+1} = r+1$. For $i, j = 1$ to r, define

$$d^{r+1}_{i,j} = \text{min. } \{d^r_{ij}; d^{r+1}_{i,r+1} + d^{r+1}_{r+1,j}\} \tag{4.17}$$

$$L^{r+1}_{ij} = \begin{cases} L^r_{ij}, \text{ if } d^{r+1}_{i,j} = d^{r+1}_{i,j} \\ \\ L^{r+1}_{r+1,j}, \text{ otherwise} \end{cases}$$

$(L^{r+1}_{ij} : i, j = 1 \text{ to } r+1)$, $(d^{r+1}_{ij} : i, j = 1 \text{ to } r+1)$ define the new label and distance matrices respectively. If $r+1 < n$ go to the next step; otherwise terminate.

Discussion

Consider the label and distance matrices at the end of Step r. If no negative length circuits have been detected up to this stage, the following facts (i), (ii) hold by the manner in which the label and distance matrices are updated during the algorithm. (iii) follows by using induction, it establishes the validity of the algorithm.

(i) For all i, j, h between 1 to r,

$$d^r_{i,h} \leqq d^r_{ij} + d^r_{jh} \tag{4.18}$$

i.e., the entries in the distance matrix satisfy the **triangle inequality**.

(ii) If node j appears on the chain from i to h traced by the labels in the label matrix, (4.18) holds as a strict equation.

(iii) For all i, j between 1 to r the chain from i to j traced by the labels in the label matrix is a shortest chain in the partial network induced by the subset of nodes $\{1, \ldots, r\}$, and d^r_{ij} is its length.

The computational effort in Step $r+1$ can be verified to be $O((r+1)^{\mathbf{2}})$. Hence, the overall computational effort in this algorithm is $O(\sum(r+1)^{\mathbf{2}}) = O(n^{\mathbf{3}})$.

The Floyd-Warshall Algorithm for All Shortest Chains

Let $G = (\mathcal{N}, \mathcal{A}, c)$ be the directed network in which the problem is being solved. Number the nodes in G serially 1 to $n = |\mathcal{N}|$. On any simple chain, nodes different from the initial and terminal nodes are called **intermediate nodes**. A simple chain has no intermediate nodes iff it consists of a single arc.

This algorithm maintains label and distance matrices of order $n \times n$ throughout, they store the current chains and their lengths respectively. The algorithm terminates after at most n steps. The distance matrix obtained at the end of Step r is denoted by (d^r_{ij}), where d^r_{ij} will be equal to the length of a shortest chain from node i to node j among those chains satisfying the condition that every intermediate node on them is from the set $\{1, \ldots, r\}$ (i, j themselves may or may not be from this set). The label matrix at that stage storing these constrained shortest chains is denoted by (L^r_{ij}). The main computational tool used repeatedly in this algorithm is called **the triangle (*or* triple) operation** given in (4.19) for nodes i, j and fixed node $r+1$.

FLOYD-WARSHALL ALGORITHM

Initialization Define the initial label and distance matrices to be $(L^0_{ij}), (d^0_{ij})$ respectively, where, for all i, j, $L^0_{ij} = i$, and $d^0_{ij} = c_{ij}$ for $i \neq j$, 0 if $i = j$. Go to Step 1.

General Step $r+1$ **for** $r \geqq 0$ At this stage we have the matrices $(L_{ij}^r), (d_{ij}^r)$. For each $i, j \in \mathcal{N}$ compute

$$d_{ij}^{r+1} = \text{min. } \{d_{ij}^r, d_{i,r+1}^r + d_{r+1,j}^r\} \tag{4.19}$$

$$L_{ij}^{r+1} = L_{ij}^r \text{ if } d_{ij}^{r+1} = d_{ij}^r; L_{r+1,j}^r \text{ otherwise}$$

$(L_{ij}^{r+1}), (d_{ij}^{r+1})$ are the new label and distance matrices. If $d_{ii}^{r+1} < 0$ for any $i \in \mathcal{N}$, the circuit obtained by putting the present chain from i to $r+1$ together with that from $r+1$ to i, is a negative length circuit, terminate. If $d_{ii}^{r+1} = 0$ for all i, and $r+1 = n$, the present chains are the shortest chains; terminate. Otherwise, go to the next step.

Discussion

We will now show that if d_{ii}^r stayed $= 0$ for all $i \in \mathcal{N}$ and $r = 1$ to n, then (d_{ij}^n) is the matrix of shortest chain lengths in G. For this we set up an induction hypothesis that (d_{ij}^r) is the matrix of shortest chain lengths subject to the constraint that every intermediate node on every chain is numbered $\leqq r$. This hypothesis clearly holds for $r = 0$, by initialization.

We will now prove that under the induction hypothesis, the statement in it also remains valid when r is replaced by $r+1$. For any $i, j \in \mathcal{N}$, a shortest chain from i to j with intermediate nodes numbered $\leqq r+1$, either does not contain node $r+1$ (in this case we must have $d_{ij}^{r+1} = d_{ij}^r$ by the triple operations carried out in Step $r+1$), or it contains node $r+1$. In the latter case, since $d_{i,r+1}^r, d_{r+1,j}^r$ are both shortest chain distances with intermediate nodes numbered $\leqq r$ by the induction hypothesis, $d_{i,r+1}^r + d_{r+1,j}^r$ is the shortest chain distance from i to j with intermediate nodes numbered $\leqq r+1$. Hence in either case, the statement in the induction hypothesis is also valid for $r+1$.

Hence, by induction, (d_{ij}^n) is the matrix of shortest chain lengths in G in this case.

In each step there are n^2 additions and n^2 comparisons to be performed. Hence the overall computational complexity of this method is $O(n^3)$.

4.6 Sensitivity Analysis in the Shortest Chain Problem

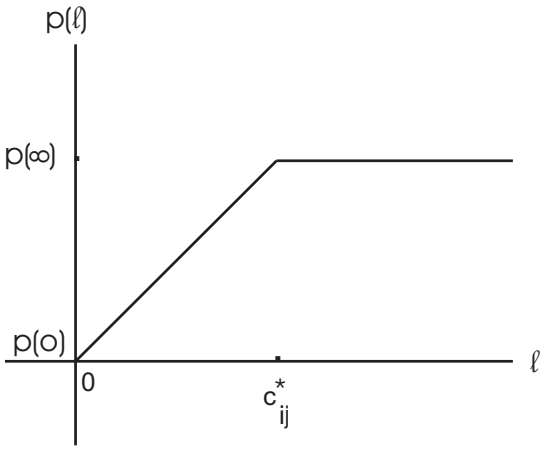

Figure 4.13: Node 1 is the origin.

Consider the problem of finding a shortest chain from node 1 to node n in the directed network $G = (\mathcal{N}, \mathcal{A}, c)$. Let (i, j) be a particular arc in $\mathcal{A}$ whose length c_{ij} is a parameter ξ which can vary, while all the other data remains unchanged. Let $\pi(\xi)$ denote the length of a shortest chain from 1 to n in G as a function of ξ.

If (i, j) is not contained on a shortest chain from 1 to n when $\xi = 0$, then $\pi(\xi) = \pi(0)$ for every $\xi \geqq 0$, and the same chain not containing (i, j) remains a shortest for all $\xi \geqq 0$.

If (i, j) is contained on a shortest chain from 1 to n when $\xi = 0$, this chain remains a shortest one as ξ increases from 0 to $\pi(\infty) - \pi(0)$. If $\xi > \pi(\infty) - \pi(0)$, the shortest chain changes, and (i, j) is not contained on any shortest chain from 1 to n in this range.

So, $\pi(\xi) = \text{min. } \{ \pi(0) + \xi, \pi(\infty) \}$ for all $\xi \geq 0$. See Figure 4.13. The quantity $\pi(\infty) - \pi(0)$ where $\pi(\xi)$ changes slope, is called the **critical length of arc** (i, j), and denoted by c^*_{ij}. Destroying an arc in a network is equivalent to making its length ∞. If arc (i, j) with present length c_{ij} and critical length c^*_{ij} is destroyed, the length of the shortest chain from 1 to n goes up by max. $\{0, c^*_{ij} - c_{ij}\}$.

Exercises

4.11 Find the shortest chains between all pairs of nodes in the mixed network in Figure 4.14. The number on each line is its length.

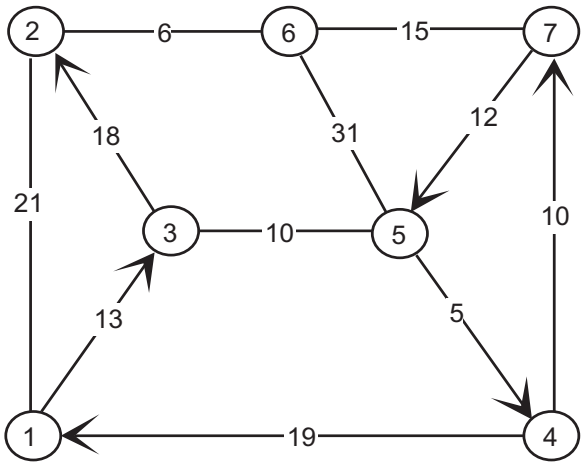

Figure 4.14: Node 1 is the origin.

4.12 Develop an efficient scheme for finding an arc in G, the destruction of which increases the length of the shortest chain from the origin to the destination by as much as possible.

4.13 Suppose G is the street network in a city, with nodes representing street intersections, and arcs representing street segments joining pairs of adjacent nodes. The length of each arc is the driving time

through the corresponding segment under normal conditions. By making improvements on an arc, its length can be reduced. Discuss how to pick a single arc for making improvements, to reduce the length of the shortest chain from the origin to the destination by as much as possible.

4.7 Algorithm for Ranking Chains in Nondecreasing Order of Length

Consider the directed network G = $(\mathcal{N}, \mathcal{A}, c)$ with $|\mathcal{N}| = n, |\mathcal{A}| = m$, and nodes 1, n as the origin and destination respectively. We assume that there are no negative length circuits in G. Here we discuss an adaptation of the ranking approach discussed in Section 3.6 to rank the chains from 1 to n in G in nondecreasing order of length, due to Lawler [1972]. Let **S** = set of all chains from 1 to n in G, simple or not; and **P** = set of all simple chains from 1 to n in G. The algorithm can rank either the chains in **S**, or only those in **P**. It obtains one new chain in the ranked sequence per step with a computational effort of at most $O(n^3)$.

We define a **special subset** to be the set of chains from 1 to n in G (from **S** or **P** as desired) containing all the arcs in a specified **initial segment** which is a chain from 1 to some node, and not containing any arc from a specified subset of **excluded arcs**. If the specified initial segment is $\mathcal{C}_1$ = 1, (1, i_1), i_1, ..., (i_p, r), r; and the specified subset of excluded arcs is **E**, we denote the special subset by the symbol **F** = $[\mathcal{C}_1; \bar{\mathbf{E}}]$. It contains all chains (from **S** or **P** as desired) obtained by combining $\mathcal{C}_1$ with a chain from r to n in G not containing any arc in **E**.

A shortest chain in a special subset **F** = $[\mathcal{C}_1; \bar{\mathbf{E}}]$, where $\mathcal{C}_1$ = 1, (1, i_1), i_1, ..., (i_p, r), r; from **S** or **P**,can be found efficiently. Let $\tilde{\text{G}}$ be the network obtained by deleting each of the arcs in the set **E** from $\mathcal{A}$. Let $\hat{\text{G}}$ be the network obtained by deleting each of the arcs incident at

the nodes 1, $i_1, \ldots, i_p$ and all the arcs incident into node r from $\tilde{G}$. For ranking the chains in **S**, find a shortest chain from r to n in $\tilde{G}$ by any method discussed earlier. If there is no chain from r to n in $\tilde{G}$, $\mathbf{F} = \emptyset$. If a shortest chain from r to n in $\tilde{G}$ is found, add it at the end of the specified initial segment $\mathcal{C}_1$ to get a shortest chain from 1 to n in **F** from **S**. For ranking the chains in **P**, carry out the same procedure replacing $\tilde{G}$ by $\hat{G}$, to get a shortest chain in **F** from **P**. The important thing is that the shortest chain from r to n, found in either of these cases, is a simple chain (since the algorithms discussed earlier deal only with simple chains) and hence consists of at most $n-1$ arcs.

The algorithm uses an operation called **partitioning a special subset using a shortest chain in it**. Let $\mathbf{F} = [1, (1, i_1), i_1, \ldots, (i_p, r), r ; \bar{\mathbf{E}}]$ be the special subset, and $\mathcal{C}_2 = 1, (1, i_1), i_1, \ldots, (i_p, r), r, (r, i_{p+1}), i_{p+1}, \ldots, (i_{p+h}, n), n$, be a shortest chain in **F** obtained as above. As mentioned above, $h+1$, the number of arcs on $\mathcal{C}_2$ between the nodes r to n is at most $n-1$. Partitioning **F** using $\mathcal{C}_2$ expresses $\mathbf{F} \setminus \{\mathcal{C}_2\}$ as a disjoint union of $h+1$ special subsets. These special subsets are

$$
\begin{aligned}
\mathbf{F}_1 &= [1, (1, i_1), \ldots, (i_p, r), r; \overline{\mathbf{E} \cup \{(r, i_{p+1})\}}] \\
\mathbf{F}_2 &= [1, (1, i_1), \ldots, (i_p, r), r, (r, i_{p+1}), i_{p+1}; \overline{\mathbf{E} \cup \{(i_{p+1}, i_{p+2})\}}] \\
&\vdots \\
\mathbf{F}_{h+1} &= [1, (1, i_1), \ldots, (i_p, r), r, (r, i_{p+1}), \ldots, (i_{p+h-1}, i_{p+h}), i_{p+h}; \overline{\mathbf{E} \cup \{(i_{p+h}, n)\}}]
\end{aligned}
$$

THE RANKING ALGORITHM

Step 1 Find a shortest chain in G from 1 to n using any of the algorithms discussed earlier. If there is no chain from 1 to n, terminate. Otherwise, by our assumption that there is no negative length circuit in G, a shortest simple chain will be obtained; let it be $\mathcal{C}^1 = 1, (1, j_1), j_1, \ldots, (j_u, n), n$. We would have obtained a shortest chain tree rooted at node 1. Nodes not on this tree cannot be reached from 1 by a chain, so eliminate them and all the arcs incident at them from further consideration. Let π_i be

the length of the chain from 1 to i in the shortest chain tree for remaining nodes i. For each remaining arc (i, j), we replace c_{ij} by $c_{ij} - (\pi_j - \pi_i) \geqq 0$, this has the effect of subtracting the same constant $(\pi_n - \pi_1)$ from the length of every chain from 1 to n and hence does not affect the ranking of these chains. With this change the arc length vector becomes nonnegative, even though the original vector may not be, and the efficient Dijkstra's method can be used to compute shortest chains in subsequent steps of the algorithm. Generate the following $u+1$ ($\leqq n-1$) special subsets.

$$
\begin{aligned}
&[\overline{\{(1, j_1)\}}] \\
&[1, (1, j_1), j_1; \overline{\{(j_1, j_2)\}}] \\
&\vdots \\
&[1, (1, j_1), \ldots, (j_{u-1}, j_u), j_u; \overline{\{(j_u, n)\}}]
\end{aligned}
$$

Find a shortest chain in each, and discard any special subsets among these which are empty. Arrange the remaining special subsets together with the shortest chain in each, in a list, in increasing order of the length of the shortest chain from 1 to n in it, top to bottom. Go to the next step.

General step p+1, for $p \geqq 1$ If the list is empty, there are no more chains from 1 to n in G; terminate. Otherwise, delete the topmost special subset, say $\mathbf{F}$; from the list. The shortest chain in $\mathbf{F}$, $\mathcal{C}^{p+1}$, say, is the next, i.e., $(p+1)$th chain in the ranked sequence. If you have enough chains in the ranked sequence already, terminate. Otherwise, partition $\mathbf{F}$ using $\mathcal{C}^{p+1}$. Find a shortest chain from 1 to n in each of the special subsets generated in this partitioning. Discard any empty ones among these, but include the nonempty ones together with the shortest chain in each, in the list in their proper position according to the length of the shortest chain from 1 to n in it. Go to the next step.

Discussion

At the end of Step 1, the list has at most $(n-1)$ special subsets. In each step, the top special subset is taken out, and at most $(n-1)$ newly generated special subsets are added to the list. Hence there will be at most $p(n-1)$ special subsets in the list at the end of Step p. Hence, the size of the list grows linearly with the number of chains ranked.

In each step the shortest chains in at most $(n-1)$ new special subsets need to be computed, each with a computational effort of at most $O(n^2)$, as pointed out above. Hence the computational effort in each step is at most $O(n^3)$, plus the effort needed to insert the new special subsets in the list. Thus the overall effort needed to find p chains in the ranked sequence is at most $O(pn^3)$.

4.8 Exercises

4.14 0-1 Knapsack Problem It is required to determine a subset of n available objects to load into a knapsack to maximize the value loaded subject to the knapsack's weight capacity constraint. The jth object has weight w_j kg. and value $\$v_j$, $j = 1$ to n, and the knapsack's capacity by weight is w_0 kg. All data are positive integers and $w_j \leqq w_0$ for all $j = 1$ to n. Objects cannot be broken; they have to be either loaded whole or left out.

Let G be a network in which the nodes are integer points $[p, q]$ in $\mathbb{R}^2$ for $0 \leqq p \leqq w_0$ and $1 \leqq q \leqq n$, and a sink node $\breve{t}$. The arcs in G are: ($[w_0,1]$, $[w_0, 2]$) with length 0, ($[w_0, 1]$, $[w_0 - w_1,2]$) with length v_1; for $2 \leqq q \leqq n-1$, ($[p, q]$, $[p, q+1]$) with length 0 for $0 \leqq p \leqq w_0$, ($[p, q]$, $[p - w_q, q+1]$) with length v_q for $w_q \leqq p \leqq w_0$; and ($[p, n]$, $\breve{t}$) with length 0 for $0 \leqq p \leqq w_0$.

Show that G is acyclic and that the knapsack problem is equivalent to that of finding a longest chain from $[w_0, 1]$ to $\breve{t}$ in G.

4.15 We have already seen that the problem of finding a longest simple chain from node 1 to node n in a directed network $G = (\mathcal{N}, \mathcal{A}, c)$ with $c > 0$ is a hard problem in general. Using this show that if $a \geqq 0, b > 0$ are two edge cost vectors in G, the problem of finding a simple chain $\mathcal{C}$ from 1 to n that minimizes the ratio objective function

$(\sum(a_{ij} :$ over $(i,j) \in \mathcal{C}))/\ (\sum(b_{ij} :$ over $(i,j) \in \mathcal{C}))$ is a hard problem (Ahuja, Batra, and Gupta [1983]).

4.16 Relationship between Assignment and Shortest Chain Problems Consider the problem of finding a shortest chain from 1 to n in a directed connected network $\mathrm{G} = (\mathcal{N}, \mathcal{A}, c)$. Define the $n \times n$ matrix $D = (d_{ij})$, where $d_{ij} = c_{ij}$ if $(i,j) \in \mathcal{A}$, $= 0$ if $i = j$, and $= \infty$ if $i \neq j$ and $(i,j) \notin \mathcal{A}$.

Find a minimum cost assignment with D as the cost matrix. Prove that the unit matrix is a minimum cost assignment in this problem iff there exist no negative length circuits in G. Conversely show that if the unit matrix is not optimal to this problem, then a negative length circuit in G can be identified from an optimum assignment.

Suppose there are no negative length circuits in G. Change d_{n1} to $-M$ where M is a very large positive number, and find a minimum cost assignment wrt the modified matrix D. If there exists no chain from 1 to n in G prove that the unit matrix is a minimum cost assignment for this problem and conversely. Otherwise, the allocations in a minimum cost assignment for this problem correspond to a circuit in G containing the arc $(n,1)$ and a shortest chain from 1 to n, and self-loops at the remaining nodes (Weintraub [1973]).

4.17 Suppose a shortest chain from node 1 to node n in the directed connected network $\mathrm{G} = (\mathcal{N}, \mathcal{A}, c)$ without any self-loops or negative length circuits, has been found using an algorithm for the assignment problem with the approach discussed in Exercise 4.16. From this information, if any of the following types of changes occur in G, show that a shortest chain from 1 to n in the modified network can be obtained with a computational effort of at most $\mathrm{O}(n^{\mathbf{2}})$: (a) new arcs are added, all incident at an existing node in G, (b) a subset of arcs incident at a node in G are deleted, (c) a node and all the arcs incident at it in G are deleted, (d) a new node is added to G together with some new arcs incident at it, (e) the lengths of arcs incident at a node have been modified (Weintraub [1973]).

4.18 Negative Length Simple Cycles in Undirected Networks Let $\bar{\mathbf{E}}$ be the set of negative length edges in an undirected network G

$= (\mathcal{N}, \mathcal{A}, c = (c_{i;j}))$ with c as the vector of edge lengths. Assume that $\bar{\mathbf{E}} \neq \emptyset$, and let $\bar{\mathcal{N}}$ be the set of nodes on edges in $\bar{\mathbf{E}}$. If there is a simple cycle in $(\bar{\mathcal{N}}, \bar{\mathbf{E}})$, it is a negative length simple cycle in G. So, assume that there is no simple cycle in $(\bar{\mathcal{N}}, \bar{\mathbf{E}})$; i.e., it is a forest.

Let $\bar{\mathcal{N}}_o$ be the set of odd degree nodes in $(\bar{\mathcal{N}}, \bar{\mathbf{E}})$. Construct the complete undirected network H $= (\bar{\mathcal{N}}_o, A, d = (d_{ij}))$, where $A = \{ (i;j) : i \neq j, i, j \in \bar{\mathcal{N}}_o \}$, and d is the vector of edge lengths in H with d_{ij} being the length of a shortest path between i and j, $\mathcal{P}_{ij}$, in $(\mathcal{N}, \mathcal{A})$

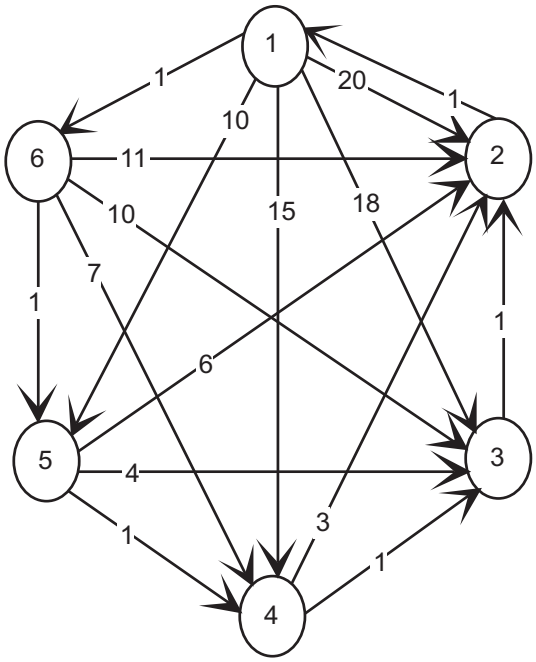

Figure 4.15:

with $(|c_{ij}|)$ as the vector of edge lengths. Let $\bar{\mathbf{M}}$ be a minimum cost perfect matching in H, and let P($\bar{\mathbf{M}}$) be the union of the sets of edges on the paths $\mathcal{P}_{ij}$ for $(i;j) \in \bar{\mathbf{M}}$.

Prove that G contains a nonpositive length simple cycle if $\bar{\mathbf{E}} \not\subset$ P($\bar{\mathbf{M}}$). Prove that G contains no negative length simple cycles if $\bar{\mathbf{E}} \subset$ P($\bar{\mathbf{M}}$). Using these results develop an efficient algorithm for detecting a

negative length simple cycle in G if one exists. Also, develop an efficient algorithm for finding shortest paths between every pair of nodes in G assuming that there are no negative length simple cycles in G (Tobin [1975]).

4.19 Find the shortest chain tree rooted at node 1 in the network in Figure 4.15 using the LIFO sequence-list driven labeling methods discussed in Exercise 4.10. Data on each arc is its length.

In general consider the following sequence of networks of which the

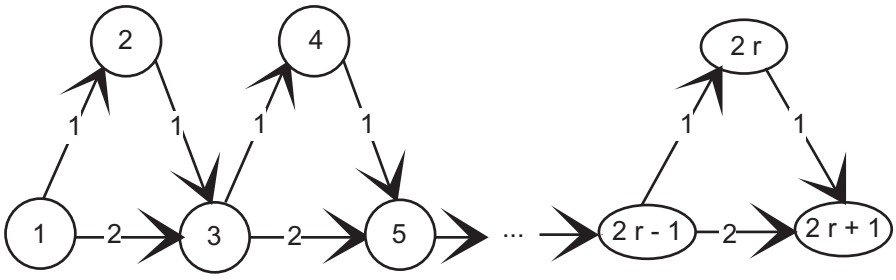

Figure 4.16:

one in Figure 4.15 is the fifth. The first network consists of two nodes 1, 2, and two arcs (1, 2) and (2, 1) each with length 1. To get the $(r+1)$th network from the rth do the following: Add node $r+2$. Add arcs $(r+2, i)$ for $i = 2$ to $r+1$ with length $c_{r+2,i}$ same as $c_{1,i}$ in the rth network. Change $c_{1,i}$ to $c_{1,i} + 2^{\mathbf{r}-1} + 1$ for $i = 2$ to $r+1$. Add an arc $(1, r+2)$ with length 1. Show that node 2 will be branched out $2^{\mathbf{r}-2}$ times in these methods on the rth network (Kershenbaum [1981]).

4.20 G $= (\mathcal{N}, \mathcal{A})$ is a given directed network. Given three nodes s, t, p in G, show that the problem of checking whether there exists a simple chain from s to t via p is an NP-complete problem.

Given nodes s, t in G, show that the problem of checking whether there exists a simple chain from s to t consisting of an even number of arcs is an NP-complete problem (Lapaugh and Papadimitriou [1984]).

4.21 G $= (\mathcal{N}, \mathcal{A}, c)$ is a directed connected network with $c \geqq 0$. **X, Y** are respectively the sets of permanently labeled and temporarily

labeled nodes at some stage in the process of finding the shortest chain tree rooted at node 1 in G by Dijkstra's algorithm. Let μ_i be the present distance index on node i. For every $j \in \mathbf{X}$, prove that min. $\{\mu_i : i \in \mathbf{Y}\} \geqq \mu_j$. Hence show that the distance index of a node at the time that it gets permanently labeled is $\geqq$ the distance indices of nodes already in $\mathbf{X}$ at that time.

4.22 In the network in Figure 4.16 with $2r + 1$ nodes and $3r$ arcs, show that every one of the $2^{\mathbf{r}}$ simple chains from node 1 to node $2r+1$ is a shortest chain.

4.23 A Constrained Shortest Chain Problem An air carrier is trying to cover r flights with m planes which is known to be inadequate. a_i, b_i are the scheduled start time (at origin), and finish time (at destination) of ith flight, $i = 1$ to r. Construct a directed network G with nodes $L_1, \ldots, L_r$ for flights, and $P_1, \ldots, P_m$ for planes. Include arcs (L_i, P_j) and (P_j, L_i) of length 0 for each i, j. For each i, j such that the destination of L_i and the origin of L_j are the same, include an arc (L_i, L_j) of length equal to the delay that occurs for flight L_j if the same aircraft performs L_j after L_i.

Show that any chain in G passing through every node once corresponds to an assignment of planes to flights; and that the length of such a chain is exactly the total delay in all the flights corresponding to that assignment. Hence show that the problem of covering all the r flights with m planes, to minimize total delay, is the problem of finding a shortest chain in G subject to the constraint that it must pass through each node exactly once (Teodorovic and Guberinic [1984]).

4.24 Application to Set Partitioning Problem The following are three important 0-1 integer programming models, in which the data consists of a 0-1 integer matrix $A = (a_{ij})$ of order $m \times n$ and a column vector $\mathbf{e}$ of all 1's.

Set partitioning problem			Set covering problem			Set packing problem		
min.	cx		min.	cx		min.	cx	
s. to	Ax	$= \mathbf{e}$	s. to	Ax	$\geqq \mathbf{e}$	s. to	Ax	$\leqq \mathbf{e}$
all	x_j	$= 0$ or 1	all	x_j	$= 0$ or 1	all	x_j	$= 0$ or 1

Consider the special case of the set partitioning problem in which each column of A consists of a single consecutive segment of ones, i.e., for each $j = 1$ to n there exists $g_j \leqq h_j$ such that $a_{ij} = 1$ for $g_j \leqq i \leqq h_j$ and 0 for all other i. Such problems arise in crew scheduling applications where each crew must do a single stretch of duty comprising a consecutive set of trips, known as one-part duty crew scheduling problems. Construct the network $G = (\mathcal{N}, \mathcal{A})$ where $\mathcal{N} = \{1, \ldots, m+1\}$ with node i corresponding to row i of A for $1 \leqq i \leqq m$, and $\mathcal{A}$ consists of arcs $(g_j, h_j + 1)$ of length c_j corresponding to $A_{.j}$ for $j = 1$ to n.

Show that G is acyclic, and that this special case of the set partitioning problem is equivalent to the problem of finding a shortest chain in G from 1 to $m + 1$.

A set partitioning problem is infeasible when an exact cover does not exist, but in practical applications relaxations that **overcover** or **undercover** with a possible penalty expense are typically permitted. So, consider the following extended set partitioning problem, where $(p_i, q_i) \geqq 0$ for all i. Show that when A satisfies the special property mentioned above, this extended set partitioning problem can also be formulated as a shortest chain problem in a network. Develop this formulation.

$$\begin{aligned}
\text{min.} \quad & \sum_j c_j \quad x_j \quad + \sum_i (p_i u_i + q_i \sigma_i) \\
\text{s. to} \quad & \sum_j a_{ij} \quad x_j \quad + u_i - \sigma_i = 1, i = 1 \text{ to } m \\
& \qquad\qquad x_j \quad , u_i = 0 \text{ or } 1 \text{ for all } i, j \\
& \qquad\qquad \sigma_i \quad \geqq 0 \text{ and integer for all } i
\end{aligned}$$

When A satisfies the special property mentioned above, both the set covering and the set packing problems can also be formulated as short chain problems. Derive these formulations (Darby-Dowman and Mitra [1985], and Shepardson and Marsten [1980]).

4.25 Manpower Planning A construction company requires the following number of steel erectors over a 6-month horizon.

Month	March	April	May	June	July	August
Erectors needed	4	6	7	4	6	2

In February there were 3 steel erectors on site, and in September this number has to be the same. It costs \$1000 to hire an erector, and \$1600 to terminate the services of one. Each erector's contract is decided on a month-to-month basis at the start of each month. No more than 3 erectors can be hired anew at the start of any month. Under a union agreement no more than a third of the current manpower can be terminated at the end of a month. The cost of keeping a surplus erector is \$3000 per month. When there is a shortage, it has to be made up in overtime; this costs \$6000 per erector per month. Overtime cannot exceed 25% of normal time. It is required to determine how many erectors to hire or terminate each month in order to minimize the total cost of hiring, terminating, surplus and shortage costs over the horizon, subject to the given constraints. Formulate this as a shortest chain problem in an acyclic network and find an optimum solution (Clark and Hastings [1977]).

4.26 Replacement Problem This problem arises in planning over a 5-year horizon at a plant. They have a machine which will be 3 years old at the beginning of the first year of the horizon. The machine is inspected at the beginning of each year and is either overhauled or replaced with a new machine. The cost of a new machine is \$40,000. The cost of overhaul and the scrap value of the machine depend on its age as given below.

Age (years)	1	2	3	4
Overhaul cost (\$1000 units)	15	6	18	
Scrap value (\$1000 units)	20	10	5	2

Company's policy is that the machine must be replaced at age 4 years. Also, assume that at the end of the 5-year horizon, the then current machine will be scrapped.

It is required to find the policy that minimizes the present value of the cost of overhauling or replacing the machine, when the costs are discounted at the rate of 15% per year. Formulate this as a shortest

chain problem in an acyclic network and find an optimum solution (Clark and Hastings [1977]).

4.27 Detecting Negative Cost Circuits by Node Elimination Consider the problem of finding negative cost circuits in the directed connected network G = $(\mathcal{N}, \mathcal{A}, c)$. $\mathcal{A}$ may contain some self-loops. Look for negative cost self-loops. If

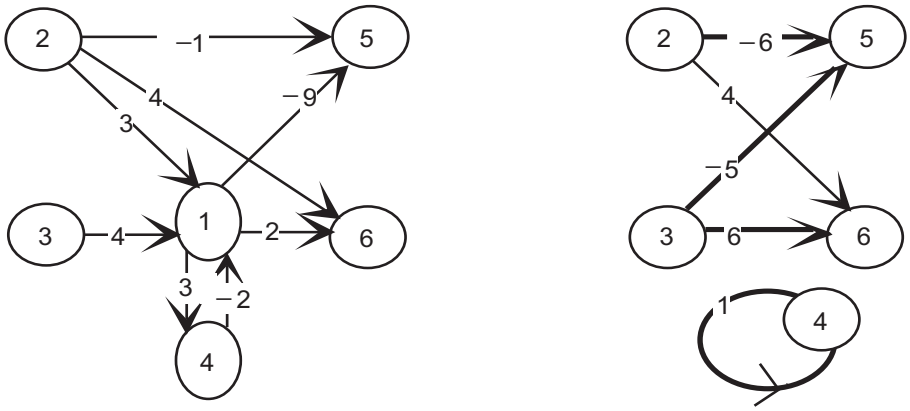

Figure 4.17: Elimination of node 1.

there are none, or if self-loops are not to be considered as circuits, eliminate all self-loops in G. Define $(i, j) \in \mathcal{A}$ as an **original arc** and associate it with the chain from i to j consisting of that arc only. During the algorithm, new arcs are created by coalescing chains in the original network into arcs. These will be called **created arcs** ; each of them is associated with a chain in G. Some of the created arcs may be self-loops; each of them is associated with a circuit in G.

Select a node, say 1. If $\mathbf{A}_1$ or $\mathbf{B}_1$ is empty, 1 is not on any circuit; eliminate it and all arcs incident at it. If both $\mathbf{A}_1, \mathbf{B}_1$ are nonempty, for each $i \in \mathbf{A}_1, j \in \mathbf{B}_1$ do the following: If (i, j) does not exist now, introduce the created arc (i, j) with cost coefficient $c_{i1} + c_{1i}$, associated with the chain $i, (i, 1), 1, (1, j), j$ in G (if $i = j$ this created arc will be a self-loop at i and the chain in G associated with it is a circuit). If (i, j) already exists, and $c_{ij} \leq c_{i1} + c_{1j}$ leave it as it is. If $c_{ij} > c_{i1} + c_{1j}$,

change its cost coefficient to $c_{i1} + c_{1j}$ and change the chain associated with it to that from i to j obtained by combining the chains associated with $(i, 1)$ and $(1, i)$ in that order.

Now eliminate node 1 and all the arcs incident at it, and let the resulting network be G^1. See Figure 4.17 where the cost coefficients are entered on the arcs, and created arcs are thick.

The same process is repeated on the network G^1, and the process continued yielding networks $G^2, \ldots$, with decreasing number of nodes. Prove that G has a negative cost circuit iff G^r has either a negative cost self-loop or a negative cost circuit. So, if G has a negative cost circuit, it will be detected by one of the networks in the sequence having a negative cost self-loop. The circuit associated with such a self-loop is a negative cost circuit in G. Apply this approach to detect any negative cost circuits in the following networks in Figure 4.18 with cost coefficients entered on the arcs. Analyze the computational complexity of this algorithm (Chen [1975]).

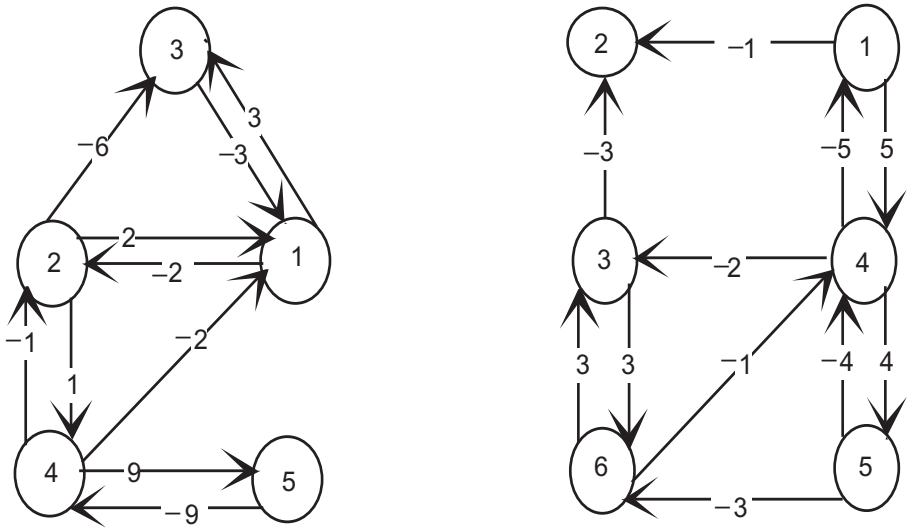

Figure 4.18:

4.28 Let $G = (\mathcal{N}, \mathcal{A})$ be an undirected network. For $i, j \in \mathcal{N}$ define

$d_{ij} = 0$ if $i = j$, $= \infty$ if there is no path from i to j in G, and = the number of edges on a path with the smallest number of edges between i and j in G otherwise. Develop efficient methods for computing the matrix (d_{ij}), and for updating this matrix when an edge or a node is added or deleted (Cheston and Corneil [1982]).

4.29 We consider a vehicle routing problem involving also fleet size and mix decisions. T is the number of vehicle types. a_r, f_r are respectively the capacity and the fixed cost of acquiring vehicle type r, $r =$

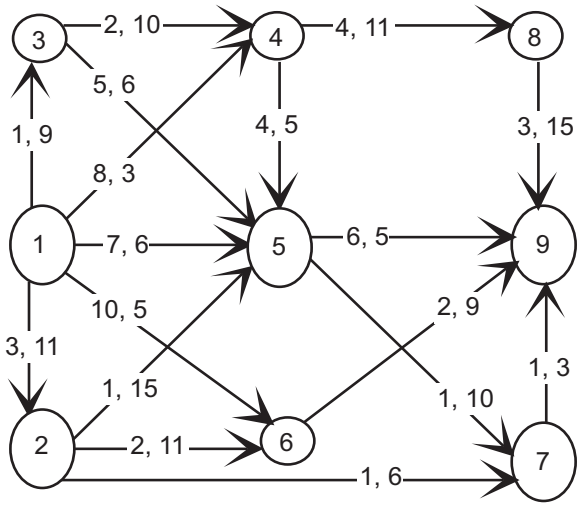

Figure 4.19: The entries on arc (i, j) are σ_{ij}, β_{ij}.

1 to T. This data satisfies $a_1 < a_2 < \ldots < a_T$, $f_1 < \ldots < f_T$. 0 denotes the depot, and 1 to n are the customers. All routes originate and terminate at the depot. c_{ij} is the cost of travel from i (depot or customer) to j, assumed to be symmetric and independent of vehicle type used. The customers are ordered so that $c_{01} > c_{02} > \ldots > c_{0n}$. d_i is the demand at customer i, a positive integer, for $i = 1$ to n, and $D = \sum d_i$. Splitting of a customer's demand between two vehicles is allowed. Develop a method for obtaining a lower bound for the minimum total cost (sum of fixed costs of acquiring the vehicles and the variable costs of travel) for making deliveries to meet the customer's demands, by

solving a shortest chain problem on a directed network with $D + 1$ nodes (Golden, Assad, Levy, and Gheysens [1984]).

4.30 Minimum Bottleneck Cost Chains Let $G = (\mathcal{N}, \mathcal{A}, c, \breve{s}, \breve{t})$ be a directed network. Define the bottleneck cost of a chain $\mathcal{C}$ from $\breve{s}$ to $\breve{t}$ to be max. $\{c_{ij} : (i, j) \in \mathcal{C}\}$. Develop an efficient algorithm for finding a minimum bottleneck cost chain from $\breve{s}$ to $\breve{t}$ in G. What is its computational complexity? Develop a special procedure for solving this problem when G is an acyclic network, that exploits the acyclic structure.

4.31 Optimal Sum-Bottleneck (SB) Chains Let $G = (\mathcal{N}, \mathcal{A}, \breve{s}, \breve{t})$ be a directed network which is doubly weighted, i.e., each arc (i, j) has a **sum weight** σ_{ij}, and a **bottleneck weight** β_{ij} associated with it. The sum weight of a chain $\mathcal{C}$, $\sigma(\mathcal{C})$ is the sum of σ_{ij} over arcs (i, j) on it. The bottleneck weight of $\mathcal{C}$, $\beta(\mathcal{C})$, is the maximum of β_{ij} over arcs (i, j) on it. The SB weight of $\mathcal{C}$ is defined to be equal to max. $\{ \sigma(\mathcal{C}), \beta(\mathcal{C}) \}$. Develop an efficient algorithm for finding a minimum SB weight chain from $\breve{s}$ to $\breve{t}$ in G and derive its computational complexity. Apply this algorithm on the network in Figure 4.19 with $\breve{s} = 1$, and $\breve{t} = 9$ (Bokhari [1988]).

4.32 The Jogger's Problem Let $G = (\mathcal{N}, \mathcal{A})$ be a connected road network at the disposal of a jogger beginning his jog at node $1 \in \mathcal{N}$. Each line in the network possesses some positive measure of undesirability. The jogger's problem is to identify a simple circuit containing node 1 for his jog, of minimum total undesirability. The route is required to be a simple circuit, so it cannot consist of the back-and-forth traversal of a single edge. In the directed case it cannot just consist of a pair of arcs of the form (1, i), (i, 1); the requirement is that it must consist of 3 or more nodes.

If $\mathcal{C}_{1p}$ denotes the chain from 1 to p obtained when the last arc, $(p, 1)$ say, from an optimum simple circuit is deleted, show that $\mathcal{C}_{1p}$ must be a minimum undesirable chain from 1 to p not containing the arc $(1, p)$. Use this to develop an algorithm for the jogger's problem, and determine its computational complexity.

Now consider several different versions of the problem.

VERSION 1: Here G is undirected, and the undesirability of an edge is independent of the direction of travel. Let $\mathbb{T}$ be a shortest (i.e., min. cost) path tree rooted at 1 using the undesirability ratings as the edge cost coefficients. Prove that there exists an optimum jogger's route in which all but one of the edges are from $\mathbb{T}$, and the other, say (x, y), is such that x and y have no common ancestor in $\mathbb{T}$ other than 1. Use this to develop a more efficient algorithm than that discussed above for this version.

VERSION 2: Even for a two-way street, the undesirability measure may depend on the direction of travel (e.g., if there is a gradient); then it becomes necessary to represent it by a pair of arcs, one in each direction. In this case G becomes directed, and arcs $(i, j), (j, i)$ may exist with different undesirability measures. From the requirements of the route, at most one of these two arcs $(i, j), (j, i)$ can appear on the jog for any i, j. In this version prove that there is an optimum jogger's route which is a concatenation of the form $\mathcal{C}_{1x}, (x, y), \mathcal{C}_{y1}$ where $\mathcal{C}_{1x}$, $\mathcal{C}_{y1}$ are minimum undesirability chains from 1 to x, and y to 1 respectively. Using this develop an efficient algorithm in this case.

VERSION 3 : G is undirected as in Version 1. We are given the length of each edge in G in addition to its undesirability measure. There is a constraint that the total length on the circuit must be $\stackrel{\geq}{=}$ some specified quantity. Develop an efficient algorithm for the jogger's problem under this constraint.
(Bird [1981]).

4.33 G $= (\mathcal{N}, \mathcal{A})$ is an acyclic network in which a subset of nodes $\mathbf{X} \subset \mathcal{N}$ have weights w_i associated with them. It is required to find a chain from 1 to n in G such that the sum of the weights of the **X**-nodes belonging to the chain is maximum. Develop an efficient O($|\mathcal{A}|$) complexity algorithm for this problem (Kundu [1978]).

Comment 4.1 As the shortest chain problem (called shortest path problem in some books) is so fundamental for modeling distribution and routing applications, it has been a major focus of research in network optimization, and the literature on it is vast. Deo and Pang [1984] contains an extensive bibliography.

The first label setting algorithm is due to Dijkstra [1959], and independently by Dantzig [1960] and Whiting and Hillier [1960]. Its worst case computational complexity of $O(n^2)$ is the best possible running time in dense networks, since any algorithm for this problem must examine every arc. However, in sparse networks, it is possible to improve the performance of this algorithm through the use of appropriate data structures. Dial's implementation [1965] is very popular and gives excellent computational performance. For other improvements in Dijkstra's algorithm see Denardo and Fox [1979], Dial, Glover, Karney, and Klingman [1979], Fredman and Tarjan [1987], and Johnson [1973, 1977].

Label correcting methods were introduced by Ford [1956], Moore [1957] and Bellman [1958]. Many improvements in the basic algorithm are discussed in Gilsinn and Witzgall [1973], Glover, Glover, and Klingman [1984], Glover, Klingman, and Phillips [1985], Glover, Klingman, Phillips, and Schneider [1985], Goldfarb, Hao, and Kai [1989a, 1989b], Pape [1974], Shier and Witzgall [1981], and Yen [1970]. With these improvements, label correcting methods are very efficient in practice, particularly in sparse networks.

Computational studies comparing various shortest chain algorithms may be found in Denardo and Fox [1979], Dial, Glover, Karney, and Klingman [1979], Gallo and Pallatino [1988], Gilsinn and Witzgall [1973], Glover, Klingman, Phillips, and Schneider [1985], Imai and Iri [1984], Kelton and Law [1978], Pape [1974], and Van Vliet [1978].

4.9 References

R. K. AHUJA, J. L. BATRA, and S. K. GUPTA, May 1983, "Combinatorial Optimization With Rational Objective Functions: A Communication," *MOR*, 8, no. 2(314).

R. BELLMAN, 1958, "On a Routing Problem," *QAM*, 16(87-90).

R. S. BIRD, Dec. 1981, "The Jogger's Problem," *IPL*, 13, no.(114-117).

S. H. BOKHARI, Jan. 1988, "Partitioning Problems in Parallel, Pipelined and Distributed Computing," *IEEE Transactions on Computers* , 37, no. 1(48-57).

I-NGO CHEN, June 1975, "A node Elimination Method for Finding Negative Cycles in a Directed Graph," *INFOR*, 13, no. 2(147-158).

G. A. CHESTON and D. G. CORNEIL, Aug. 1982, "Graph Property Update Al-

gorithms and Their Application to Distance Matrices," *INFOR*, 20, no. 3(178-201).
N. CHRISTOFIDES, A. MINGOZZIA, and P. TOTH, 1981, "Exact Algorithms for the Vehicle Routing Problem, Based on Spanning Trees and Shortest Path Relaxations," *MP*, 20(255-282).
J. A. CLARK and N. A. J. HASTINGS, 1977, "Decision Networks," *ORQ*, 28, no. 1i(51-68).
G. B. DANTZIG, 1960, "On the Shortest Route Through a Network," *MS*, 6(187-190).
G. B. DANTZIG, 1967, "All Shortest Routes in a Graph," in P. Rosentiehl (Ed.), *Theory of Graphs*, Dunod, Paris(91-92).
K. DARBY-DOWMAN and G. MITRA, Aug. 1985, "An Extension of Set Partitioning With Applications to Scheduling Problems," *EJOR*, 21, no. 2(200-205).
E. DENARDO and B. FOX, 1979, "Shortest-route Methods; 1. Reaching, Pruning and Buckets," *OR*, 27(161-186).
N. DEO and C. - Y. PANG, 1984, "Shortest Path Algorithms: Taxonomy and Annotation," *Networks*, 14(275-323).
R. DIAL, 1965, "Algorithm 360: Shortest Path Forest With Topological Ordering," *CACM*, 12(632-633).
R. DIAL, F. GLOVER, D. KARNEY, and D. KLINGMAN, 1979, "A Computational Analysis of Alternative Algorithms and Labeling Techniques for Finding Shortest Path Trees," *Networks*, 9(215-248).
E. DIJKSTRA, 1959, "A Note on Two Problems in Connection With Graphs," *Numerische Mathematik*, 1(269-271).
P. DOULBIEZ and M. RAS, 1975, "Optimal Network Capacity Planning: A Shortest Path Scheme," *OR*, 23(810-818).
S. DREYFUS, 1969, "An Appraisal of Some Shortest-path Algorithms," *OR*, 17(395-412).
J. EDMONDS, 1970, "Exponential Growth of the Simplex Method for Shortest Path Problems," unpublished report, University of Waterloo.
M. FLORIAN, S. NGUYEN, and S. PALLOTINO, 1981, "A Dual Simplex Algorithm for Finding All Shortest Paths," *Networks*, 11(367-378).
R. W. FLOYD, 1962, "Algorithm 97, Shortest Path," *CACM*, 5(345).
L. R. FORD, Jr. 1956, "Network Flow Theory," Rand report P-923.
M. L. FREDMAN and R. E. TARJAN, 1987, "Fibonacci Heaps and Their Uses in Improved Network Optimization Algorithms," *JACM*, 34(596-615).
G. GALLO and S. PALLOTINO, 1988, "Shortest Path Algorithms," in B. Simeone, P. Toth, G. Gallo, F. Maffioli, and S. Pallotino (Eds.), *Fortran Codes for Network Optimization*, *AOR*, 13(3-79).
A. GARCIA-DIAZ and J. LIEBMAN, 1980, "An Investment Staging Model for a Bridge Replacement Problem," *OR*, 28(736-753).
J. GILSINN and C. WITZGALL, 1973, "A Performance Comparison of Labeling Algorithms for Calculating Shortest Path Trees," Technical note 772, NBS, Washington, D.C.
F. GLOVER, R. GLOVER, and D. KLINGMAN, 1984, "Computational Study of

an Improved Shortest Path Algorithm," *Networks*, 14, no. 1(25-36).
F. GLOVER, D. KLINGMAN, and N. PHILLIPS, 1985, "A New Polynomially Bounded Shortest Path Algorithm," *OR*, 33(65-73).
F. GLOVER, D. KLINGMAN, N. PHILLIPS, and R. F. SCHNEIDER, 1985, "New Polynomial Shortest Path Algorithms and Their Computational Attributes," *MS*, 31(1106-1128).
B. GOLDEN, A. ASSAD, L. LEVY, and F. GHEYSENS, 1984, "The Fleet Size and Mix Vehicle Routing Problems," *COR*, 11, no. 1(49-66).
B. GOLDEN and T. MAGNANTI, 1978, "Transportation Planning: Network Models and Their Implementation," *Studies in Operations Management*, North Holland, Amsterdam(365-518).
D. GOLDFARB, J. HAO, and S. R. KAI, July-Aug. 1990, "Efficient Shortest Path Simplex Algorithms," *OR*, 38, no. 4(79 - 91).
D. GOLDFARB, J. HAO, and S. R. KAI, 1991, "Shortest Path Algorithms Using Dynamic Breadth-First Search," *Networks*, 21(29-50).
H. IMAI and M. IRI, 1984, "Practical Efficiencies of Existing Shortest Path Algorithms and a New Bucket Algorithm," *Journal of the OR Society of Japan*, 27(43-58).
D. B. JOHNSON, July 1973, "A Note on Dijkstra's Shortest Path Algorithm," *JACM*, 20(385-388).
D. B. JOHNSON, Jan. 1977, "Efficient Algorithms for Shortest Paths in Sparse Networks," *JACM*, 24, no. 1(1-13).
W. D. KELTON and A. M. LAW, 1978, "A Mean-time Comparison of Algorithms for all Pairs Shortest Path Problem With Arbitrary Arc Lengths," *Networks*, 8(97-106).
A. KERSHENBAUM, 1981, "A Note on Finding Shortest Path Trees," *Networks*, 11, no. 4,(399-400).
M. KLEIN and R. K. TIBREWALA, Feb. 1973, "Finding Negative Cycles," *INFOR*, 11, no. 1(59-65).
S. KUNDU, Jan. 1978, "Note on a Constrained-path Problem in Program Testing," *IEEE Transactions on Software Engineering*, SE-4, no. 1(75-76).
A. S. LAPAUGH and C. H. PAPADIMITRIOU, 1984, "The Even Path Problem for Graphs and Digraphs," *Networks*, 14, no. 4(507-513).
E. L. LAWLER, Mar. 1972, "A Procedure for Computing the k Best Solutions to Discrete Optimization Problems and its Application to the Shortest Path," *MS*, 18(401-405).
K. MALIK, A. K. MITTAL, and S. K. GUPTA, Aug. 1989, "The k-most Vital Arcs in the Shortest Path Problem," *OR Letters*, 8, no. 4(223-227).
G. J. MINTY, 1958, "A Variant on the Shortest Route Problem," *OR*, 6(882-883).
E. F. MOORE, 1957, "The Shortest Path Through a Maze," in *Proc. of the International Symposium on the Theory of Switching, Part 2*; The Annals of the Computation Laboratory of Harvard Univ. 30(285-292).
U. PAPE, 1974, "Implementation and Efficiency of Moore-algorithms for the Shortest Route Problem," *MP*, 7(212-222).

A. R. PIERCE, 1975, "Bibliography on Algorithms for Shortest Path, Shortest Spanning Tree and Related Circuit Routing Problems," *Networks*, 5(129-143).
M. SCHWARTZ and T. STERN, 1980, "Routing Techniques Used in Computer Communication Networks," *IEEE Transactions on Comm.*, COM-28(539-552).
F. SHEPARDSON and R. E. MARSTEN, 1980, "A Lagrangian Relaxation Algorithm for the Two Duty Period Scheduling Problem," *MS*, 3(274-281).
D. SHIER and C. WITZGALL, 1981, "Properties of Labeling Methods for Determining Shortest Path Trees," *Journal of Research of the NBS*, 86(317-330).
D. TEODOROVIC and S. GUBERINIC, Feb. 1984, "Optimal Dispatching Strategy on an Airline Network After a Schedule Perturbation," *EJOR*, 15, no. 2(178-182).
R. L. TOBIN, 1975, "Minimal Complete Matchings and Negative Cycles," *Networks*, 5, no. 4(371-387).
D. VAN VLIET, 1978, "Improved Shortest Path Algorithms for Transport Networks," *Transportation Research*, 12(7-20).
S. WARSHALL, 1962, "A Theorem on Boolean Matrices," *JACM*, 9(11-12).
A. WEINTRAUB, 1973, "The Shortest and the k-shortest Routes as Assignment Problems," *Networks*, 3(61-73).
P. D. WHITTING and J. A. HILLIER, 1960, "A Method for Finding the Shortest Route Through a Road Network," *ORQ*, 11(37-40).
J. YEN, 1970, "An Algorithm for Finding Shortest Routes From All Source Nodes to a Given Destination in General Networks," *QAM*, 27(526-530).

Index

For each index entry we provide the page number where it is defined or discussed first.

Ancestor checking 343

Bellman-Ford eqs. 333
BFM method 347
Branching out 345

Critical length 363

DBFS algorithm 349
Distance matrix 357

Floyd-Warshall 360

Greedy 336
- Algorithm 336
- Selection 336

Inductive algorithm 358

Label 337
- Correcting method 342
- Depth index 349
- Matrix 357
- Permanent 337
- Setting method 336
- Temporary 337

Matrix methods 357

Next list 349

Primal method 343

Ranking algorithm 364

Sensitivity analysis 362
Shortest Chain problem 329
- Acyclic 355
- LP formulation 329
- Relation to assignment 368
- Set partition application 371

Shortest chain tree 329
Shortest path tree 329

Triangle inequality 360
Triangle operation 360
Triple operation 360